การเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยดีไซน์แพทเทิร์น

นางสาวนริศรา ธาระพุทธ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

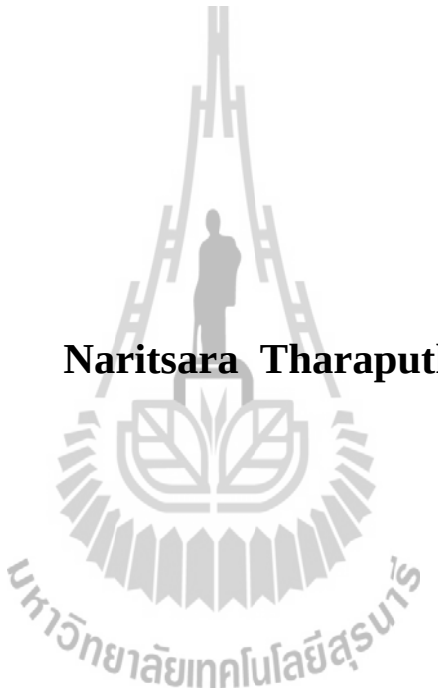
สาขาวิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ปีการศึกษา 2557

A COMPARISON AND EVALUATION OF THE DART LANGUAGE WITH DESIGN PATTERNS

Naritsara Tharaputh



A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2014

การเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยดีไซน์แพทเทิร์น

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

_____

(ผศ. ดร.พิชโยทัย มหัทธนาภิวัฒน์)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____

(ผศ. ดร.ชาญวิทย์ แก้วกสิ)
กรรมการ

_____

(ศ. ดร.ชูกิจ ลิมปิจำนงค์)
รองอธิการบดีฝ่ายวิชาการและนวัตกรรม

_____

(รศ. ร.อ. ดร.กนต์ธร ชำนิประศาสน์)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

นริศรา ธาระพุทธ : การเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยดีไซน์แพทเทิร์น (A COMPARISON AND EVALUATION OF THE DART LANGUAGE WITH DESIGN PATTERNS) อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์, 167 หน้า

การเปรียบเทียบและการประเมินผลภาษาเขียนโปรแกรมถือเป็นงานที่สำคัญงานหนึ่ง วัตถุประสงค์ของการเปรียบเทียบและการประเมินผลจะขึ้นอยู่กับผู้ที่ทำการศึกษาเช่น เพื่อช่วยให้นักพัฒนาสามารถลดระยะเวลาในการศึกษาภาษานั้น ๆ และเพื่อสนับสนุนการตัดสินใจในการเลือกภาษาที่จะนำไปใช้ในการพัฒนาซอฟต์แวร์ งานวิจัยนี้มีวัตถุประสงค์หลักในการเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยตัวอย่างการใช้งานดีไซน์แพทเทิร์นทั้งหมด 23 แพทเทิร์น ทั้งนี้การศึกษาดังกล่าวจะทำการเปรียบเทียบกับภาษาจาวา ซึ่งเป็นภาษาที่ได้รับความนิยมเป็นอย่างมากในการนำไปใช้สำหรับพัฒนาซอฟต์แวร์ แนวทางในการเปรียบเทียบและการประเมินผลพิจารณาศึกษาแนวคิดเบื้องต้นในการเขียนโปรแกรมนำเสนอในลักษณะของการเขียนโปรแกรมอย่างง่าย คุณสมบัติ คุณลักษณะ และประสิทธิภาพของเวลาที่ใช้ในการประมวลผลภาษาดาร์ท

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2557 ลายมือชื่ออาจารย์ที่ปรึกษา_____

NARITSARA THARAPUTH : A COMPARISON AND EVALUATION OF THE DART LANGUAGE WITH DESIGN PATTERNS. THESIS ADVISOR : ASST. PROF. PICHAYOTAI MAHATTHANAPIWAT, Ph.D., 167 PP.

COMPARISON/EVALUATE/DART LANGUAGE/DESIGN PATTERNS/REUSE

A comparison and evaluation of programming language is important. The purpose of comparison and evaluation is based on the person who studies, such as time reduction for learning the programming language, decision support for the chosen language to be used. This research intends to compare and evaluate the Dart language with the Java language which is widely used in software development by using implementation of examples from 23 design patterns. The approach in comparison and evaluation consists of basic concepts of simple programming, properties, features and execution time efficiency.

School of Computer Engineering Student's Signature _____

Academic Year 2014 Advisor's Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี เนื่องจากได้รับการสนับสนุนจากบุคคลและกลุ่มบุคคลดังต่อไปนี้ที่ได้ให้คำปรึกษา ช่วยเหลือ และแนะนำ ในการดำเนินงานวิจัยจนประสบความสำเร็จ

ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้การสนับสนุน การแนะนำแนวทางในการพัฒนาศึกษา และช่วยเหลือในด้านต่าง ๆ การแนะนำแนวทางในการเขียนบทความ และวิทยานิพนธ์ รวมถึงการตรวจทานแก้ไขเนื้อหาของบทความและวิทยานิพนธ์จนสำเร็จ

ผู้ช่วยศาสตราจารย์ ดร.ชาญวิทย์ แก้วกสิ อาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ให้คำแนะนำทางด้านคุณสมบัติของภาษาเพื่อนำมาพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์น

นายศุภกฤษฎิ์ ตั้งเสริมสิทธิ์ ที่ให้การสนับสนุนและการช่วยเหลือในการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์น

นางสาวกัลญา พับโพธิ์ ที่ให้การสนับสนุนและให้การช่วยเหลือทางด้านเอกสารที่เกี่ยวข้องมาโดยตลอด

สุดท้ายนี้ผู้วิจัยขอกราบขอบพระคุณบิดา มารดา ที่ให้การอุปการะ อบรมเลี้ยงดู ตลอดจนส่งเสริมทางด้านการศึกษา และให้กำลังใจเป็นอย่างดีมาโดยตลอด จนกระทั่งวิทยานิพนธ์ฉบับนี้สำเร็จ

นริศรา ธาระพุทธ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญตาราง (ต่อ)

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

ในอดีตการพัฒนาซอฟต์แวร์ขึ้นมานั้นมีวัตถุประสงค์เพื่อให้เป็นเครื่องมือที่ช่วยในการคำนวณ หรือช่วยในการทำงานที่ต้องการความถูกต้องของงานสูง แต่ในปัจจุบันนี้ซอฟต์แวร์ถูกมองเป็นเครื่องมือที่มีลักษณะของการให้บริการมากขึ้น เพื่อช่วยสนับสนุนการทำงานของผู้ใช้ซอฟต์แวร์ เป็นผลให้จากองค์กรที่เคยทำหน้าที่ในการพัฒนาซอฟต์แวร์ กลายเป็นอุตสาหกรรมสำหรับการพัฒนาซอฟต์แวร์ ในการพัฒนาซอฟต์แวร์มีการทำงานที่เป็นกระบวนการมากขึ้นตลอดจนมีการนำความรู้ทางด้านวิศวกรรมศาสตร์มาใช้ในกระบวนการพัฒนาซอฟต์แวร์ เพื่อให้การออกแบบและการพัฒนาซอฟต์แวร์นั้นมีคุณภาพสูง และสามารถลดต้นทุนต่าง ๆ ไม่ว่าจะเป็นเวลา หรือค่าใช้จ่ายที่อาจเกิดขึ้นตั้งแต่ขั้นตอนการเริ่มพัฒนาซอฟต์แวร์ ไปจนถึงขั้นตอนของการบำรุงรักษาซอฟต์แวร์ ทำให้นักพัฒนาซอฟต์แวร์พยายามที่จะคิดค้นภาษาใหม่ ๆ เพื่อตอบสนองต่อการพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพมากขึ้น โดยในแต่ละภาษาที่มีการคิดค้นและพัฒนาขึ้นมาใหม่นั้น จะมีจุดเด่นและคุณสมบัติที่เหมาะสมกับซอฟต์แวร์แต่ละประเภทแตกต่างกันออกไปตามแต่วัตถุประสงค์ของภาษาที่ได้รับการพัฒนา

เมื่อมีผู้คิดค้นและพัฒนาภาษาที่ใช้สำหรับพัฒนาซอฟต์แวร์ขึ้นมาใหม่ สิ่งที่จะตามมาด้วยคือนักพัฒนาซอฟต์แวร์จำนวนมากจะให้ความสนใจทันทีว่าภาษาที่ใช้สำหรับพัฒนาซอฟต์แวร์ที่ได้รับการพัฒนาใหม่นี้ ดีกว่าภาษาที่ใช้พัฒนาซอฟต์แวร์ที่มีอยู่แล้วอย่างไร มีคุณสมบัติหรือคุณลักษณะเฉพาะอะไรเป็นพิเศษบ้าง เมื่อเปรียบเทียบกับภาษาที่กำลังเป็นที่นิยมหรือภาษาที่ตนเองมีความเชี่ยวชาญ ลักษณะของการศึกษาภาษาที่ได้รับการพัฒนาขึ้นมาใหม่นั้นมีอยู่หลายวิธีด้วยกันคือการเปรียบเทียบ การประเมินผล การใช้แบบสอบถาม หรือแม้แต่การสร้างเป็นหลักสูตรเพื่อการศึกษา

ภาษาดาร์ท (Dart) เป็นภาษาใหม่สำหรับพัฒนาซอฟต์แวร์ที่ได้รับพัฒนาและเปิดตัวเมื่อเดือนตุลาคม ค.ศ. 2011 (Bracha and Bak, 2011) โดยเป็นภาษาที่ใช้สำหรับการสร้างโปรแกรมประยุกต์บนเว็บ ภาษาดาร์ทนี้ได้รับการพัฒนาขึ้นจากทีมพัฒนาของกูเกิล (Google) วัตถุประสงค์หลักของการพัฒนาก็เพื่อให้การสร้างโปรแกรมประยุกต์บนเว็บสามารถทำได้เร็วขึ้น ง่ายต่อการสร้างหรือการพัฒนาโปรแกรมประยุกต์บนเว็บที่มีความทันสมัยและความซับซ้อนสูง การใช้ภาษาดาร์ทจะ

ช่วยให้สามารถเขียนต้นแบบได้อย่างรวดเร็ว เป็นเหตุผลให้สามารถพัฒนาเว็บได้เร็วยิ่งขึ้น และนอกจากนี้ยังมีคลังข้อมูล (Library) ที่มีความน่าเชื่อถือ ความสามารถในการเข้าถึงเครื่องมือที่ทันสมัย และเทคนิคทางวิศวกรรมซอฟต์แวร์ที่ดี แม้ว่าภาษาดาร์ทจะเป็นภาษาที่ถูกพัฒนาขึ้นมาใหม่ แต่ก็มีเครื่องมือที่ช่วยสนับสนุนเป็นจำนวนมาก เช่น Dartboard ที่สามารถช่วยให้เขียนโปรแกรมและรันโค้ดดาร์ทในเบราว์เซอร์ Dart Editor ช่วยให้สามารถสร้าง แก้ไข และรันโปรแกรมประยุกต์ดาร์ท นอกจากนี้ยังมี SDK ที่มีตัวคอมไพเลอร์สำหรับคอมไพล์ภาษาดาร์ท ไปเป็นภาษาจาวาสคริปต์ (JavaScript) และมี Dart VM (Dart virtual machine) ที่ช่วยให้สามารถรันโค้ดที่เป็นภาษาดาร์ทบนเซิร์ฟเวอร์ได้ เป็นต้น (Walrath and Ladd, 2012) ยกตัวอย่างความน่าสนใจที่เกี่ยวกับภาษาดาร์ทที่ได้รับนำเสนอจากทีมผู้พัฒนา (Walrath and et al., 2013a)

1. ภาษาที่ง่ายต่อการเรียนรู้ และมีไวยากรณ์ที่นักพัฒนาคุ้นเคย
2. ความสามารถในการคอมไพล์ดาร์ทไปยังจาวาสคริปต์
3. ความสามารถในการทำงานได้ทั้งบนเซิร์ฟเวอร์และฝั่งไคลเอนต์
4. มีคลังข้อมูลที่พร้อมใช้งานและคุณสมบัติพื้นฐานจำนวนมาก
5. สนับสนุนการทำงานแบบภาวะพร้อมกัน (Concurrency) และความปลอดภัย
6. สนับสนุนโค้ดที่ใช้ร่วมกันกับโค้ดที่มาจากคลังข้อมูลของภาษาอื่น
7. เป็นโอเพนซอร์สภายใต้ใบอนุญาต BSD (BSD license) เป็นต้น

ในการออกแบบซอฟต์แวร์เชิงวัตถุซึ่งถือเป็นเรื่องที่ยาก แต่การออกแบบเพื่อให้สามารถนำซอฟต์แวร์เชิงวัตถุนั้นกับมาใช้ซ้ำได้เป็นเรื่องที่ยากยิ่งกว่า เพราะฉะนั้นแล้วในการออกแบบซอฟต์แวร์เชิงวัตถุเพื่อให้ซอฟต์แวร์มีประสิทธิภาพและสามารถใช้ซ้ำได้มากที่สุด และมีความยืดหยุ่นต้องอาศัยประสบการณ์จากการออกแบบและการพัฒนาซอฟต์แวร์ ในทางวิศวกรรมซอฟต์แวร์ ดีไซน์แพทเทิร์น (Design pattern) ถือเป็นแนวทางที่ได้รับความนิยมในการนำไปใช้แก้ไขปัญหาการออกแบบซอฟต์แวร์ ซึ่งดีไซน์แพทเทิร์นไม่ใช่การออกแบบที่จะนำไปใช้ในการแก้ไขปัญหาโดยตรง แต่ดีไซน์แพทเทิร์นเป็นการอธิบายแนวทางที่จะสามารถนำไปประยุกต์ใช้ในการออกแบบตามแต่สถานการณ์ที่แตกต่างกัน (Gamma and et al., 1995)

การศึกษาและพัฒนาดีไซน์แพทเทิร์นนั้นถือเป็นเรื่องสำคัญ จึงมีการบันทึกลักษณะของการออกแบบให้อยู่ในลักษณะที่ผู้สนใจสามารถศึกษาได้ ดีไซน์แพทเทิร์นที่ได้รับความนิยมเป็นอย่างมาก และถือว่ามีอิทธิพลเป็นอย่างมากในงานของวิศวกรรมซอฟต์แวร์ และยังเป็นแหล่งข้อมูลที่สำคัญสำหรับทฤษฎีการออกแบบเชิงวัตถุ คือ ดีไซน์แพทเทิร์นของ Gang-of-Four (GoF) ที่มีวัตถุประสงค์หลักคือ การบันทึกประสบการณ์ที่เกี่ยวข้องกับการออกแบบ ซอฟต์แวร์เชิงวัตถุให้เป็นดีไซน์แพทเทิร์น และสามารถนำการออกแบบนั้นไปใช้ในการเขียนโปรแกรมได้อย่างมี

ประสิทธิภาพ (Gamma and et al., 1995)

จากข้อมูลของภาษาดาร์ทและดีไซน์แพทเทิร์นที่ได้มีการนำเสนอไว้ในข้างต้นแล้วนั้น เป็นเหตุผลสำคัญที่ทำให้ผู้วิจัยพิจารณาเลือกภาษาดาร์ท ในการศึกษาแนวคิดเบื้องต้นในการเขียนโปรแกรมอย่างง่าย การศึกษาคุณสมบัติ และคุณลักษณะต่าง ๆ ของภาษาดาร์ทด้วยดีไซน์แพทเทิร์น เพื่อให้เห็นถึงคุณสมบัติและคุณลักษณะได้อย่างชัดเจน และทำการทดสอบประสิทธิภาพในเรื่องของเวลาที่ใช้ในการประมวลผลโดยชุดทดสอบมาตรฐาน (Benchmark) ในการศึกษาดังกล่าวจะศึกษาโดยทำการเปรียบเทียบกับภาษาจาวา (Java) ซึ่งเป็นภาษาที่เป็นที่รู้จักกันอย่างกว้างขวาง และยังเป็นภาษาที่ได้รับความนิยมในการนำไปพัฒนาซอฟต์แวร์เป็นอย่างมาก แม้ว่าภาษาดาร์ทจะเป็นภาษาที่ใกล้เคียงกับภาษาจาวา และภาษาจาวาสคริปต์ แต่ในการศึกษาครั้งนี้จะเปรียบเทียบเฉพาะภาษาจาวาเท่านั้น ส่วนในภาษาจาวาสคริปต์จะไม่นำมาเปรียบเทียบ เนื่องจากภาษาจาวาสคริปต์เป็นการเขียนโปรแกรมแบบโพรโทไทป์ (Prototype-based programming) ซึ่งแม้จะเป็นลักษณะของการเขียนโปรแกรมเชิงวัตถุชนิดหนึ่งแต่ไม่ได้ใช้แนวความคิดแบบคลาส การเรียกใช้งานซ้ำจะอยู่ในลักษณะของการสืบทอดพฤติกรรมโดยใช้วิธีการคัดลอกต้นแบบที่มีอยู่แล้ว (Flanagan, 2011) ด้วยลักษณะของภาษาจาวาสคริปต์ที่ได้กล่าวมาซึ่งมีแนวคิดของภาษาและลักษณะของภาษาที่แตกต่างกันกับภาษาดาร์ท ทำให้สามารถเห็นถึงความแตกต่างระหว่างภาษาได้อย่างชัดเจนแล้ว จึงไม่มีประโยชน์หากจะนำภาษาดาร์ทมาเปรียบเทียบกับภาษาจาวาสคริปต์

1.2 วัตถุประสงค์ของการวิจัย

งานวิจัยนี้มีวัตถุประสงค์หลักเพื่อนำเสนอการเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยดีไซน์แพทเทิร์น โดยมีวัตถุประสงค์ย่อยดังต่อนี้

1.2.1 เพื่อศึกษาแนวคิดเบื้องต้นในการเขียนโปรแกรม และวากยสัมพันธ์ (Syntax) ต่าง ๆ โดยวิธีการเปรียบเทียบระหว่างภาษาดาร์ท และภาษาจาวา

1.2.2 เพื่อศึกษาคุณสมบัติและคุณลักษณะในการเขียนโปรแกรมของภาษาดาร์ทเปรียบเทียบกับภาษาจาวาด้วยดีไซน์แพทเทิร์น

1.2.3 เพื่อศึกษาจำนวนบรรทัดของรหัสต้นฉบับ (Source code) ที่ได้พัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นของภาษาดาร์ทเปรียบเทียบกับภาษาจาวา

1.2.4 เพื่อศึกษาประสิทธิภาพของเวลาที่ใช้ในการประมวลผลของภาษาดาร์ทเปรียบเทียบกับภาษาจาวา ด้วยชุดทดสอบประสิทธิภาพมาตรฐาน

1.3 สมมุติฐานการวิจัย

จากวัตถุประสงค์หลักและวัตถุประสงค์ย่อยในการวิจัยครั้งนี้ ทำให้ผู้วิจัยสามารถตั้งสมมุติฐานของการวิจัยได้ดังนี้

1.3.1 แนวคิดเบื้องต้นในการเขียนโปรแกรมและวากยสัมพันธ์ของภาษาดาร์ท และภาษาจาวา จากเปรียบเทียบลักษณะต่าง ๆ ของทั้งสองภาษา มีความใกล้เคียงกัน เนื่องจากเป็นภาษาเขียนโปรแกรมเชิงวัตถุเหมือนกัน

1.3.2 ภาษาดาร์ทจะมีคุณสมบัติและคุณลักษณะในการเขียนโปรแกรมด้วยดีไซน์แพทเทิร์นเพื่อให้สามารถนำซอร์สโค้ดกลับมาใช้ใหม่ได้มากกว่าหรือใกล้เคียงกับภาษาจาวา

1.3.3 การพัฒนาตัวอย่างดีไซน์แพทเทิร์นด้วยภาษาดาร์ท สามารถพัฒนาได้มีประสิทธิภาพกว่าภาษาจาวา เช่น การเขียนโปรแกรมที่เหมือนกัน แต่ภาษาดาร์ทใช้จำนวนบรรทัดในการเขียนน้อยกว่าภาษาจาวาและภาษาดาร์ทใช้เวลาในการประมวลผลน้อยกว่าภาษาจาวาเมื่อทดสอบด้วยชุดทดสอบประสิทธิภาพมาตรฐาน

1.4 ขอบเขตของการวิจัย

ในการวิจัยตามวัตถุประสงค์ และเพื่อให้การวิจัยเป็นไปตามสมมติฐานที่ได้ตั้งไว้ จำเป็นต้องมีการกำหนดขอบเขตของงานวิจัยอย่างชัดเจน ซึ่งผู้วิจัยได้กำหนดขอบเขตของการวิจัยไว้ดังนี้

1.4.1 การศึกษาแนวคิดเบื้องต้นในการเขียนโปรแกรม คุณสมบัติ และคุณลักษณะของภาษาใช้ภาษาดาร์ท รุ่นที่ 1.2 เทียบกับ ภาษาจาวา รุ่นที่ 1.7.0_40

1.4.2 ดีไซน์แพทเทิร์นที่จะนำมาใช้ในการศึกษาคุณสมบัติ คุณลักษณะ และประสิทธิภาพของภาษา จะใช้ดีไซน์แพทเทิร์นของ Gang-of-Four ทั้ง 23 แพทเทิร์น จากหนังสือ “Design Patterns : Elements of Reusable Object - Oriented Software” เท่านั้น

1.4.3 ชุดทดสอบประสิทธิภาพในการจับเวลาสำหรับภาษาดาร์ทและภาษาจาวาคือ DeltaBlue Benchmark

1.5 ประโยชน์ที่คาดว่าจะได้รับ

งานวิจัยนี้มีวัตถุประสงค์เพื่อนำเสนอการเปรียบเทียบและการประเมินภาษาดาร์ท เพื่อศึกษาคุณสมบัติ คุณลักษณะและประสิทธิภาพของภาษาด้วยดีไซน์แพทเทิร์น และให้เป็นไปตามสมมติฐานที่ได้ตั้งไว้ ทำให้สามารถคาดถึงประโยชน์ของงานวิจัยที่จะได้รับ ซึ่งประโยชน์ที่คาดว่าจะได้รับจากงานวิจัยมีดังต่อไปนี้

1.5.1. เพื่อช่วยให้นักพัฒนาเห็นถึงความแตกต่างระหว่างภาษาดาร์ท เมื่อเปรียบเทียบกับภาษาจาวาได้ชัดเจน

1.5.2. เพื่อช่วยให้นักพัฒนาสามารถลดระยะเวลาและลดภาระที่จะใช้ในการศึกษาคุณสมบัติและคุณลักษณะต่าง ๆ ของภาษาดาร์ท

1.5.3. เพื่อช่วยให้นักพัฒนาสามารถศึกษาถึงประสิทธิภาพของเวลาที่ใช้ในการประมวลผลของภาษาดาร์ท เมื่อเปรียบเทียบกับภาษาจาวาได้สะดวกขึ้น

1.5.4. เพื่อช่วยในนักพัฒนาใช้ในการตัดสินใจเพื่อทำการเลือกภาษาที่จะใช้ในการพัฒนากรอบงานหรือโปรแกรมประยุกต์ในอนาคต

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะเป็นการนำเสนอปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้องกับการเปรียบเทียบและการประเมินภาษาสำหรับเขียนโปรแกรม โดยในหัวข้อที่ 2.1 จะเป็นการอธิบายถึงภาพรวมและลักษณะเฉพาะของภาษาดาร์ทที่น่าสนใจ หัวข้อที่ 2.2 เป็นการอธิบายถึงดีไซน์แพทเทิร์นของ GoF ทั้ง 23 แพทเทิร์น และในหัวข้อที่ 2.3 เป็นงานวิจัยที่เกี่ยวข้องกับการเปรียบเทียบและการประเมินผลภาษาโปรแกรม

2.1 ภาษาดาร์ท

ภาษาดาร์ทถูกพัฒนาขึ้นโดยมีวัตถุประสงค์เพื่อให้เป็นภาษาสำหรับการพัฒนาโปรแกรมประยุกต์บนเว็บ ซึ่งเป็นภาษาที่ถูกพัฒนาขึ้นมาใหม่โดยทีมพัฒนาจากกูเกิล และได้รับความสนใจเป็นอย่างมากในวงการพัฒนาโปรแกรมประยุกต์บนเว็บ (Walrath and Ladd, 2012) ทีมพัฒนามีความคาดหวังว่าดาร์ทจะช่วยให้การพัฒนาและประสิทธิภาพของโปรแกรมประยุกต์บนเว็บดีขึ้น มีการนำเสนอถึงความน่าสนใจในการใช้ดาร์ทเพื่อพัฒนาโปรแกรมประยุกต์บนเว็บดังต่อไปนี้ (Walrath and et al., 2013a)

1. ดาร์ทเป็นภาษาที่ถูกพัฒนามาให้ง่ายต่อการเรียนรู้ และเป็นภาษาเขียนโปรแกรมเชิงวัตถุ อีกทั้งยังมีไวยากรณ์ของภาษาเป็นที่คุ้นเคยของนักพัฒนา ทำให้นักพัฒนาสามารถเรียนรู้ได้อย่างรวดเร็วในเวลาเพียงไม่กี่ชั่วโมง
2. ดาร์ทสามารถคอมไพล์ไปเป็นจาวาสคริปต์ได้ ทั้งนี้ก็เพื่อให้โปรแกรมประยุกต์ที่พัฒนาด้วยดาร์ทสามารถทำงานข้ามแพลตฟอร์ม (Platform) ของเว็บที่มีทั้งหมดได้ ซึ่งเป็นเหตุผลเบื้องต้นที่ดาร์ทได้รับการออกแบบให้สามารถคอมไพล์ไปเป็นจาวาสคริปต์
3. ดาร์ทสามารถทำงานได้ทั้งบนเซิร์ฟเวอร์และฝั่งไคลเอนต์ มีการพัฒนา Dart VM ที่แม้ว่าจะเป็นการทำงานโดยลำพัง (Standalone) บนบรรทัดคำสั่ง (Command line) แต่ก็ถูกพัฒนาให้สามารถทำงานร่วมกับเว็บเบราว์เซอร์ได้
4. ดาร์ทถูกพัฒนาขึ้นมาพร้อมกับเอดิเตอร์แบบเบา (Lightweight) ที่จะเป็นตัวช่วยในการเริ่มพัฒนาโปรแกรม การตรวจสอบข้อผิดพลาดของการเขียนโปรแกรม และการแก้ไขจุดบกพร่องของโปรแกรมประยุกต์ดาร์ท การใช้เอดิเตอร์ในการพัฒนาโปรแกรมจะมีส่วนช่วยให้การพัฒนาโปรแกรมสามารถทำได้ดี รวดเร็ว และสำเร็จได้ง่าย

5. ดาร์ทมีระบบแบบชนิดเชิงทางเลือก (Optional type system) และยังสนับสนุนการกำหนดแบบชนิดของตัวแปร เนื่องจากบางครั้งไม่จำเป็นที่จะต้องกำหนดแบบชนิดของตัวแปร เพราะในการพัฒนาอาจยังไม่แน่ใจถึงโครงสร้างของโปรแกรมประยุกต์ และการเปลี่ยนแปลงแบบชนิดของตัวแปรจะสามารถทำได้อย่างรวดเร็ว
6. ดาร์ทสนับสนุนการพัฒนาโปรแกรมประยุกต์ที่มีความซับซ้อน ทำให้สามารถพัฒนาสคริปต์จากขนาดเล็กไปเป็นสคริปต์ขนาดใหญ่ได้ นั่นคือสามารถทำการเพิ่มโค้ดและโครงสร้างที่มากขึ้นได้ตลอดเวลา เพราะการพัฒนาเว็บนั้นเป็นการพัฒนาด้วยกระบวนการเชิงวนซ้ำ (Iterative process)
7. ดาร์ทมีคลังข้อมูลในตัว (Built-in libraries) และคุณสมบัติพื้นฐานจำนวนมาก เช่น คอลเลคชัน (Collections) วันที่ (Dates) เป็นต้น และยังสนับสนุนการใช้คลังข้อมูลอื่น ๆ เช่น การพัฒนาโปรแกรมประยุกต์บนเว็บ สามารถเรียกใช้คลังข้อมูลของ HTML ได้
8. ดาร์ทสนับสนุนการทำงานแบบภาวะพร้อมกัน และความปลอดภัยในการทำงานแบบภาวะพร้อมกันนี้ได้รับแนวคิดมาจากภาษาเออแลง (Erlang)
9. ดาร์ทสนับสนุนโค้ดที่ใช้ร่วมกันกับโค้ดที่มาจากคลังข้อมูลของภาษาอื่น ด้วย Dart pub (Dart package manager) ซึ่งในการเขียนเว็บแบบดั้งเดิมนั้นไม่สามารถใช้คลังข้อมูลจากคลังข้อมูลของบุคคลที่สาม (Third party) ได้โดยพลการ
10. ดาร์ทเป็นโอเพนซอร์ภายใต้ใบอนุญาต BSD ซึ่งทำให้สามารถติดตามปัญหา (Issue tracker) และการเก็บต้นฉบับ (Source repository) แบบออนไลน์ได้

2.1.1 ภาพรวมของภาษาดาร์ท

ดาร์ทเป็นภาษาเขียนโปรแกรมเชิงวัตถุ (Object-oriented) มีลักษณะของการสืบทอดเป็นการสืบทอดแบบเดี่ยว (Single-inheritance) การใช้คลาสเป็นพื้นฐาน (Class-based) การกำหนดแบบชนิดเชิงทางเลือก และยังมีการสนับสนุนการระบุแบบชนิดทั่วไปด้วย ชนิดของการรันไทม์ (Runtime type) จะแสดงในลักษณะอินสแตนซ์ของคลาสสำหรับทุก ๆ อ๊อบเจกต์ เนื่องจากดาร์ทมีรากฐานคลาสที่เป็นแบบลำดับชั้น การจะให้ได้มาซึ่งแบบชนิดนั้นสามารถทำการเรียกเกทเทอร์ (Getter) runtimeType ในคลาสของอ๊อบเจกต์นั้นได้ คอมไพเลอร์ดาร์ทมีการตรวจสอบแบบเชิงสถิตย์ (Statically checked) และจะทำการดำเนินการ (Execute) หนึ่งในสองแบบวิธีต่อไปนี้ แบบวิธีแรกคือ แบบวิธีการผลิต (Production mode) เป็นแบบวิธีที่ถูกกำหนดให้เป็นแบบวิธีเริ่มต้นของการนำไปใช้งาน ซึ่งในแบบวิธีนี้จะไม่สนใจแบบชนิดเชิงสถิตย์ (Static types) และคำสั่ง assert และแบบวิธีที่สองคือ แบบวิธีตรวจสอบ (Checked mode) เป็นแบบวิธีสำหรับการพัฒนาและการดัก

จับข้อผิดพลาด ซึ่งจะช่วยให้สามารถจัดการกับข้อผิดพลาดบางอย่างในช่วงระยะเวลาดำเนินการ (Run-time) (The Dart Team, 2013)

ดาร์ทนอกจากจะมีเป้าหมายสำหรับการพัฒนาโปรแกรมประยุกต์บนเว็บแล้ว ยังมีเป้าหมายสำหรับการพัฒนาโปรแกรมประยุกต์แบบตั้งโต๊ะ (Desktop) การพัฒนาเบราว์เซอร์บนมือถือให้มีประสิทธิภาพสูง และความสามารถในการคอมไพล์ดาร์ทไปเป็นจาวาสคริปต์ ทั้งนี้ในการพัฒนามีเป้าหมายในเรื่องของความมีประสิทธิภาพ ความเรียบง่าย และความสามารถในการปรับการตั้งค่าและขนาดเพื่อให้พอดีกับเงื่อนไขใหม่ สิ่งที่จะกล่าวถึงต่อไปนี้เป็นเป้าหมายที่สำคัญของการออกแบบและยังเป็นแนวทางในการพัฒนาอย่างต่อเนื่องของโครงการการพัฒนาดาร์ท (Walrath and et al., 2013b)

1) การสร้างภาษาเขียนโปรแกรมที่โครงสร้างมีความยืดหยุ่นสำหรับการพัฒนาเว็บ
2) การทำให้ดาร์ทง่ายต่อการเรียนรู้ โดยการทำให้มีความรู้สึกที่คุ้นเคยสำหรับไวยากรณ์ในการเขียนโปรแกรม
3) การตรวจสอบให้แน่ใจว่าโครงสร้างของภาษาช่วยให้การพัฒนาโปรแกรมประยุกต์มีประสิทธิภาพสูงและเริ่มต้นโปรแกรมประยุกต์ได้อย่างรวดเร็ว
4) การทำให้ดาร์ทเหมาะสำหรับอุปกรณ์บนเว็บ รวมทั้งแท็บเล็ต แลปท็อป โทรศัพท์มือถือ และเซิร์ฟเวอร์
5) การให้บริการเครื่องมือที่ทำให้ดาร์ททำงานผ่านเบราว์เซอร์ได้อย่างรวดเร็ว

เป้าหมายของการออกแบบที่ได้กล่าวถึงนั้นเพื่อแก้ไขปัญหาต่าง ๆ ที่นักพัฒนาเว็บกำลังเผชิญ เช่น สคริปต์ขนาดเล็กบ่อยครั้งที่จะถูกพัฒนาให้เป็นสคริปต์ขนาดใหญ่ทั้งที่ยังมีโครงสร้างที่ไม่ชัดเจน การที่นักพัฒนาถูกบังคับให้เลือกระหว่างภาษาเชิงสถิตย์ ซึ่งจำเป็นต้องใช้รูปแบบโค้ดที่ไม่ยืดหยุ่น มีข้อจำกัดและกลุ่มของเครื่องมือขนาดใหญ่ และภาษาเชิงพลวัต ในภาษาสคริปต์ส่วนใหญ่มักจะมีการอธิบายการส่งผ่านข้อมูลต่าง ๆ ไว้ในคอมเมนต์มากกว่าในโครงสร้างของตัวภาษา ซึ่งอาจทำให้เป็นเรื่องยากสำหรับผู้อื่นที่ไม่ใช่ผู้เขียนในการอ่านและบำรุงรักษารหัสต้นฉบับ

2.1.2 ลักษณะเฉพาะของดาร์ท

ภาษาดาร์ทเป็นภาษาที่ง่ายต่อการเรียนรู้ เพราะถูกออกแบบและพัฒนาให้มีลักษณะใกล้เคียงกับภาษาที่ใช้เขียนโปรแกรมที่มีอยู่ก่อนหน้านี้ โดยเฉพาะภาษาจาวา และภาษาจาวาสคริปต์ ทำให้สามารถเขียนโปรแกรมได้เหมือนกับว่าเป็นหนึ่งในภาษานั้นได้ นอกจากภาษาจาวาและภาษาจาวาสคริปต์แล้ว ดาร์ทยังสามารถเขียนโปรแกรมได้เหมือนเป็นหนึ่งในภาษาฟอร์แทน (Fortran) ได้อีกด้วย แต่การกระทำดังกล่าวก็จะทำให้ความสนุกในการเขียนโปรแกรมและลักษณะเฉพาะของภาษาดาร์ทหายไป การแนะนำลักษณะเฉพาะของภาษาดาร์ทจึงเป็นสิ่งที่จำเป็น

เพื่อให้ผู้เขียนโปรแกรมมีความคิดที่ออกมาจากการเขียนโปรแกรมด้วยภาษาจาวา หรือภาษาจาวาสคริปต์ และแทนที่ความคิดนั้นด้วยลักษณะเฉพาะของภาษาดาร์ท ลักษณะเฉพาะของภาษาดาร์ทที่ได้รับการนำเสนอจากทีมผู้พัฒนาดังต่อไปนี้ (Nystrom, 2013; Ladd, 2013; Walrath and et al., 2013c)

1. แบบชนิดของข้อมูล

ดาร์ทเป็นภาษาที่ถูกพัฒนาขึ้นโดยเน้นลักษณะของแบบชนิดเชิงทางเลือก แต่ก็มีการสนับสนุนแบบชนิดที่มีลักษณะพิเศษอยู่ในตัวของภาษาทั้งหมด 6 ชนิด คือ

1.1 ตัวเลข (Numbers): ดาร์ทมีแบบชนิดของตัวเลขอยู่ 2 ชนิด คือ int และ double ซึ่งทั้งคู่เป็นซับคลาส (Subclass) ของคลาส num โดยปกติการเขียนโค้ดด้วยดาร์ทนั้น จะพบการใช้ตัวเลขอยู่ 2 ชนิด คือ

1.1.1 จำนวนเต็มเท่านั้น ไม่รวมจุดทศนิยม : ใช้ int ในการจัดการ

1.1.2 จำนวนตัวเลขใด ๆ รวมจุดทศนิยม : จะใช้ num ในการจัดการ

ตัวอย่างการใช้งานแบบชนิดของตัวเลขดังแสดงในรูปที่ 2.1 ด้วยการใช้แบบชนิดของตัวแปรเป็น var แต่ทำการเปรียบเทียบว่าเป็นแบบชนิดของคลาส num และซับคลาส โดยการใช้ตัวดำเนินการ is (ผลลัพธ์ที่ได้จะเป็น true ถ้าอ๊อบเจกต์มีแบบชนิดของตัวแปรที่กำหนด)

```
void main() {
  var doub = 4.22;
  var hex = 0xDEAEBEFF;

  print('line 1 : $doub');              //line 1
  print('line 2 : ${doub is double}'); //line 2
  print('line 3 : $hex');               //line 3
  print('line 4 : ${hex is num}');      //line 4
  print('line 5 : ${hex is int}');      //line 5
  print('line 6 : ${hex is double}');   //line 6
}
```

```
Output:
        line 1 : 4.22
        line 2 : true
        line 3 : 3735994111
        line 4 : true
        line 5 : true
        line 6 : false
```

รูปที่ 2.1 การแสดงคุณสมบัติแบบชนิดของตัวแปรตัวเลข

1.2. สตริง (Strings): ดาร์ทสนับสนุนการเก็บรหัสของสตริงแบบ UTF-16 การใช้งานสตริงจะถูกนำเสนอในหัวข้อสตริงต่อไป

1.3. บูลีน (Booleans): แบบชนิดของบูลีนในดาร์ทจะเก็บค่าตัวอักษร true และ false การใช้ค่าของบูลีนจะกำหนดแบบชนิดเป็น bool ดังแสดงในรูปที่ 2.2

```
void main() {
  bool fname = true;
  bool lname = false;
  if(fname){
    print("Naritsara");
  }
  if (lname != true) {
    print ("Tharaputh");
  }
}
```

```
Output:
        Naritsara
        Tharaputh
```

รูปที่ 2.2 การใช้งานแบบชนิดของบูลีน

1.4. ลิสต์ (Lists): มีลักษณะคล้ายกับอะเรย์ (Array) ในภาษาทั่วไป การใช้งานแบบชนิดของลิสต์จะถูกนำเสนอในหัวข้อลิสต์ต่อไป

1.5. แมป (Maps): คือการกำหนดอ๊อบเจกต์สำหรับเก็บตัวกำกับ (Key) และค่า (Value) ดาร์ทสนับสนุนทั้งการแมปตัวอักษรและแบบชนิดของแมปที่มีอยู่ในตัวภาษาดาร์ท ยกตัวอย่างการใช้งานแมปทั้งสองโดยกำหนดให้ตัวแปร fruit เป็นการแมปแบบตัวอักษร ตัวกำกับของแบบอักษรจะเป็นสตริง และตัวแปร map เป็นการเรียกใช้งานคอนสตรัคเตอร์ (Constructors)ของแมป และตัวกำกับของการเรียกใช้งานนี้จะมีลักษณะเป็นอ๊อบเจกต์ ดังแสดงในรูปที่ 2.3

1.6. สัญลักษณ์ (Symbols): คือการประกาศตัวแทนของตัวดำเนินการหรือตัวระบุ การใช้สัญลักษณ์สำหรับตัวระบุ ใช้สัญลักษณ์ # และตามด้วยตัวระบุ ตัวอย่างการใช้งานดังแสดงในรูปที่ 2.4

```
void main() {
  var fruit = {
                'first' : 'Mangosteen',
                'second': 'Sugar',
                'thrid' : 'Mango'
               };

  var map = new Map();
  map[3] = 'Mangosteen';
  map[5] = 'Apple';
  map[9] = 'Mango';

  print(fruit['first']);
  print('${fruit["second"]} ${map[5]}');
  print(map[9]);
}
```

```
Output:
        Mangosteen
        Sugar Apple
        Mango
```

รูปที่ 2.3 การใช้งานแบบชนิดของแมป

```
import 'dart:mirrors';

const className = #MyClass;
void main(){
  print('MyClass' == MirrorSystem.getName(className));
}
```

```
Output:
        true
```

รูปที่ 2.4 การใช้งานแบบชนิดของสัญลักษณ์

จากแบบชนิดของข้อมูลในภาษาดาร์ทที่ได้มีการนำเสนอในข้างต้นแล้ว สามารถทำการสรุปแบบชนิดได้ดังแสดงในตารางที่ 2.1

ตารางที่ 2.1 แบบชนิดของข้อมูลพื้นฐานในภาษาดาร์ท

แบบชนิดของข้อมูล	รายละเอียด
String	เป็นรหัสสตริงแบบ UTF-16
num	เป็นอ๊อบเจกต์สำหรับเก็บค่าตัวเลข
int	ขนาดจะขึ้นอยู่กับเครื่องจักรเสมือน (Virtual machines)
double	มีขนาด 64 บิต ตามมาตรฐานของ IEEE 754
bool	เก็บค่า true และ false
list	การเก็บข้อมูลเป็นกลุ่ม
map	อ๊อบเจกต์สำหรับเก็บตัวกำกับและค่า
symbols	การสร้างสัญลักษณ์ให้กับตัวดำเนินการหรือตัวระบุ

2. ฟังก์ชัน

ดาร์ทมีคุณสมบัติฟังก์ชัน (Functions) เป็นวัตถุชั้นหนึ่ง (First-class objects) และเป็นอ๊อบเจกต์เหมือนภาษาที่นิยมกันทั่วไป ในการสร้างฟังก์ชันของดาร์ทสัญลักษณ์ที่ใช้ในการสร้างฟังก์ชันมี 3 สัญลักษณ์ คือ

2.1 ฟังก์ชันระบุชื่อ (Named functions)

2.2 ฟังก์ชันไม่ระบุชื่อ (Anonymous functions) สำหรับประโยคคำสั่ง (Statement)

2.3 ฟังก์ชันลูกศร (Arrow functions) สำหรับนิพจน์ (Expression)

ตัวอย่างการสร้างฟังก์ชันจากสัญลักษณ์ทั้งสามคือ ฟังก์ชันที่ระบุชื่อดังแสดงในรูปที่ 2.5 (ก) ฟังก์ชันที่ไม่ระบุชื่อดังแสดงในรูปที่ 2.5 (ข) และฟังชันลูกศรดังแสดงในรูปที่ 2.5 (ค)

3. ลิสต์

ในการเก็บข้อมูลเป็นรายการของดาร์ทจะต้องใช้ตัวแปรประเภทลิสต์ ซึ่งเป็นแบบชนิดที่มีอยู่ในภาษาดาร์ทและมีลักษณะคล้ายกับอะเรย์ในภาษาอื่น ๆ โดยมีค่าดัชนี (Index) ที่เป็นองค์ประกอบแรกของลิสต์เริ่มต้นที่ 0 และค่าดัชนีที่เป็นองค์ประกอบสุดท้ายจะเป็นความยาวลบ 1 (Length–1) ยกตัวอย่างจากรูปที่ 2.6 เป็นตัวอย่างการใช้งานลิสต์เบื้องต้นอย่างง่าย ๆ มีการกำหนดตัวแปร list สำหรับเก็บค่าของกลุ่มข้อมูลตัวเลขตั้งแต่ 1 ถึง 5 มีการเปรียบเทียบค่าว่าตัวแปร list

เป็นแบบชนิดของลิสต์ และการแสดงผลข้อมูลตามรูป

(ก) ฟังก์ชันระบุชื่อ

```
bool isWhispering (String msg){
  return (msg.toLowerCase() == msg);
}
```

(ข) ฟังก์ชันไม่ระบุชื่อ

```
var whisper2 = msg.where((m) {
  return (m.toLowerCase() == m);
}).toList();
```

(ค) ฟังก์ชันลูกศร

```
var whisper3 = msg.where((m)
     => m.toLowerCase() == m).toList();
```

รูปที่ 2.5 วิธีการสร้างฟังก์ชันจากสัญลักษณ์ทั้ง 3 สัญลักษณ์

```
void main() {
  var list = [1, 2, 3, 4, 5];
  assert(list is List);
  print('Value index[0] : ${list[0]}');
  print('Last index     : ${list.length - 1}');
  print('Length of list : ${list.length}' );
}
```

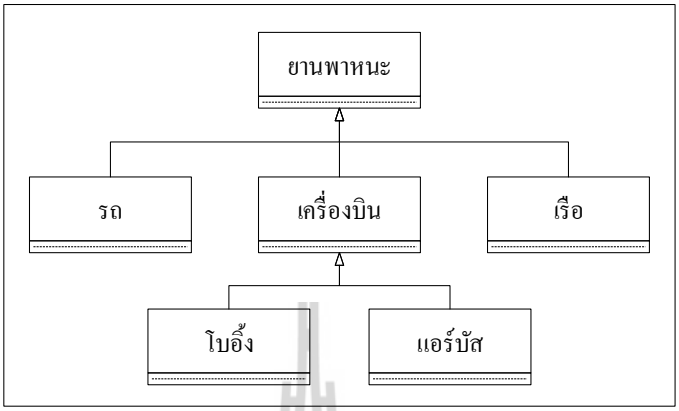
```
Output:
        Value index[0]: 1
        Last index    : 4
        Length of list: 5
```

รูปที่ 2.6 การใช้งานลิสต์เบื้องต้น

4. คลาส อ๊อบเจกต์และคอนสตรัคเตอร์

4.1 คลาส: เป็นแม่แบบที่มีคุณลักษณะและพฤติกรรมที่คล้ายกัน หรืออาจกล่าวได้ว่าคลาสเป็นแนวความคิดที่มีไว้สำหรับสร้างอ๊อบเจกต์ให้เป็นจริง คลาสสามารถถูกขยายให้มีลักษณะที่มีความเฉพาะเจาะจงมากยิ่งขึ้นได้ในลักษณะของลำดับชั้นแบบต้นไม้ ยกตัวอย่างเช่น การขยายประเภทของยานพาหนะ สามารถขยายโดยการกำหนดแนวคิดว่าเป็นรถ เครื่องบิน หรือเรือ และยังสามารถกำหนดชนิดที่เฉพาะเจาะจงลงไปได้ว่าเป็นเครื่องบินชนิดไหนดังแสดงในรูปที่ 2.7 (การ

อธิบายคุณลักษณะของคลาส และโครงสร้างแผงผังคลาสจะใช้สัญลักษณ์ของ UML)



รูปที่ 2.7 ตัวอย่างคลาสลำดับชั้นแบบต้นไม้

ตัวอย่างการสร้างคลาสและการเรียกใช้งานคลาสอย่างง่าย ๆ ในภาษาดาร์ทดังแสดงในรูปที่ 2.8 จะเห็นว่าการสร้างคลาสจะต้องประกอบไปด้วยคำสงวนเพื่อใช้ในการประกาศคลาสคือ class ชื่อของคลาสคือ Car และข้อมูลของคลาส (Body) ที่อยู่ภายในสัญลักษณ์ { } ในที่นี้คือ การกำหนดสตริงให้กับตัวแปร car และการสร้างเมธอด doPrint()

```
class Car{
  String car = "I'm class Car" ;
  doPrint(){
    return car;
  }
}
```

รูปที่ 2.8 คลาส `Car`

4.2 อ๊อบเจกต์: ดาร์ทถูกพัฒนาให้เป็นภาษาเชิงวัตถุ ทำให้ทุกอย่างถูกมองเป็นอ๊อบเจกต์ทั้งหมด เช่น การกำหนดค่าให้กับตัวแปร อินสแตนซ์ของคลาส ตัวเลข ฟังก์ชัน ไม่เว้นแม้แต่ค่าว่าง (null) ก็ถูกมองเป็นอ๊อบเจกต์เช่นกัน ดาร์ทได้มีการกำหนดให้อ๊อบเจกต์ทั้งหมดจะต้องสืบทอดมาจากคลาส Object ซึ่งเป็นคลังข้อมูลในตัว ตัวอย่างการสร้างอ๊อบเจกต์อย่างง่ายในดาร์ทดังแสดงในรูปที่ 2.9 เป็นการสร้างอ๊อบเจกต์จากคลาส Car ก่อนหน้านี้ ภายในฟังก์ชัน main() มีการ

สร้างอ๊อบเจกต์โดยการใช้คำสั่ง new เก็บไว้ในตัวแปร c เมื่อทำการแสดงค่าของตัวแปร c จะพบว่าตัวแปร c เป็นอินสแตนซ์ของคลาส Car

```
class Car{
  String car = "I'm class Car" ;
  doPrint(){
    return car;
  }
}

void main(){
  Car c = new Car();
  print(c);
}
```

```
Output:
        Instance of 'Car'
```

รูปที่ 2.9 การสร้างอ๊อบเจกต์อย่างง่าย

4.3 คอนสตรัคเตอร์: คือโครงสร้างคล้ายเมธอดที่ชื่อเหมือนกับคลาส โดยคอนสตรัคเตอร์นั้นจะทำงานทันทีเมื่อมีการสร้างอ๊อบเจกต์ของคลาสดังกล่าว ในภาษาดาร์ทมี การนำเสนอแนวคิดของคอนสตรัคเตอร์ที่น่าสนใจอยู่ 3 แนวคิดในขณะนี้คือ การตั้งค่าเริ่มต้นให้เขตข้อมูลโดยอัตโนมัติ (Automatic field initialization) คอนสตรัคเตอร์ที่มีชื่อ (Named constructors) และแฟกทอรีคอนสตรัคเตอร์ (Factory constructors)

4.3.1) การตั้งค่าเริ่มต้นให้เขตข้อมูลโดยอัตโนมัติ: คอนสตรัคเตอร์จะกำหนดค่าเขตข้อมูลให้กับตัวแปรจากอาร์กิวเมนต์ (Arguments) ที่ได้รับเมื่อมีการสร้างอ๊อบเจกต์โดยอัตโนมัติ ยกตัวอย่างการกำหนดเขตข้อมูลโดยมีการรับค่าอาร์กิวเมนต์ดังแสดงในรูปที่ 2.10 (ก) คลาส Draw มีการสร้างคอนสตรัคเตอร์ที่จำเป็นต้องรับค่าอาร์กิวเมนต์จากการสร้างอ๊อบเจกต์ 2 ค่า ในการเริ่มต้นเขตข้อมูล แต่ด้วยคุณลักษณะเฉพาะของดาร์ทสามารถทำให้การสร้างคอนสตรัคเตอร์ทำได้ง่ายขึ้น ดังแสดงในรูปที่ 2.10 (ข) คลาส Draw ในอาร์กิวเมนต์มี this.ซึ่งจะทำให้ค่าเขตข้อมูลจะเริ่มต้นโดยอัตโนมัติ นอกจากนี้ถ้าภายในคอนสตรัคเตอร์นั้นไม่มีการทำงานใด ๆ นอกจากการเริ่มต้นของเขตข้อมูลแล้ว สามารถละตัวของคอนสตรัคเตอร์ได้ โดยการใช้เครื่องหมายเซมิโคลอนแทนด้วยสัญลักษณ์ ; แทนการใช้เครื่องหมายวงเล็บปีกกา แทนด้วยสัญลักษณ์ { }

```
class Draw {
  num x, y;
  Draw (num x, num y){
    this.x = x;
    this.y = y;
  }
}
```

(ก)

```
class Draw {
  num x, y;
//Draw (this.x, this.y){}
  Draw (this.x, this.y);
}
```

(ข)

รูปที่ 2.10 วิธีการสร้างคอนสตรัคเตอร์

4.3.2) คอนสตรัคเตอร์ที่มีชื่อ: บางครั้งในหนึ่งคลาสอาจมีการสร้างคอนสตรัคเตอร์ได้มากกว่าหนึ่งคอนสตรัคเตอร์ แต่เนื่องจากดาร์ทไม่สนับสนุนการโอเวอร์โหลด (Overloading) (โดยทั่วไปแล้วภาษาที่มีตัวแปรแบบเชิงพลวัตจะไม่มีการสนับสนุนการโอเวอร์โหลด) ดาร์ทจึงได้นำเสนอวิธีในการตั้งชื่อให้กับคอนสตรัคเตอร์ เพื่อให้ในหนึ่งคลาสที่จะสามารถสร้างคอนสตรัคเตอร์ได้มากกว่าหนึ่งคอนสตรัคเตอร์ ดังแสดงในรูปที่ 2.11 แสดงตัวอย่างการสร้างคอนสตรัคเตอร์แบบระบุชื่อทั้งหมด 3 คอนสตรัคเตอร์ คือ คอนสตรัคเตอร์ Draw() คอนสตรัคเตอร์ Draw.pointZero() และคอนสตรัคเตอร์ Draw.pointDouble(num x, num y)

```
class Draw {
  num x, y;
  Draw(this.x, this.y);
  Draw.pointZero(){
    x = 0; y = 0;
  }
  Draw.pointDouble(num x, num y){
    x = x * x;
    y = y * y;
  }
}
```

รูปที่ 2.11 วิธีการตั้งชื่อให้กับคอนสตรัคเตอร์

```
class Name {
  String fname;
  String lname;

  factory Name(String name) {
    var parts = name.split(' ');
    return new Name.parts(parts[0], parts[1]);
  }

  Name.parts(this.fname, this.lname);
}

void main() {
  Name name = new Name("Naritsara Tharaputh");
  print('${name.fname} ${name.lname}');
}
```

```
Output:
        Naritsara Tharaputh
```

รูปที่ 2.12 วิธีการสร้างแฟกทอรีคอนสตรัคเตอร์

4.3.3) แฟกทอรีคอนสตรัคเตอร์: คอนสตรัคเตอร์ทั่วไปจะไม่สามารถส่งค่ากลับได้ แต่ในดาร์ทสามารถกำหนดให้คอนสตรัคเตอร์นั้นส่งค่ากลับได้โดยการประกาศคำสงวน factory ไว้ด้านหน้าของคอนสตรัคเตอร์ที่ต้องการให้มีการส่งค่ากลับ ดังแสดงในรูปที่ 2.12 เป็นแฟกทอรีคอนสตรัคเตอร์ที่ส่งค่าอ๊อบเจกต์ของตัวแปรกลับ ส่วนการเรียกใช้ แฟกทอรีนั้นสามารถเรียกได้เหมือนกับการเรียกใช้คอนสตรัคเตอร์ทั่วไป

5. การห่อหุ้ม

การห่อหุ้ม (Encapsulation) ในภาษาดาร์ทมีลักษณะเหมือนภาษาเชิงอ๊อบเจกต์ทั่วไป แต่ในการกำหนดค่าดาร์ทช่วยให้สามารถทำได้ง่ายขึ้น เพราะมีคำสงวน get และ set ให้สามารถทำการกำหนดและเรียกใช้ค่าโดยไม่จำเป็นต้องสร้างเมธอดเอง ตัวอย่างการห่อหุ้มดังแสดงใน คลาส Encap มีตัวแปร 2 ตัว คือ _name และ _age ที่มีการประกาศไว้เป็นส่วนตัว (Private) ใช้ get สำหรับการคืนค่ากลับเมื่อมีการเรียกใช้ และ set เพื่อกำหนดค่าให้กับตัวแปรนั้น ฟังก์ชัน main() มีการสร้างอ๊อบเจกต์ของคลาส Encap ให้กับตัวแปร encap เมื่อต้องการกำหนดค่าให้กับตัวแปร _name ในคลาส Encap สามารถทำได้โดยการกำหนดค่าให้กับ encap.Name และการเรียกใช้ก็สามารถเรียกใช้ผ่าน encap.Name

```
class Encap {
  var _name;
  var _age;

  String get Name            => _name;
         set Name(var nName) => _name = nName;
  int    get Age             => _age;
         set Age(int nAge)   => _age = nAge;
}
void main() {
  Encap encap = new Encap();
  encap.Name = "Naritsara Tharaputh";
  encap.Age = 24;
  print('Name: ${encap.Name}');
  print('Age : ${encap.Age}');
}
```

```
Output:
        Name: Naritsara Tharaputh
        Age : 24
```

รูปที่ 2.13 การห่อหุ้มตัวแปรในคลาส

```
abstract class Animal {
  String eat() {
    return 'Eat a food.';
  }
  String noise();
}

class Cat extends Animal {
  String noise() {
    Return 'Mew Mew!!';
  }
}

void main() {
  Cat cat = new Cat();
  print('Cat: ${cat.eat()}');
  print('Cat: ${cat.noise()}');
}
```

```
Output:
        Cat: Eat a food.
        Cat: Mew Mew!!
```

รูปที่ 2.14 การสืบทอดคลาส

6. การสืบทอด

ดาร์ทมีการสืบทอด (Inheritance) แบบเดี่ยว คือ การสร้างอ๊อบเจกต์ให้มีความสัมพันธ์เป็นลำดับชั้นแบบต้นไม้ คลาสที่เป็นต้นแบบจะถูกเรียกว่า ซูเปอร์คลาส (Superclass) หรือคลาสแม่

และคลาสที่ทำการสืบทอดจากคลาสแม่จะถูกเรียกว่า ซับคลาสหรือคลาสลูก ยกตัวอย่างการสืบทอดดังแสดงในรูปที่ 2.14

คลาสแม่คือคลาส Animal และคลาสลูกคือคลาส Cat มีการใช้คำสงวน extends เพื่อบอกว่าคลาส Cat สืบทอดจากคลาส Animal ทำให้คลาส Cat มีการสืบทอดฟังก์ชัน noise() จากคลาส Animal แต่ถูกทำให้เฉพาะเจาะจงมากขึ้น

7. การมีหลายรูปแบบ

การมีหลายรูปแบบ (Polymorphism) คือ การสร้างฟังก์ชันหรือเมธอดที่มีพารามิเตอร์เป็นแบบชนิดของคลาสแม่ ดังแสดงใน รูปที่ 2.15 ฟังก์ชัน callPet(Animal pet) มีการกำหนดพารามิเตอร์เป็นแบบชนิดของคลาสแม่ ภายใน main() มีการสร้างอ๊อบเจกต์ของคลาส Cat และส่งอินสแตนซ์ของอ๊อบเจกต์ Cat ให้กับฟังก์ชัน callPet(Animal pet) จะเป็นผลให้ตัวแปร pet ในฟังก์ชัน callPet(Animal pet) จะเปลี่ยนตัวแปร pet จาก Animal เป็น Cat โดยอัตโนมัติ

```
abstract class Animal {
  String eat() => 'Eat a food.';
  String noise();
}

class Cat extends Animal {
  String noise() => 'Mew Mew!!';
}

class Dog extends Animal {
  String noise() => 'Box Box!!';
}

String petEat(Animal pet) => pet.eat();

String callPet(Animal pet)=> pet.noise();

void main() {
  Cat cat = new Cat();
  print('Cat: ${callPet(cat)}');
  print('Cat: ${petEat(cat)}');
}
```

```
Output:
        Cat: Mew Mew!!
        Cat: Eat a food.
```

รูปที่ 2.15 การมีหลายรูปแบบ

(ก)

```
abstract class Animal {
  String eat() => 'Eat a food.';
  String noise();
}

class Cat implements Animal{
  String eat()    =>
                  'Eat a \'Whiskas\'.';
  String noise() => 'Mew Mew!!';
}


class Dog extends Animal {
  String noise() => 'Box Box!!';
}

void main() {
  Cat cat = new Cat();
  Dog dog = new Dog();
  print('Cat: ${cat.eat()}');
  print('Cat: ${cat.noise()}');
  print('Dog: ${dog.eat()}');
  print('Dog: ${dog.noise()}');
}
```

```
Output:
        Cat: Eat a 'Whiskas'.
        Cat: Mew Mew!!
        Dog: Eat a food.
        Dog: Box Box!!
```

(ข)

```
abstract class Pet{
  factory Pet(){
   print('I\'m factory Pet() method');
  }
  String isPet();
}

abstract class Animal {
  String eat() => 'Eat a food.';
  String noise();
}

class Cat implements Animal, Pet {
  String eat()    =>
                   'Eat a \'Whiskas\'';
  String noise() => 'Mew Mew!!';
  String isPet() => 'It\'s a pet.';
}

class Dog extends Animal {
  String noise() => 'Box Box!!';
}

void main() {
  Dog dog = new Dog();
  Pet pet = new Pet();
}
```

```
Output:
        I'm factory Pet() method
```


รูปที่ 2.16 คลาสนามธรรม

8. คลาสนามธรรม

คลาสนามธรรม (Abstract class) ในภาษาดาร์ทเป็นตัวช่วยในการกำหนดอินเตอร์เฟซ ซึ่งการสร้างคลาสนามธรรมส่วนใหญ่มักจะพบการสร้างเมธอดนามธรรมด้วย ปกติคลาสนามธรรมจะไม่สามารถสร้างอินสแตนซ์ได้ (อินสแตนซ์ คืออ๊อบเจกต์ที่ถูกสร้างจากคลาส) แต่ถ้าหากมีความต้องการสร้างอินสแตนซ์สามารถทำได้โดยการใช้ลักษณะของแฟกทอรีคอนสตรัคเตอร์ การนำคลาสนามธรรมไปใช้สามารถทำได้โดยการสืบทอด หรืออิมพลีเมนต์ ดังแสดงในรูปที่ 2.16 (ก) คลาส Animal เป็นคลาสนามธรรมที่ไม่สามารถดำเนินการสร้างอินสแตนซ์ได้ คลาส Cat เป็นการอิมพลีเมนต์คลาส Animal จึงจำเป็นต้องทำการโอเวอร์ไรด์ (Override) เมธอดทั้งหมด ส่วนคลาส Dog ใช้การสืบทอดในลักษณะ extends จึงทำการโอเวอร์ไรด์เฉพาะเมธอดที่ต้องการให้มี

ความเฉพาะเจาะจงมากยิ่งขึ้นคือเมธอด noise() รูปที่ 2.16 (ข) มีการเพิ่มในส่วนของคลาส Pet ซึ่งเป็นคลาสนามธรรมที่สามารถดำเนินการสร้างอินสแตนซ์ได้ โดยการใช้แฟกทอรีคอนสตรัคเตอร์ในคลาสนามธรรม และคลาส Cat สามารถอิมพลีเมนต์ทั้งคลาส Animal และคลาส Pet ได้

```
void main(){
  var string1 =
      'I\'m "single" line';
  print (string1);
}
```

```
Output:
       I'm "single" line
```

(ก)

```
void main(){
  var string2 =
       "I'm 'single' line";
  print (string2);
}
```

```
Output:
       I'm 'single' line
```

(ข)

รูปที่ 2.17 วิธีการสร้างสตริงแบบบรรทัดเดียว

```
void main(){
  var string3 = '''
                  I'm
                  multiple
                  lines ''';
  print(string3);
}
```

```
Output:
       I'm
       multiple
       lines
```

(ก)

```
void main(){
  var string4 = """
                  I'm
                  multiple
                  lines """;
  print(string4);
 }
```

```
Output:
       I'm
       multiple
       lines
```

(ข)

รูปที่ 2.18 วิธีการสร้างสตริงแบบหลายบรรทัด

9. สตริง

การสร้างสตริง ในดาร์ทสำหรับการสร้างสตริงแบบบรรทัดเดียวสามารถสร้างได้ด้วยเครื่องหมายอัญประกาศเดี่ยว แทนด้วยสัญลักษณ์ ‘ ’ ดังแสดงในรูปที่ 2.17 (ก) หรือเครื่องหมายอัญประกาศคู่ แทนด้วยสัญลักษณ์ “ ” ดังแสดงในรูปที่ 2.17 (ข) และสร้างสตริงแบบหลายบรรทัดได้ด้วยเครื่องหมายอัญประกาศเดี่ยวสามครั้ง แทนด้วยสัญลักษณ์ ‘‘‘ ’’’ ดังแสดงในรูปที่ 2.18 (ก) หรืออัญประกาศคู่สามครั้ง แทนด้วยสัญลักษณ์ “ “ “ ” ” ” ดังแสดงในรูปที่ 2.18 (ข)

การแทรกนิพจน์ในสตริง ถือว่าเป็นสิ่งที่จำเป็นเมื่อต้องการสร้างสตริงจากค่าอื่น ๆ โดยการใช้เครื่องหมายดอลลาร์ แทนด้วยสัญลักษณ์ $ ตามด้วยตัวแปร หรือแทรกนิพจน์โดยการวางไว้ในเครื่องหมายวงเล็บปีกกาดังแสดงในรูปที่ 2.19

```
import 'dart:math';

void main(){
  var name = 'Nan';
  var greeting = 'Hi!';
  print('$greeting, $name');

  var weight = 46 ;
  var tall = 1.52;
  print('Your BMI is ${weight/(pow(tall,2))}');
}
```

```
Output:
       Hi!, Nan
       Your BMI is 19.909972299168974
```

รูปที่ 2.19 วิธีการแทรกสตริง

10. ตัวกำกับแบบชนิด

ตัวกำกับแบบชนิด (Type annotations) คือ การอธิบายชนิดของตัวแปรเพิ่มเติม หรือการระบุแบบชนิด ซึ่งภาษาดาร์ทนั้นเป็นภาษาที่มีแบบชนิดเชิงทางเลือก (เช่น var name) แต่ในขณะเดียวกันก็สามารถระบุแบบชนิดได้ (เช่น String name) ทำให้ไม่จำเป็นที่จะต้องมีคำอธิบายประกอบแบบชนิดเหมือนกับภาษาจาวาสคริปต์ ยกตัวอย่างดังแสดงในรูปที่ 2.20 เป็นการแสดงให้เห็นการเขียนโค้ดโดยการอธิบายแบบชนิดของฟังก์ชัน ตัวแปร และการส่งค่ากลับใน (ก) จาวาสคริปต์ และ (ข) ในดาร์ทตามลำดับ

```javascript
/**
 * @param {String} name
 * @return {String}
 */
makeGreeting = function(name) {
  /** @type {String} */
  var greeting = 'Hi!, ' + name;
  return greeting;
}
```

(ก) จาวาสคริปต์

```dart
String MakeGreeting (String
name){
  String greeting = 'Hi!, $name';
  return greeting;
}
```

(ข) ดาร์ท

รูปที่ 2.20 วิธีการเขียนที่อธิบายแบบชนิดของฟังก์ชัน ตัวแปร และการส่งค่ากลับ

```dart
class Button {
  num bottom, left, width, height;

  Button (this.bottom, this.left, this.width, this.height);

  num get top               => bottom + height;
      set top(num value)    => bottom = value - height;
  num get right             => left + width;
      set right(num value)  => left = value - width;
}

void main (){
  var button = new Button(3, 4, 20, 15);
  print('top   : ${button.top}');
  print('right: ${button.right}');
}
```

```
Output:
       top   : 18
       right: 24
```

รูปที่ 2.21 วิธีการใช้เกทเทอร์และเซตเทอร์ในการกำหนดเขตข้อมูล

11. เขตข้อมูล เกทเทอร์ และเซตเทอร์

ในหลาย ๆ ภาษาการทำงานจะเกิดขึ้นได้นั้นก็ต่อเมื่อมีการกำหนดเขตข้อมูล (Fields) จริงในคลาส แต่ดาร์ทสามารถช่วยให้เรากำหนดเมธอดที่เหมือนกับเป็นการเข้าถึงคุณสมบัติได้ แต่จะมีการทำงานก็ต่อเมื่อมีการเรียกใช้เท่านั้น ซึ่งเรียกว่าเกทเทอร์ (Getters) และเซตเทอร์ (Setters) ดังแสดงในรูปที่ 2.21 คลาส Button สร้างคอนสตรัคเตอร์สำหรับรับพารามิเตอร์

4 ตัว เพื่อนำไปใช้ในการกำหนดเขตข้อมูลให้กับ top และ right โดยการใช้ get และ set ในการกำหนดคุณสมบัติของ top และ right เพื่อนำไปใช้งานจริง

12. การกำหนดในระดับบนสุด

ดาร์ทเป็นภาษาที่มีอิสระในการกำหนดฟังก์ชัน ตัวแปร เกทเทอร์ และเซทเทอร์ไว้ในตำแหน่งบนสุด (Top-level) ได้ เพราะเป็นภาษาที่ไม่ได้ถูกออกแบบให้การกำหนดทุกอย่างไว้ภายในคลาสดังแสดงในรูปที่ 2.22 แต่โดยส่วนใหญ่แล้วมักจะพบว่าจะทำการกำหนดฟังก์ชัน main() ไว้ในระดับบนสุด

```
import 'dart:math';

final TWO_E = E * 2;

num abs(num value) => value < 0 ? -value : value;

String get toDay{
  final date = new DateTime.now();
  final today = '$date';
  return today;
}

void main (){
  print(TWO_E);
  print (toDay);
}
```

```
Output:
       5.43656365691809
       2013-06-10 14:00:04.994
```

รูปที่ 2.22 วิธีการกำหนดฟังก์ชันระดับบนสุดที่ไม่ใช่ฟังก์ชัน main()

13. ตัวดำเนินการ

ตัวดำเนินการ (Operators) ในภาษาดาร์ทมีลักษณะที่คล้ายกับภาษาอื่น ๆ เช่น ภาษาจาวา และยังมีการจัดความสำคัญของตัวดำเนินการเช่นเดียวกับภาษาอื่น ๆ เช่นกัน แต่ในดาร์ทนิพจน์ใช้ตัวดำเนินการในการเรียกเมธอด นอกจากจะมีการใช้ตัวดำเนินการทำงานตามปกติแล้ว ยังสามารถทำการโอเวอร์ไรด์ ตัวดำเนินการได้อีกด้วย (การโอเวอร์ไรด์ในภาษาดาร์ทจะเหมือนกับใน

ภาษาจาวา) ยกตัวอย่างการโอเวอร์ไรด์ตัวดำเนินเครื่องหมายลบ ดังแสดงในรูปที่ 2.23 แต่ในการโอเวอร์ไรด์ตัวดำเนินการที่กำหนดเองนั้นก็ไม่ควรมีการเปลี่ยนแปลงรูปแบบมากเกินไป

```
class Vector {
  num x, y;
  Vector(this.x, this.y);
  // Overrides - (a - b).
  operator -(Vector other) => new Vector(x - other.x, y - other.y);
}

void main(){
  var position = new Vector(3, 4);
  var velocity = new Vector(1, 2);
  var newPosition = position - velocity;

  // a - b
  assert((3 - 1) == 2 && (4 - 2) == 2);
  // Overrides - (a - b).
  assert((position - velocity).x == 2 && (position - velocity).y == 2);
}
```

รูปที่ 2.23 วิธีการโอเวอร์ไรด์ตัวดำเนินการเครื่องหมายลบ

14. ความเท่าเทียม

ดาร์ทมีตัวดำเนินการที่เกี่ยวกับความเท่าเทียม (Equality) 2 ชนิดคือ สัญลักษณ์ == สำหรับการทดสอบการเท่ากันของค่าที่นำมาเปรียบเทียบว่าเท่ากัน และสัญลักษณ์ != สำหรับการทดสอบการเท่ากันของค่าที่นำมาเปรียบเทียบว่าไม่เท่ากัน (โดยที่สัญลักษณ์ == สามารถทำการโอเวอร์ไรด์ตัวดำเนินการได้ แต่สัญลักษณ์ != ไม่สามารถทำการโอเวอร์ไรด์ตัวดำเนินการได้) และมีฟังก์ชันระดับบนที่เรียก identical() สำหรับการทดสอบว่าอ๊อบเจกต์ทั้งสองนั้นเป็นอ๊อบเจกต์เดียวกันแน่นอน ตัวอย่างการทดสอบความเท่าเทียมดังแสดงในรูปที่ 2.24 ผลลัพธ์ของการใช้ identical() เป็น false เพราะว่าอินสแตนซ์ของ lang และ lang2 แตกต่างกัน

```
class Language {
  String lang = 'Dart';
  Language(){
    this.lang;
  }
}

void main() {
  var s = "Dart language";
  assert(s == "Dart language");
  assert(s != "DartLanguage");

  var lang  = new Language();
  var lang2 = new Language();
  print(identical(lang, lang2));
}
```

```
Output:
       false
```

รูปที่ 2.24 วิธีการทดสอบความเท่าเทียม

15. คอมเมนต์

การสร้างเอกสารของดาร์ทด้วย Dartdoc จะมีการคอมเมนต์ (Comments) ที่น้อยมากเมื่อเทียบกับจาวา ดังแสดงในรูปที่ 2.25 (ก) เป็นการเขียนคอมเมนต์เพื่อสร้างเอกสารของจาวา และรูปที่ 2.25 (ข) เป็นการเขียนคอมเมนต์เพื่อสร้างเอกสารของดาร์ท ดาร์ทสามารถเขียนคอมเมนต์แบบอินไลน์ (Inline) และไม่จำเป็นที่จะต้องทำการฝังแท็ก HTML ในเอกสาร

(ก) จาวา

```
/**
 * Returns an Beers object that can then be petted.
 * The age argument must specify an non-zero integer. The amount
 * argument is the amount of {@link Money} paid for the beers.
 * <p>
 * This method throws {@link NoMoreBeersException} is thrown
 * if there are no more Beers to purchase.
 * {@link IllegalArgumentException} is
 * thrown if age is less than zero.
 *
 * @param  age  a non-zero age
 * @param  amount the amount of money paid for the beers
 * @throws NoMoreBeerssException if there are no more beers available
 * @throws IllegalArgumentException if age is less than zero
 * @return       the beers
 * @see          Farmer
 */
public Beers buyBeers (Age age, Money amount) {
  // ...
}
```

(ข) ดาร์ท

```
/**
 * Returns a Beers that can be petted.
 * An [ArgumentError] is thrown if age is less than zero, and
 * [NoMoreBeersError] is thrown if they are all out of beers.
 */
Beers buyBeers(int age, Money amount) {
  // ...
}
```

รูปที่ 2.25 วิธีการใช้คอมเมนต์เพื่อสร้างเอกสาร (Walrath and et al., 2013c)

2.2 ดีไซน์แพทเทิร์น

ในการออกแบบซอฟต์แวร์เชิงวัตถุเพื่อให้สามารถนำซอฟต์แวร์เชิงวัตถุนั้นกับมาใช้ซ้ำได้เป็นเรื่องที่ยากกว่าการออกแบบซอฟต์แวร์เชิงวัตถุโดยทั่วไป ดังนั้นการบันทึกประสบการณ์ที่เกี่ยวข้องกับการออกแบบซอฟต์แวร์เชิงวัตถุ และการนำการออกแบบนั้นไปใช้ในการเขียนโปรแกรมได้อย่างมีประสิทธิภาพจึงถือเป็นสิ่งที่สำคัญ ดีไซน์แพทเทิร์นจึงเป็นทางเลือกสำหรับการออกแบบที่ทำให้ซอฟต์แวร์สามารถนำกลับมาใช้ใหม่ได้อย่างมีประสิทธิภาพ (Gamma and et al., 1995)

Erich Gamma และคณะ (GoF) ได้ทำการรวบรวมการออกแบบซอฟต์แวร์เชิงวัตถุและจัดกลุ่มของดีไซน์แพทเทิร์นเพื่อให้ง่ายต่อการศึกษา ดังแสดงในตารางที่ 2.2 เป็นการจัดกลุ่มของดีไซน์แพทเทิร์นใช้เกณฑ์การแบ่ง 2 เกณฑ์ด้วยกันคือ

1) จุดประสงค์ เพื่อสะท้อนให้เห็นถึงลักษณะของแพทเทิร์น โดยแบ่งออกเป็น 3 ลักษณะต่อไปนี้คือ เชิงการสร้าง (Creational) เชิงโครงสร้าง (Structural) และเชิงพฤติกรรม (Behavioral)
2) ขอบเขต เพื่อเป็นการระบุว่าแพทเทิร์นสามารถนำไปใช้ได้กับคลาสหรืออ๊อบเจกต์

และ GoF ยังได้ทำการอธิบายรายละเอียดต่าง ๆ ของดีไซน์แพทเทิร์นทั้ง 23 แพทเทิร์นไว้ดังต่อไปนี้

ตารางที่ 2.2 การจัดกลุ่มของดีไซน์แพทเทิร์นตามจุดประสงค์และขอบเขต

จุดประสงค์ / ขอบเขต	การสร้างอ๊อบเจกต์	โครงสร้าง	พฤติกรรม
คลาส	เมธอดโรงงาน	ตัวแปลง	ตัวแปล เมธอดแม่แบบ
อ๊อบเจกต์	โรงงานเชิงนามธรรม ตัวสร้าง ต้นแบบ เดี่ยว	ตัวแปลง บริดจ์ เชิงประกอบ ตัวตกแต่ง ส่วนหน้า น้ำหนักเบา ตัวแทน	ห่วงโซ่ความรับผิดชอบ คำสั่ง ตัววนซ้ำ ตัวกลาง ที่ระลึก ตัวสังเกตการณ์ สถานะ กลยุทธ์ ตัวเยี่ยมเยือน

2.2.1 แพตเทิร์นเชิงการสร้าง

แพตเทิร์นเชิงการสร้าง (Creational patterns) จะช่วยให้อิสระกับระบบที่เกี่ยวกับวิธีการที่อ๊อบเจกต์จะถูกสร้างขึ้น ประกอบขึ้น และการเป็นตัวแทน เพื่อให้มีความยืดหยุ่นมากในสิ่งที่ได้รับการสร้าง ผู้ที่จะสร้าง วิธีการในการสร้าง และได้รับการสร้างเมื่อใด ซึ่งจะช่วยให้สามารถกำหนดค่าของระบบด้วยอ๊อบเจกต์ผลิตภัณฑ์ (Product) ที่แตกต่างกันในฟังก์ชันและโครงสร้าง มีการนำเสนอแพทเทิร์นเชิงการสร้าง 5 แพทเทิร์น ดังต่อไปนี้

1. แพทเทิร์นโรงงานเชิงนามธรรม

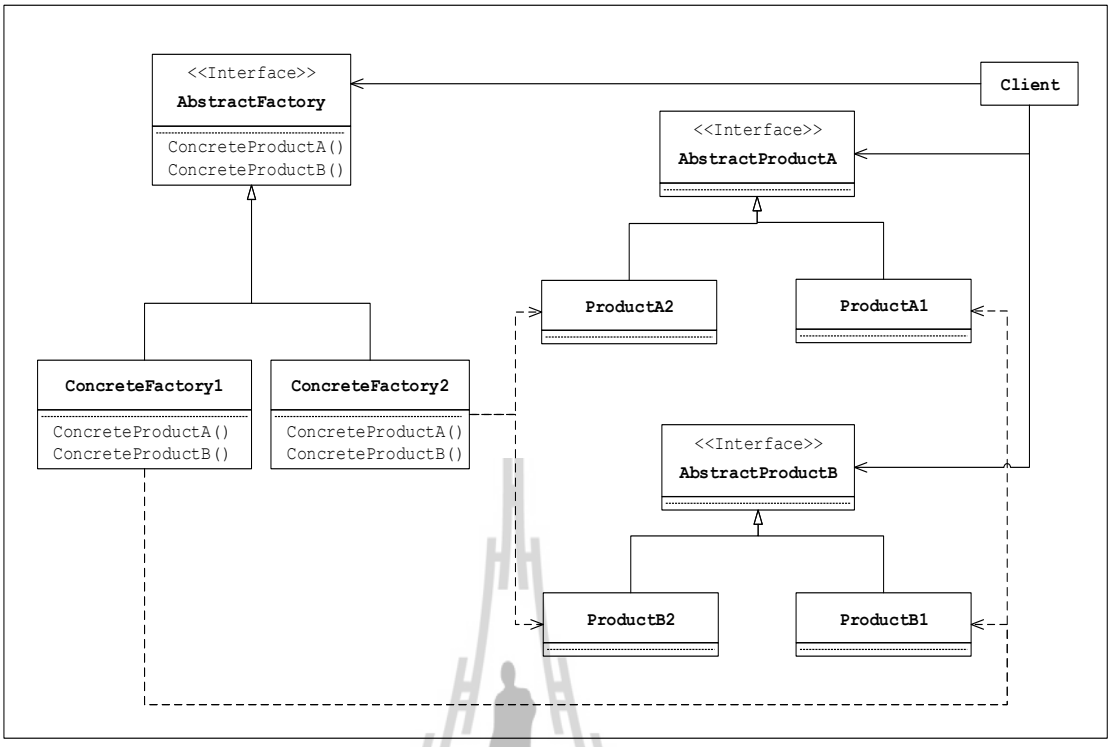
แพทเทิร์นโรงงานเชิงนามธรรม (Abstract factory pattern) มีวัตถุประสงค์หลัก เพื่อประกาศอินเตอร์เฟซ (Interface) สำหรับอ๊อบเจกต์ที่มีการขึ้นต่อกัน หรืออ๊อบเจกต์ที่เกี่ยวข้องกันโดยที่ไม่ต้องทำการระบุคลาสอย่างเป็นรูปธรรม (Concrete classes)

แนวคิดและโครงสร้าง

เมื่อพิจารณาชุดเครื่องมือส่วนติดต่อผู้ใช้ (User interface) ที่มีการใช้หลักการของ “รูปลักษณ์และการใช้งาน” (Look and feel) ในการกำหนดพฤติกรรมและลักษณะของการแสดงผลต่าง ๆ เช่น ปุ่ม (Buttons) แถบเลื่อน (Scroll bars) ที่มีรูปร่างในการแสดงผลที่แตกต่างกันเมื่อมีการทำงานอยู่บนระบบปฏิบัติการที่แตกต่างกัน ในการสร้างเครื่องมือที่ใช้หลักการของรูปลักษณ์และการใช้งาน ไม่ควรพัฒนาซอฟท์แวร์ โดยการแนบอินพุทและการตั้งค่าต่าง ๆ เอาไว้ในรหัสต้นฉบับของโปรแกรมโดยตรง (Hard-code) เพราะหากมีการเปลี่ยนแปลงรูปลักษณ์และการใช้งานของส่วนติดต่อผู้ใช้ภายหลังจะทำให้การเปลี่ยนแปลงทำได้ยาก

แพทเทิร์นโรงงานเชิงนามธรรมช่วยแก้ไขปัญหาดังกล่าวโดยการกำหนดคลาสแบบนามธรรมที่ทำการประกาศเป็นอินเตอร์เฟซ ไว้สำหรับการสร้างเครื่องมือพื้นฐานในแต่ละชนิด และมีซับคลาสที่เป็นตัวตัดสินใจว่าจะทำการอิมพลีเมนต์เครื่องมือชนิดไหน

โครงสร้างแผงผังคลาสของแพทเทิร์นโรงงานเชิงนามธรรมดังแสดงในรูปที่ 2.26



รูปที่ 2.26 โครงสร้างของแพทเทิร์นโรงงานเชิงนามธรรม

2. แพทเทิร์นตัวสร้าง

แพทเทิร์นตัวสร้าง (Builder pattern) มีวัตถุประสงค์หลัก เพื่อให้กระบวนการที่มีโครงสร้าง (Construction) เหมือนกัน สามารถสร้างตัวแทนที่แตกต่างกันได้ โดยการแยกโครงสร้างของอ๊อบเจกต์ที่มีความซับซ้อนมากออกจากตัวแทน

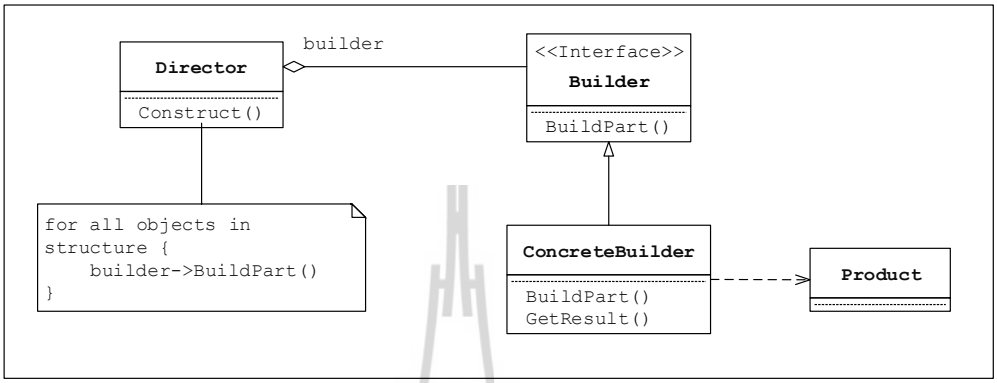
แนวคิดและโครงสร้าง

เมื่อพิจารณาตัวอ่านเอกสาร (Reader) สำหรับเอกสารพีดีเอฟ (Portable Document Format : PDF) ควรที่จะสามารถแปลงเอกสารพีดีเอฟไปเป็นรูปแบบข้อความอื่น ๆ ได้ เช่นการแปลงให้เป็นเอกสารข้อความ (ไฟล์นามสกุล .txt) ดังนั้นการเพิ่มการแปลงใหม่ควรจะทำได้ง่ายโดยที่ไม่ต้องทำการปรับปรุงตัวอ่านนั้นถือเป็นเรื่องที่สำคัญมากเพื่อแก้ไขปัญหาสำหรับจำนวนของการแปลงที่เป็นไปได้

วิธีการในการแก้ไขปัญหาข้างต้นนั้นสามารถทำได้โดยการกำหนดอินเตอร์เฟซตัวสร้างสำหรับการสร้างผลิตภัณฑ์ (ชนิดของเอกสารที่สามารถทำการแปลงไปได้) ซับคลาสจะทำการอิมพลีเมนต์อินเตอร์เฟซของตัวสร้างที่ประกอบไปด้วยโครงสร้างและส่วนประกอบของ

ผลิตภัณฑ์สุดท้าย เพื่อที่จะให้ได้ผลิตภัณฑ์นั้นกลับมา และไดเรกเตอร์ (Director) จะทำการสร้างอ๊อบเจกต์จากอินเตอร์เฟซของตัวสร้าง

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวสร้างดังแสดงในรูปที่ 2.27



รูปที่ 2.27 โครงสร้างของแพทเทิร์นตัวสร้าง

3. แพทเทิร์นเมธอดโรงงาน

แพทเทิร์นเมธอดโรงงาน (Factory method pattern) มีวัตถุประสงค์หลัก เพื่อเป็นเมธอดสำหรับทำหน้าที่สร้างอ๊อบเจกต์ โดยการกำหนดอินเตอร์เฟซสำหรับการสร้างอ๊อบเจกต์ชนิดหนึ่ง ๆ แต่ซับคลาสจะเป็นผู้กำหนดและคนตัดสินใจว่าจะทำการสร้างอ๊อบเจกต์จากคลาสใด

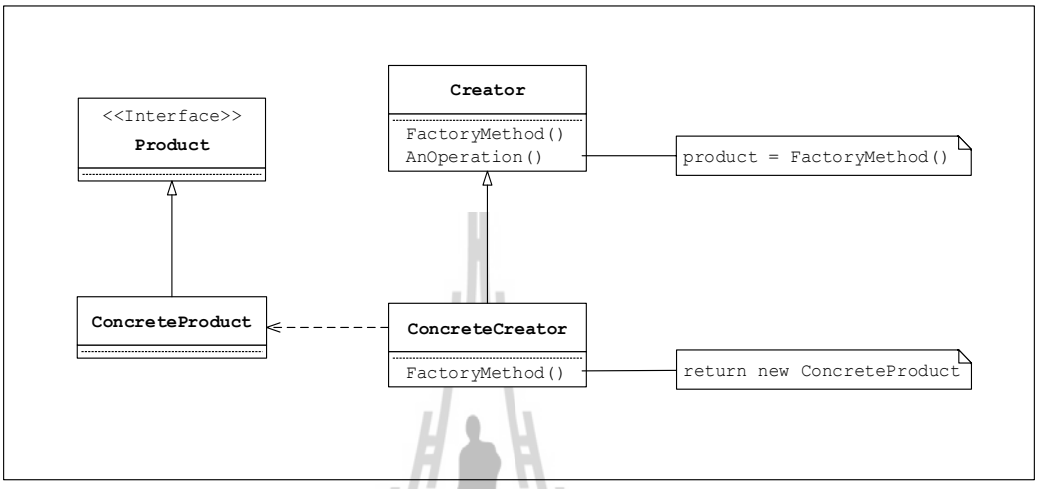
แนวคิดและโครงสร้าง

เมื่อพิจารณากรอบงาน (Framework) ที่รับผิดชอบการสร้างอ๊อบเจกต์ เมื่อกรอบงานใช้คลาสนามธรรมในการกำหนดและปรับปรุงความสัมพันธ์ระหว่างอ๊อบเจกต์ เช่น กรอบงานโปรแกรมประยุกต์สำหรับสร้างเอกสารได้หลายชนิด (รายงานหรือจดหมาย) โดยทั้งสองชนิดจะใช้หน้าของเอกสารที่แตกต่างกัน ในการสร้างเอกสารกรอบงานโปรแกรมประยุกต์จะไม่สามารถรู้ได้ว่าจะต้องสร้างเอกสารชนิดไหน เพราะรู้จักแต่คลาสนามธรรมเท่านั้น ซึ่งทำให้เกิดปัญหาสำหรับกรอบงานเมื่อต้องการสร้างเอกสาร

แพทเทิร์นเมธอดโรงงานนำเสนอวิธีการแก้ไขปัญหาดังกล่าว โดยการห่อหุ้มอ๊อบเจ็กต์ที่มีความรู้ในการสร้างเอกสาร และย้ายออกไปจากกรอบงาน คือมีการกำหนดเมธอดในซับคลาสสำหรับสร้างเอกสาร (ซึ่งก็คือแพทเทิร์นเมธอดโรงงาน) ให้กับกรอบงาน และส่งค่าชนิด

ของเอกสารที่เหมาะสมกลับ

โครงสร้างแผงผังคลาสของแพทเทิร์นเมธอดโรงงานดังแสดงในรูปที่ 2.28



รูปที่ 2.28 โครงสร้างของแพทเทิร์นเมธอดโรงงาน

4. แพทเทิร์นต้นแบบ

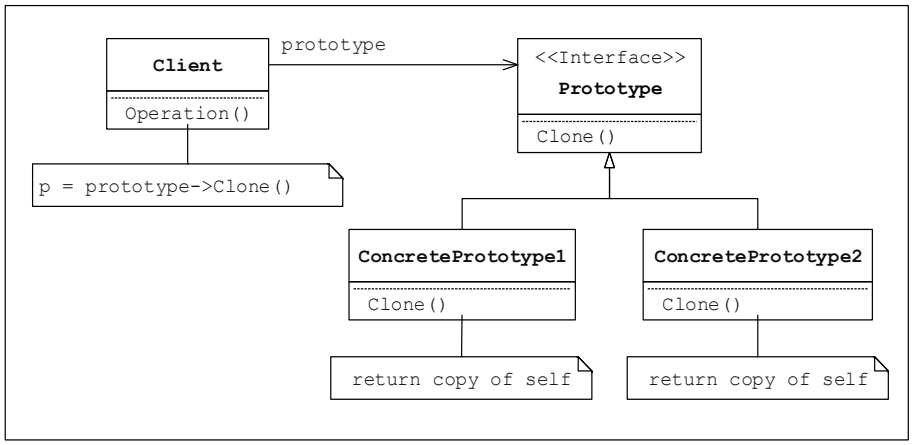
แพทเทิร์นต้นแบบ (Prototype pattern) มีวัตถุประสงค์หลัก เพื่อสร้างอ๊อบเจกต์ใหม่โดยการคัดลอกจากอ๊อบเจกต์ต้นแบบ กล่าวคือเป็นการสร้างอ๊อบเจกต์เพื่อกำหนดต้นแบบเริ่มต้นแล้วสร้างอ๊อบเจกต์ใหม่โดยการคัดลอกจากอ๊อบเจกต์เริ่มต้น

แนวคิดและโครงสร้าง

เมื่อพิจารณาชุดเครื่องมือส่วนติดต่อกับผู้ใช้ในการสร้างปุ่มหรือแถบเลื่อน ด้วยหลักการของรูปลักษณ์และการใช้งาน เมื่อมีความต้องการใช้ปุ่มหรือแถบเลื่อนก็จะทำการสร้างอ๊อบเจกต์ใหม่ทุกครั้ง ทำให้มีการสร้างอ๊อบเจกต์ใหม่บ่อยมาก

แพทเทิร์นต้นแบบช่วยในการแก้ไขปัญหาดังกล่าว โดยการสร้างปุ่มหรือแถบเลื่อนเพื่อให้ได้อินสแตนซ์ต้นแบบในครั้งแรก และเมื่อมีความต้องการใช้งานปุ่มหรือแถบเลื่อนในครั้งต่อไปก็ทำเพียงการโคลนอินสแตนซ์ต้นแบบ

โครงสร้างแผงผังคลาสของแพทเทิร์นต้นแบบดังแสดงในรูปที่ 2.29



รูปที่ 2.29 โครงสร้างแพทเทิร์นต้นแบบ

5. แพทเทิร์นเดี่ยว

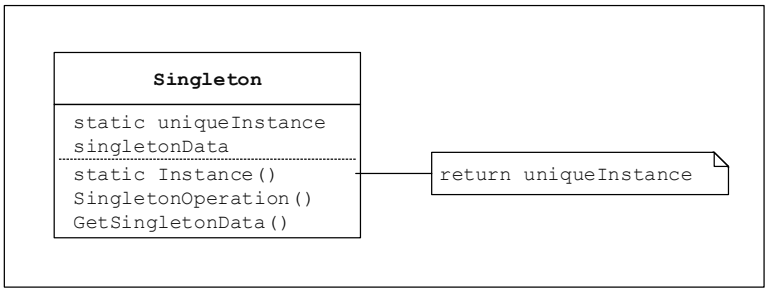
แพทเทิร์นเดี่ยว (Singleton pattern) มีวัตถุประสงค์หลัก เพื่อทำการตรวจสอบให้แน่ใจว่าในคลาสจะมีอินสแตนซ์เพียงอินสแตนซ์เดียว และใช้จุดโกลบอล (Global point) ในการเข้าถึงอินสแตนซ์นั้น

แนวคิดและโครงสร้าง

เมื่อพิจารณาตัวอย่างเช่นเครื่องพิมพ์ในระบบแม้ว่าจะมีหลายตัว แต่ก็ควรจะมีหน่วยความจำที่สร้างขึ้นไว้ภายในเครื่องพิมพ์ (Printer spooler) เดียวเท่านั้น รวมถึงการมีหน้าต่างสำหรับการจัดการกับการสั่งพิมพ์ เพราะสำหรับคลาสบางคลาสแล้วการมีอินสแตนซ์เพียงหนึ่งอินสแตนซ์ถือเป็นสิ่งที่มีความสำคัญมาก เนื่องจากในบางครั้งการมีอินสแตนซ์มากกว่าหนึ่งอินสแตนซ์อาจทำให้เกิดข้อผิดพลาดในการทำงานได้

แพทเทิร์นเดี่ยวได้ให้วิธีในการแก้ปัญหาที่ดีคือ การกำหนดคลาสสำหรับรับผิดชอบและติดตามอินสแตนซ์แต่เพียงผู้เดียว โดยการสกัดกั้น (Intercepting) คำขอในการสร้างวัตถุใหม่เพื่อให้มั่นใจว่าจะไม่มีการสร้างอินสแตนซ์ขึ้นใหม่ได้ และการเข้าถึงอินสแตนซ์ให้เข้าถึงด้วยตัวแปรแบบโกลบอล (Global variable)

โครงสร้างแผงผังคลาสของแพทเทิร์นเดี่ยวดังแสดงในรูปที่ 2.30



รูปที่ 2.30 โครงสร้างแพทเทิร์นเดี่ยว

2.2.2 แพตเทิร์นโครงสร้าง

แพตเทิร์นโครงสร้าง (Structural patterns) เป็นแพทเทิร์นที่เกี่ยวข้องกับวิธีการในการทำให้คลาสและอ๊อบเจกต์ประกอบกันเพื่อสร้างโครงสร้างขนาดใหญ่ สำหรับแพทเทิร์นเชิงโครงสร้างคลาส (Structural class patterns) เป็นการประกอบด้วยอินเตอร์เฟซหรือการอิมพลีเมนต์โดยการใช้การสืบทอด แพทเทิร์นเชิงโครงสร้างอ๊อบเจกต์ (Structural object patterns) เป็นการอธิบายวิธีการประกอบอ๊อบเจกต์โดยต้องคำนึงถึงฟังก์ชันใหม่ การเพิ่มความยืดหยุ่นให้กับองค์ประกอบของอ๊อบเจกต์ มีการนำเสนอแพทเทิร์นเชิงโครงสร้าง 7 แพทเทิร์น ดังต่อไปนี้

1. แพทเทิร์นตัวแปลง

แพทเทิร์นตัวแปลง (Adapter pattern) มีวัตถุประสงค์หลัก เพื่อแปลงอินเตอร์เฟซของคลาสไปยังอินเตอร์เฟซอื่น ๆ ที่ไคลเอนต์คาดหวัง จากเดิมคลาสที่ไม่สามารถทำงานร่วมกันได้เนื่องมาจากอินเตอร์เฟซที่ไม่เข้ากัน ตัวแปลง (Adapter) จะช่วยให้คลาสสามารถทำงานร่วมกันได้

แนวคิดและโครงสร้าง

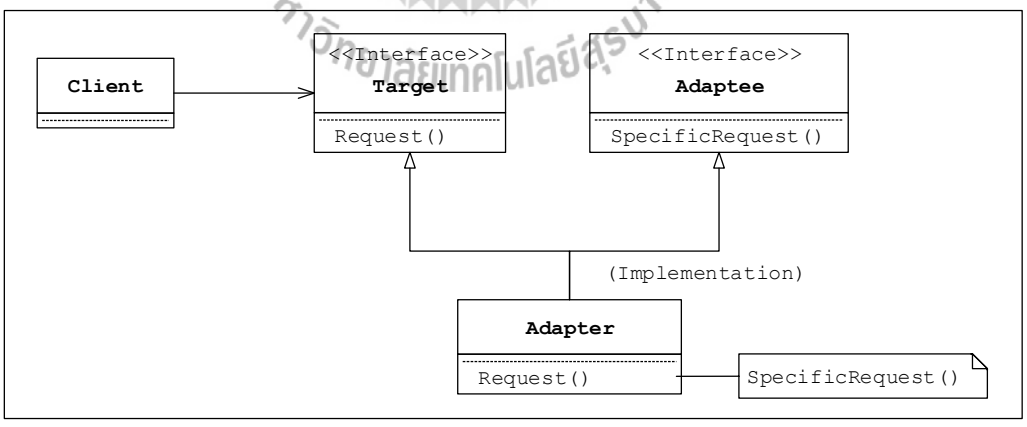
บ่อยครั้งที่ชุดเครื่องมือถูกออกแบบสำหรับการนำโค้ดมาใช้ซ้ำได้ แต่ก็มักจะติดปัญหาตรงที่อินเตอร์เฟซที่ไม่ตรงกับอินเตอร์เฟซโดเมนเฉพาะของโปรแกรมประยุกต์ที่จำเป็นต้องใช้ จึงไม่สามารถนำโค้ดมาใช้ซ้ำได้จริง เมื่อพิจารณาเอดิเตอร์สำหรับวาดภาพ ซึ่งช่วยให้ผู้ใช้วาดหรือจัดองค์ประกอบต่าง ๆ (เช่น เส้น ข้อความ รูปหลายเหลี่ยม ฯลฯ) เป็นแผนภาพหรือรูปภาพ โดยการกำหนดอินเตอร์เฟซสำหรับการสร้างรูปร่าง ซึ่งมีซับคลาสสำหรับการสร้างรูปหลายเหลี่ยม ซับคลาสสำหรับการสร้างเส้นตรง และซับคลาสสำหรับการสร้างข้อความ จะสังเกตเห็นว่าสองซับคลาสแรกเป็นคลาสรูปทรงเรขาคณิตซึ่งง่ายในการดำเนินการ เพราะความสามารถในการวาดและแก้ไขจะถูกกำหนดโดยการสืบทอด แต่ซับคลาสสำหรับการสร้างข้อความ เป็นคลาสที่อิมพลีเมนต์

ได้ยาก เพราะมีความซับซ้อนและยังต้องจัดการกับบัฟเฟอร์ (Buffer) แต่ในขณะเดียวกันชุดเครื่องมืออาจมีการกำหนดอินเตอร์เฟซสำหรับการสร้างข้อความอยู่แล้ว ซึ่งทำให้อาจมีแนวคิดในการนำอินเตอร์เฟซสำหรับการสร้างข้อความ มาใช้ในการอิมพลีเมนต์คลาสสำหรับการสร้างข้อความ แต่ในความเป็นจริงไม่สามารถทำได้เพราะชุดเครื่องมือไม่ได้มีการออกแบบให้สามารถใช้อินเตอร์เฟซของการสร้างรูปร่างสลับกับอินเตอร์เฟซของการสร้างข้อความ

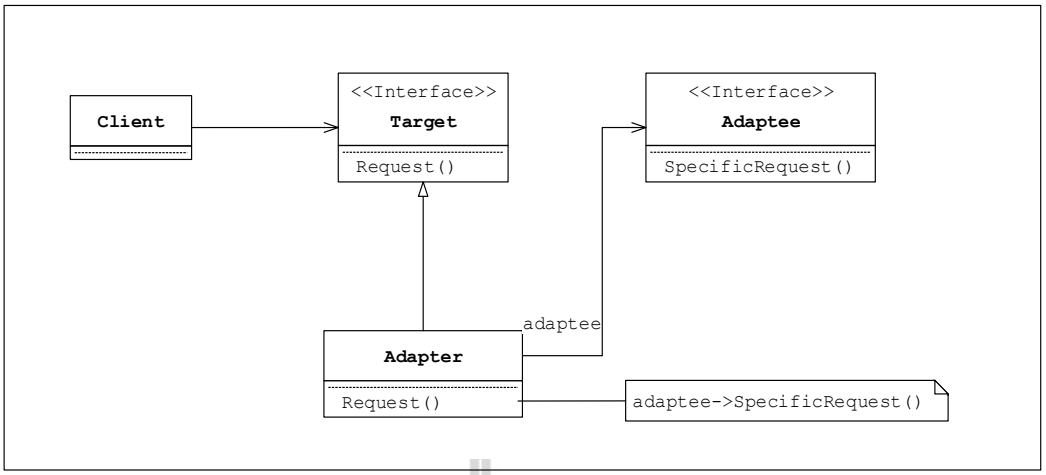
แพทเทิร์นตัวแปลงสามารถกำหนดให้คลาสสำหรับการสร้างข้อความ เป็นตัวแปลงในการที่จะกำหนดอินเตอร์เฟซการสร้างข้อความไปยังอินเตอร์เฟซการสร้างรูปร่าง ได้ด้วยหนึ่งใน 2 วิธีดังต่อไปนี้

1. คลาสตัวแปลง (Class adapter) โดยการสืบทอดอินเตอร์เฟซของการสร้างรูปร่างหรือการอิมพลีเมนต์ของอินเตอร์เฟซการสร้างข้อความ
2. อ๊อบเจกต์ตัวแปลง (Object adapter) โดยการจัดวางองค์ประกอบอินสแตนซ์ของอินเตอร์เฟซการสร้างข้อความไว้ภายในคลาสการสร้างข้อความและอิมพลีเมนต์คลาสการสร้างข้อความในลักษณะอินเตอร์เฟซของการสร้างข้อความ

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวแปลงแบบคลาสตัวแปลงดังแสดงในรูปที่ 2.31 และโครงสร้างแผงผังคลาสของแพทเทิร์นตัวแปลงแบบอ๊อบเจกต์ตัวแปลงดังแสดงในรูปที่ 2.32



รูปที่ 2.31 โครงสร้างแพทเทิร์นตัวแปลงแบบคลาสตัวแปลง



รูปที่ 2.32 โครงสร้างแพทเทิร์นตัวแปลงแบบอ๊อบเจกต์ตัวแปลง

2. แพทเทิร์นบริดจ์

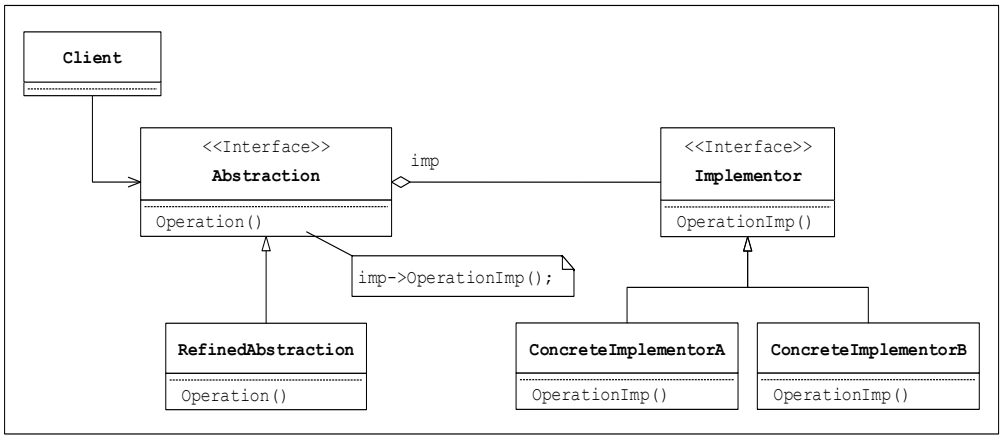
แพทเทิร์นบริดจ์ (Bridge pattern) มีวัตถุประสงค์หลัก ในการแยกคลาสนามธรรมและคลาสการอิมพลีเมนต์ ออกจากกันเพื่อให้ทั้งสองส่วนนั้นสามารถเปลี่ยนแปลงหรือทำงานที่แตกต่างกันได้อย่างเป็นอิสระ

แนวคิดและโครงสร้าง

เมื่อพิจารณาวิธีการทั่วไปในการรองรับเมื่อมีเพียงอินเตอร์เฟซเดียวในการนำไปอิมพลีเมนต์มักจะใช้การสืบทอด การกำหนดอินเตอร์เฟซ และซับคลาสที่ทำการอิพพลีเมนต์ จะถูกพัฒนาด้วยวิธีการที่แตกต่างกันออกไป แต่ในการสืบทอดนั้นจะทำให้อินเตอร์เฟซผูกกับการอิมพลีเมนต์อย่างถาวร ซึ่งจะทำให้ไม่มีความยืดหยุ่นเพียงพอในการที่จะขยาย แก้ไข การนำมาใช้ซ้ำและการใช้งานทั้งสองอย่างอิสระเป็นสิ่งที่ยาก

แพทเทิร์นบริดจ์ช่วยในการแก้ไขปัญหาดังกล่าว คือการทำให้อินเตอร์เฟซและการอิมพลีเมนต์เป็นสะพานเชื่อมต่อกัน ซึ่งจะทำให้ทั้งสองทำงานและเปลี่ยนแปลงได้อย่างอิสระ

โครงสร้างแผงผังคลาสของแพทเทิร์นบริดจ์ดังแสดงในรูปที่ 2.33



รูปที่ 2.33 โครงสร้างแพทเทิร์นบริดจ์

3. แพทเทิร์นเชิงประกอบ

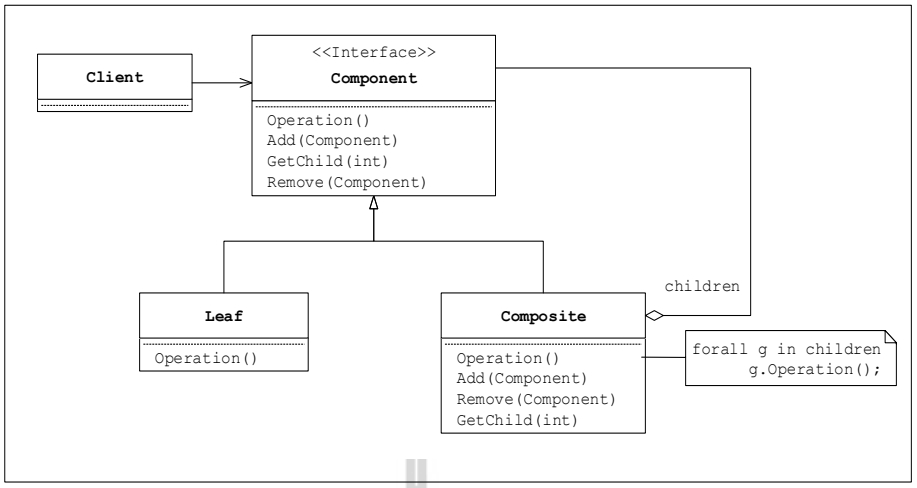
แพทเทิร์นเชิงประกอบ (Composite pattern) มีวัตถุประสงค์หลัก เพื่อช่วยให้ไคลเอนต์ดำเนินการกับแต่ละองค์ประกอบของอ๊อบเจกต์อย่างเท่าเทียมกัน สำหรับอ๊อบเจกต์ที่ภายในมีโครงสร้างประกอบกันเป็นแผนผังรูปต้นไม้แบบลำดับชั้น

แนวคิดและโครงสร้าง

เมื่อพิจารณาเอดิเตอร์สำหรับวาดภาพ สามารถช่วยให้สร้างแผนภาพที่มีความซับซ้อนมากจากองค์ประกอบง่าย ๆ ได้ แต่กลุ่มขององค์ประกอบจะมีขนาดใหญ่ การอิมพลีเมนต์อย่างง่าย ๆ สามารถกำหนดคลาสสำหรับรูปร่างพื้นฐานของรูปทรงเรขาคณิต (Graphical primitives) เพื่อทำหน้าที่เป็นที่เก็บของ (Containers) สำหรับพื้นฐานของรูปทรงเรขาคณิต แต่วิธีการดังกล่าวก็ทำให้เกิดปัญหาคือ โค้ดที่ใช้ในคลาสเหล่านั้นจะต้องรักษาอ๊อบเจกต์ที่แตกต่างกันของพื้นฐานรูปทรงเรขาคณิตและที่เก็บของ การแยกความแตกต่างของอ๊อบเจกต์จะทำให้โปรแกรมประยุกต์เกิดความซับซ้อนมากขึ้น ซึ่งถือว่าเป็นปัญหาสำหรับวิธีการในข้างต้น

แพทเทิร์นเชิงประกอบช่วยแก้ไขปัญหาดังกล่าว โดยการกำหนดให้คลาสนามธรรมนั้นจะต้องแสดงทั้งพื้นฐานของรูปเรขาคณิตและที่เก็บของเหล่านั้น เพราะแพทเทิร์นมีลักษณะของการเรียกตัวเองซ้ำเพื่อให้ไคลเอนต์ไม่สร้างความแตกต่างของอ๊อบเจกต์

โครงสร้างแผงผังคลาสของแพทเทิร์นเชิงประกอบดังแสดงในรูปที่ 2.34



รูปที่ 2.34 โครงสร้างแพทเทิร์นเชิงประกอบ

4. แพทเทิร์นตัวตกแต่ง

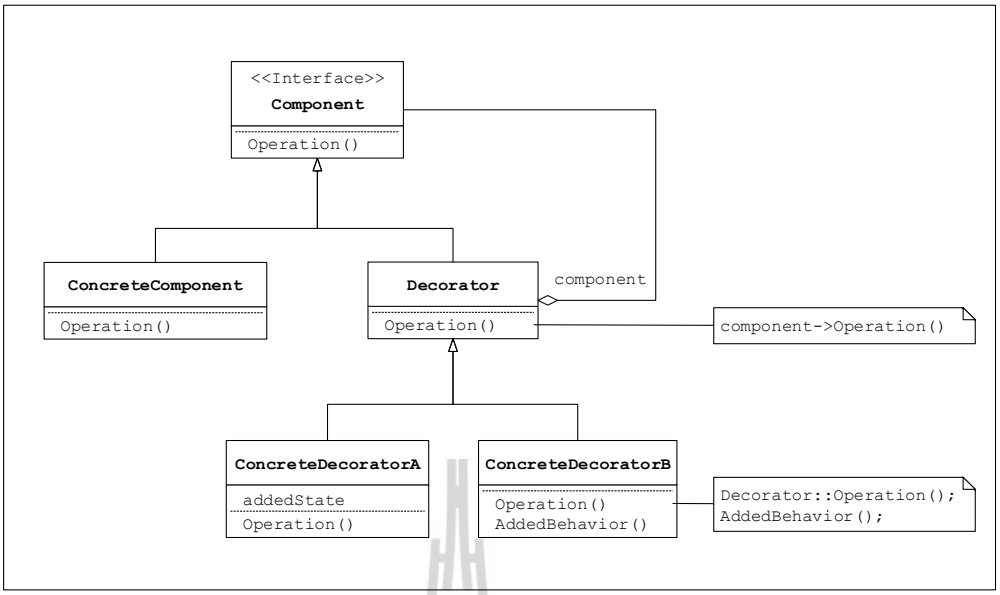
แพทเทิร์นตัวตกแต่ง (Decorator pattern) มีวัตถุประสงค์หลัก เพื่อให้บริการทางเลือกที่มีความยืดหยุ่นของซับคลาสในการการขยายฟังก์ชันในการทำงาน สำหรับการเพิ่มความรับผิดชอบไปยังอ๊อบเจกต์แบบเชิงพลวัต

แนวคิดและโครงสร้าง

เมื่อพิจารณาการเพิ่มความรับผิดชอบให้กับบางอ๊อบเจกต์ในคลาสหนึ่ง ๆ เช่นการเพิ่มคุณสมบัติเส้นขอบ (Border) ไปยังส่วนประกอบของส่วนติดต่อกับผู้ใช้ วิธีการหนึ่งที่จะใช้ในการเพิ่มความรับผิดชอบให้กับอ๊อบเจกต์คือ การใช้คุณสมบัติการสืบทอด โดยการสืบทอดการสร้างเส้นขอบจากคลาสอื่นที่มีอยู่ แต่การสืบทอดเส้นขอบจากคลาสอื่นนี้เป็นการทำงานแบบคงที่ทำให้เกิดความไม่ยืดหยุ่น

แพทเทิร์นตัวตกแต่งช่วยในการแก้ไขปัญหาดังกล่าวให้มีการทำงานแบบเชิงพลวัตและมีความยืดหยุ่นมากขึ้นโดยการปิดล้อม (Enclose) ส่วนประกอบไว้ในอีกอ๊อบเจกต์หนึ่งสำหรับการเพิ่มเส้นขอบ

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวตกแต่งดังแสดงในรูปที่ 2.35



รูปที่ 2.35 โครงสร้างแพทเทิร์นตัวตกแต่ง

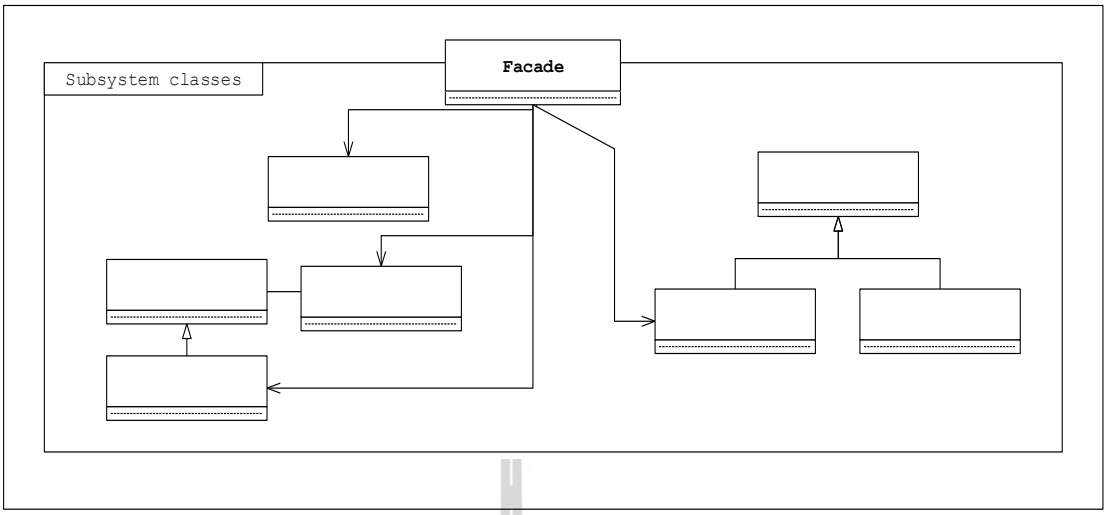
5. แพทเทิร์นส่วนหน้า

แพทเทิร์นส่วนหน้า (Façade pattern) มีวัตถุประสงค์หลัก เพื่อให้บริการอินเตอร์เฟซสำหรับการใช้ระบบย่อย กล่าวคือเพื่อให้สามารถเข้าใช้งานระบบย่อยได้ง่าย แพทเทิร์นส่วนหน้าจะทำการกำหนดอินเตอร์เฟซระดับสูงสำหรับการเข้าถึงระบบย่อย

แนวคิดและโครงสร้าง

โดยทั่วไปแล้วโครงสร้างของระบบจะมีความซับซ้อนและประกอบด้วยระบบย่อยเป็นจำนวนมาก หากคลาสอื่น ๆ มีความต้องการใช้งานระบบย่อยดังกล่าวการติดต่อสื่อสารอาจทำให้เกิดข้อผิดพลาดได้ง่าย มีการนำเสนอการใช้อ๊อบเจกต์ส่วนหน้าเพื่อจัดการกับระบบย่อยให้เป็นหนึ่งเดียวกัน โดยการทำตัวเป็นอินเตอร์เฟซเพื่อให้บริการกับคลาสอื่น ๆ ในการติดต่อกับระบบย่อย อีกทั้งวิธีการดังกล่าวยังช่วยบรรลุเป้าหมายในการทำโครงสร้างของระบบให้เป็นระบบย่อยเพื่อลดความซับซ้อนของระบบ การออกแบบโดยทั่วไปมักมีเป้าหมายเพื่อลดการสื่อสารและการพึ่งพากันระหว่างระบบย่อย

โครงสร้างแผงผังคลาสของแพทเทิร์นส่วนหน้าดังแสดงในรูปที่ 2.36



รูปที่ 2.36 โครงสร้างแพทเทิร์นส่วนหน้า

6. แพทเทิร์นน้ำหนักเบา

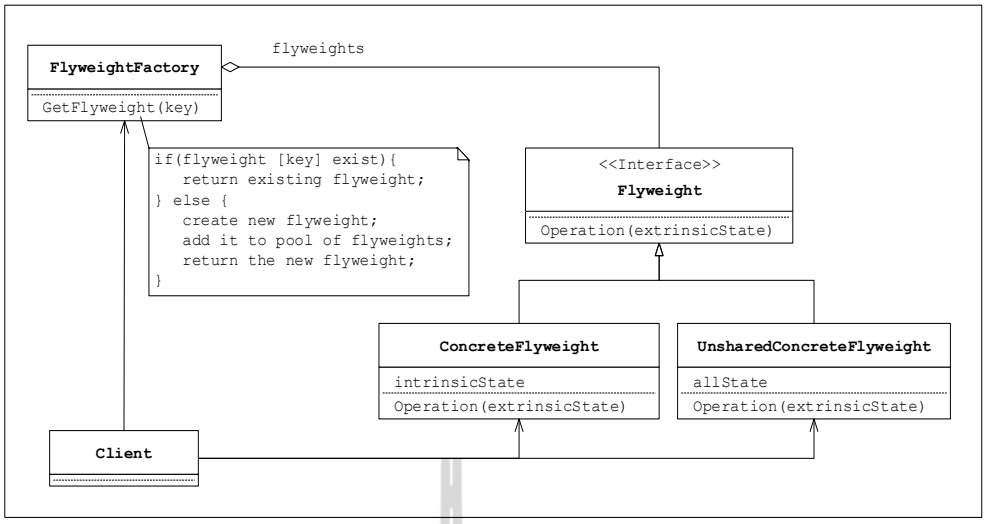
แพทเทิร์นน้ำหนักเบา (Flyweight pattern) มีวัตถุประสงค์หลัก เพื่อช่วยลดการใช้ทรัพยากรหน่วยความจำด้วยวิธีการแบ่งปัน (Sharing) ในการที่จะสนับสนุนอ๊อบเจกต์ขนาดใหญ่จำนวนมากอย่างมีประสิทธิภาพ

แนวคิดและโครงสร้าง

สำหรับบางโปรแกรมประยุกต์จะมีการใช้ประโยชน์จากการออกแบบอ๊อบเจกต์แต่หากการดำเนินการดังกล่าวไม่มีประสบการณ์ที่เพียงพออาจทำให้มีค่าใช้จ่าย (Cost) ที่สูงเกินจริง แพทเทิร์นน้ำหนักเบาอธิบายถึงวิธีการแบ่งปันอ๊อบเจกต์เพื่อให้สามารถจัดการกับอ๊อบเจกต์ที่มีความละเอียดอ่อนได้อย่างมีประสิทธิภาพโดยไม่ต้องเสียค่าใช้จ่ายที่ไม่จำเป็น

แพทเทิร์นน้ำหนักเบา คือ การใช้อ๊อบเจกต์ร่วมกันซึ่งทำให้สามารถใช้บริบท (Context) จำนวนมากพร้อมกันได้ โดยในแต่ละบริบทอ๊อบเจกต์จะเป็นอิสระต่อกัน Flyweight มีสถานะอยู่ 2 สถานะคือ *1) สถานะภายนอก* ซึ่งจะขึ้นอยู่กับบริบทของ Flyweight ที่แตกต่างกันไป จึงทำให้ไม่สามารถแบ่งปันได้ 2) *สถานะภายใน* จะถูกเก็บไว้ที่ Flyweight ซึ่งจะประกอบไปด้วยข้อมูลที่เป็นอิสระจากบริบทของ Flyweight จึงทำให้สามารถแบ่งปันได้

โครงสร้างแผงผังคลาสของแพทเทิร์นน้ำหนักเบาดังแสดงในรูปที่ 2.37



รูปที่ 2.37 โครงสร้างแพทเทิร์นน้ำหนักเบา

7. แพทเทิร์นตัวแทน

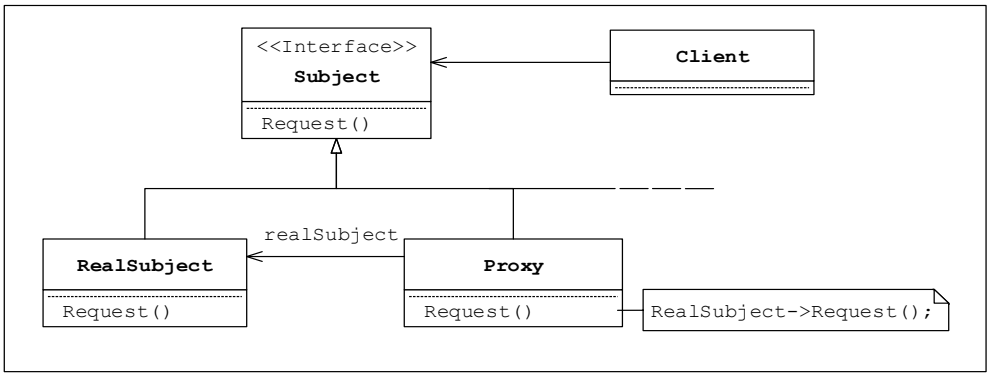
แพทเทิร์นตัวแทน (Proxy pattern) มีวัตถุประสงค์หลัก เพื่อสร้างตัวแทน (Surrogate) ให้กับอ๊อบเจกต์อื่น ในการควบคุมการเข้าถึงอ๊อบเจกต์ตัวเอง

แนวคิดและโครงสร้าง

เหตุผลที่สำคัญเหตุผลหนึ่งในการที่จะต้องทำการควบคุมการเข้าถึงอ๊อบเจกต์ก็เพื่อทำการเลื่อนสิ่งที่จะต้องเสียไปในการสร้างหรือการเริ่มต้น จนกว่าจะจำเป็นที่จะต้องใช้งานจริง ๆ เมื่อพิจารณาเอดิเตอร์เอกสารที่สามารถฝังอ๊อบเจกต์กราฟฟิกในเอดิเตอร์ได้ บางครั้งการสร้างอ๊อบเจกต์กราฟฟิกบางส่วนอาจมีค่าใช้จ่ายที่สูง เพื่อให้สามารถเปิดเอกสารได้อย่างรวดเร็วก็ควรที่จะหลีกเลี่ยงการสร้างอ๊อบเจกต์ทั้งหมด เพราะในเวลาเดียวกันจะมีเพียงแค่อ๊อบเจกต์บางส่วนที่จะต้องปรากฏในเอกสาร

เพื่อลดค่าใช้จ่ายที่สูงในการสร้างอ๊อบเจกต์สามารถแก้ไขปัญหาโดยการใช้อ๊อบเจกต์อื่น (Proxy) ให้ทำหน้าที่เป็นตัวแทนสำหรับอ๊อบเจกต์ที่แท้จริง ซึ่งจะทำหน้าที่เช่นเดียวกับอ๊อบเจกต์นั้นและดูแลการสร้างเมื่อมีความต้องการ

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวแทนดังแสดงในรูปที่ 2.38



รูปที่ 2.38 โครงสร้างแพทเทิร์นตัวแทน

2.2.3 แพทเทิร์นเชิงพฤติกรรม

แพตเทิร์นเชิงพฤติกรรม (Behavioral patterns) เป็นแพทเทิร์นที่เกี่ยวข้องกับการมอบหมายหน้าที่ความรับผิดชอบระหว่างวัตถุ และยังเป็นแพทเทิร์นที่ระบุวิธีในการสื่อสารระหว่างอ๊อบเจกต์หรือคลาส สำหรับแพทเทิร์นเชิงพฤติกรรมคลาส (Behavioral class patterns) จะใช้ลักษณะของการสืบทอดเพื่อแจกจ่ายพฤติกรรมระหว่างคลาส และแพทเทิร์นเชิงพฤติกรรมอ๊อบเจกต์ (Behavioral object patterns) จะใช้ลักษณะขององค์ประกอบของอ๊อบเจกมากกว่าการสืบทอด มีการนำเสนอแพทเทิร์นเชิงพฤติกรรมทั้ง 11 แพทเทิร์นดังต่อไปนี้

1. แพทเทิร์นห่วงโซ่ความรับผิดชอบ

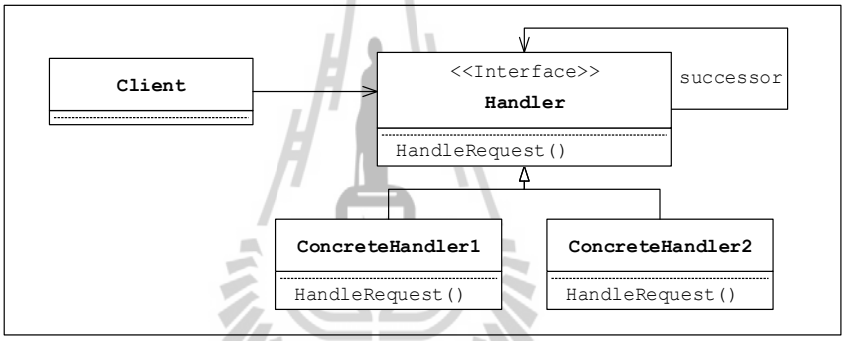
แพทเทิร์นห่วงโซ่ของความรับผิดชอบ (Chain of responsibility pattern) มีวัตถุประสงค์หลัก เพื่อสร้างห่วงโซ่สำหรับจัดการกับกระบวนการร้องขอที่เกิดขึ้นต้นแต่ต้นไปจนสิ้นสุดกระบวนการ สำหรับหลีกเลี่ยงการขึ้นต่อกันของตัวส่งการร้องขอไปยังตัวรับ

แนวคิดและโครงสร้าง

เมื่อพิจารณาส่วนติดต่อกับผู้ใช้ที่เป็นกราฟฟิกสำหรับเครื่องมือที่ให้ความช่วยเหลือ (Help) เมื่อผู้ใช้คลิกปุ่มช่วยเหลือก็จะได้รับข้อมูลที่ต้องการนั้นผ่านอินเตอร์เฟซ ซึ่งความช่วยเหลือที่จะได้รับก็จะขึ้นอยู่กับบริบทและอินเตอร์เฟซที่เลือกนั้น ดังนั้นจึงเป็นเรื่องปกติในการจัดการกับตัวช่วยเหลือจากที่เฉพาะเจาะจงที่สุดไปยังตัวช่วยเหลือแบบทั่วไปตามหลักการทั่วไปของการจัดการกับตัวช่วยเหลือ ซึ่งการจัดการกับตัวช่วยเหลือจะถูกจัดการโดยอ๊อบเจกต์หนึ่งจากหลายอ๊อบเจกต์ของส่วนติดต่อผู้ใช้ การที่จะเป็นหนึ่งอ๊อบเจกต์นั้นจะขึ้นอยู่กับบริบทและวิธีการช่วยเหลือที่มีความเฉพาะที่พร้อมใช้งาน

แต่ปัญหาสำหรับวิธีนี้ก็คือ อ๊อบเจกต์สุดท้ายจะไม่รู้จักอ๊อบเจกต์ (เช่น ปุ่ม) ที่เริ่มมีการร้องขอความช่วยเหลืออย่างชัดเจน สิ่งที่ต้องทำการแยกปุ่มของการเริ่มร้องขอออกจากอ๊อบเจกต์ที่อาจให้ข้อมูลสำหรับการช่วยเหลือ การนำแนวคิดของแพทเทิร์นห่วงโซ่ความรับผิดชอบมาใช้กำหนดวิธีการที่เกิดขึ้นดังกล่าว โดยการแยกตัวรับและตัวส่งให้หลายอ๊อบเจกต์มีโอกาสที่จะจัดการกับการร้องขอ การร้องขอจะถูกส่งไปตามห่วงโซ่ของอ๊อบเจกต์จนกว่าจะมีอ๊อบเจกต์ตัวช่วยเหลือจัดการกับการร้องขอนั้น

โครงสร้างแผงผังคลาสของแพทเทิร์นห่วงโซ่ของความรับผิดชอบดังแสดงในรูปที่ 2.39



รูปที่ 2.39 โครงสร้างแพทเทิร์นห่วงโซ่ของความรับผิดชอบ

2. แพทเทิร์นคำสั่ง

แพทเทิร์นคำสั่ง (Command pattern) มีวัตถุประสงค์หลัก เพื่อช่วยให้การร้องขอที่แตกต่างกันจากพารามิเตอร์ของไคลเอนต์ (เช่น ล็อกของการร้องขอ (Log requests) หรือ ลำดับ) และการสนับสนุนตัวดำเนินการที่สามารถย้อนกลับได้ โดยวิธีการการห่อหุ้มการร้องขอให้เป็นอ๊อบเจกต์

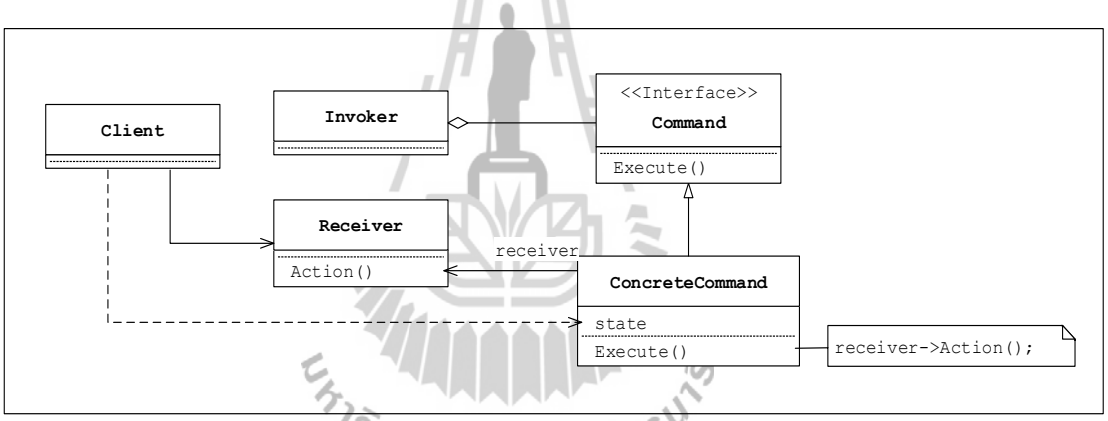
แนวคิดและโครงสร้าง

บางครั้งก็เป็นสิ่งที่จำเป็นที่จะจัดการกับปัญหาการร้องขอให้เป็นอ๊อบเจกต์โดยไม่ต้องรู้เกี่ยวกับการดำเนินการร้องขอหรือตัวรับของการร้องขอนั้น พิจารณาส่วนติดต่อกับผู้ใช้ของชุดเครื่องมือรวมถึงอ๊อบเจกต์ เช่น ปุ่ม ในการที่จะทำให้การตอบสนองต่อการป้อนข้อมูลของผู้ใช้เมื่อมี

การร้องขอเกิดขึ้น จากการที่โปรแกรมประยุกต์เท่านั้นที่จะรู้ว่าควรจะทำอะไรกับอ๊อบเจกต์ ทำให้ชุดเครื่องมือนั้นไม่สามารถดำเนินการกับการร้องขอที่เกิดจากปุ่มนั้นได้ เป็นผลให้นักออกแบบชุดเครื่องมือไม่มีทางที่จะรู้เกี่ยวกับการดำเนินการหรือตัวรับของการร้องขอที่จะทำให้ได้ผลลัพธ์ออกมา

แพทเทิร์นคำสั่งจะเป็นตัวช่วยให้อ๊อบเจกต์ชุดเครื่องมือสร้างการร้องขอ โดยการเปลี่ยนแปลงการร้องขอไปเป็นอ๊อบเจกต์ เมื่อการสร้างการร้องขอนั้นไม่ได้ระบุไว้ในอ๊อบเจกต์ของโปรแกรมประยุกต์ อ๊อบเจกต์ดังกล่าวสามารถทำการจัดเก็บและส่งผ่านค่าได้เหมือนกับอ๊อบเจกต์อื่น ๆ

โครงสร้างแผงผังคลาสของแพทเทิร์นคำสั่งดังแสดงในรูปที่ 2.40



รูปที่ 2.40 โครงสร้างแพทเทิร์นคำสั่ง

3. แพทเทิร์นตัวแปล

แพทเทิร์นตัวแปล (Interpreter pattern) มีวัตถุประสงค์หลัก เพื่อระบุวิธีการที่จะแปลหรือประเมินประโยคในภาษา แพทเทิร์นช่วยให้ภาษาสามารถกำหนดตัวที่จะทำหน้าที่แทนไวยากรณ์และตัวแปลนั้นเพื่อใช้เป็นตัวแทนในการแปลประโยคในภาษา

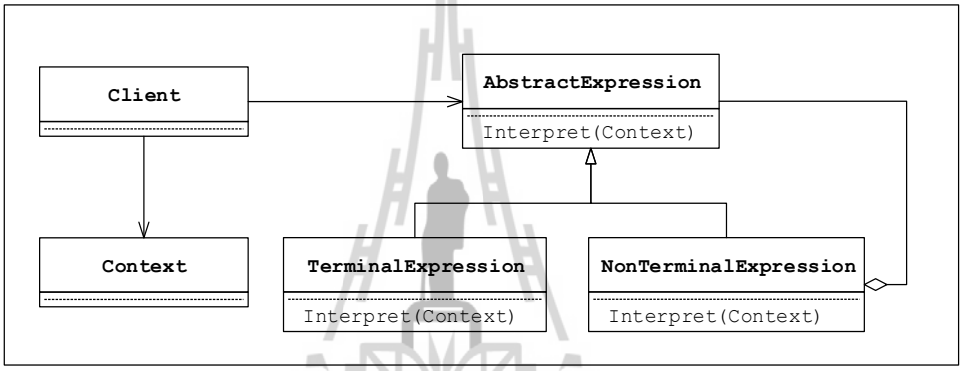
แนวคิดและโครงสร้าง

การที่จะสร้างตัวแปลสำหรับการแก้ไขปัญหาโดยการตีความประโยค ก็ต่อเมื่อชนิดของปัญหานั้นเกิดขึ้นบ่อยมากพอ และคุ้มค่าพอที่จะแสดงอินสแตนซ์ของปัญหาเป็นประโยคใน

ภาษาอย่างง่าย ตัวอย่างปัญหาที่พบโดยทั่วไป เช่นการค้นหาสตริงให้ตรงกับรูปแบบที่กำหนดไว้ นิพจน์ปกติสำหรับการกำหนดแพทเทิร์นของสตริงนั้นเป็นนิพจน์ของภาษามาตรฐาน แทนที่จะสร้างอัลกอริทึมกำหนดเองในการที่จะให้ตรงกับแต่ละแพทเทิร์นที่เกี่ยวข้องกับสตริง

แพทเทิร์นตัวแปลจะเป็นตัวอธิบายถึงวิธีในการกำหนดไวยากรณ์สำหรับภาษาอย่างง่าย โดยการกำหนดตัวแทนของประโยคในภาษา และการแปลประโยคในภาษานั้น

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวแปรภาษาดังแสดงในรูปที่ 2.41



รูปที่ 2.41 โครงสร้างแพทเทิร์นตัวแปรภาษา

4. แพทเทิร์นตัววนซ้ำ

แพทเทิร์นตัววนซ้ำ (Iterator pattern) มีวัตถุประสงค์หลัก เพื่อให้สามารถเข้าถึงองค์ประกอบของอ๊อบเจกต์ที่รวมตัวกัน (Aggregate object) ตามลำดับ โดยที่ไม่จำเป็นต้องเปิดเผยโครงสร้างพื้นฐานของอ๊อบเจกต์ที่มีการเข้าถึง

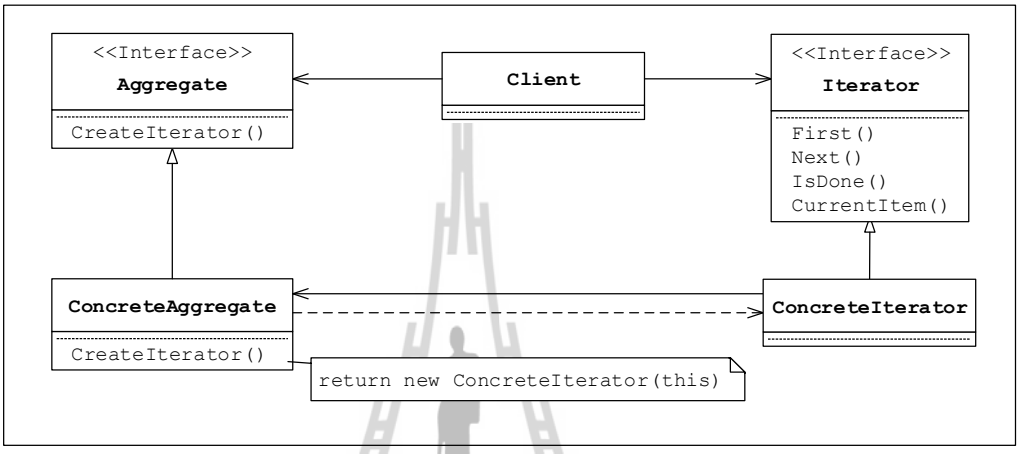
แนวคิดและโครงสร้าง

เมื่อพิจารณาอ๊อบเจกต์ที่รวมตัวกัน เช่น รายการ (List) ที่ควรจะยอมให้เข้าถึงองค์ประกอบโดยที่ไม่เปิดเผยโครงสร้างภายใน บางครั้งการจะทำสิ่งที่ต้องการให้สำเร็จ อาจจะต้องมีการเข้าถึงรายการซึ่งการเข้าถึงก็จะแตกต่างกันไปขึ้นอยู่กับว่าต้องการทำอะไร และบางครั้งในการเข้าถึงรายการนั้นอาจมีมากกว่าหนึ่งการเข้าถึงที่เข้าไปรออยู่ในรายการเดียวกัน

แพทเทิร์นตัววนซ้ำมีแนวคิดหลักคือการสร้างความรับผิดชอบสำหรับการเข้าถึงและการแวะผ่านอ๊อบเจกต์ของรายการ ตลอดจนการวางลงในอ๊อบเจกต์การเข้าถึง โดยคลาส Iterator จะ

เป็นตัวกำหนดอินเตอร์เฟซสำหรับการเข้าถึงองค์ประกอบของรายการ และอ๊อบเจกต์การเข้าถึง (ConcreteIterator) จะเป็นผู้รับผิดชอบสำหรับการติดตามองค์ประกอบปัจจุบัน

โครงสร้างแผงผังคลาสของแพทเทิร์นตัววนซ้ำดังแสดงในรูปที่ 2.42



รูปที่ 2.42 โครงสร้างแพทเทิร์นตัววนซ้ำ

5. แพทเทิร์นตัวกลาง

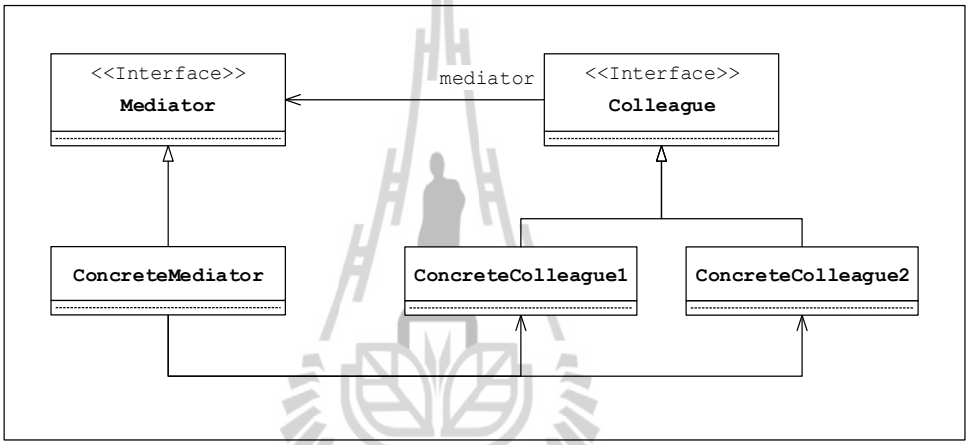
แพทเทิร์นตัวกลาง (Mediator pattern) มีวัตถุประสงค์หลัก เพื่อสร้างอ๊อบเจกต์ให้เป็นตัวกลางในการติดต่อสื่อสารกันระหว่างอ๊อบเจกต์จำนวนมาก การใช้แพทเทิร์นตัวกลางนี้จะช่วยให้การขึ้นต่อกันของอ๊อบเจกต์ลดลง ทำให้การสื่อสารกันระหว่างอ๊อบเจกต์สามารถทำได้อย่างอิสระ

แนวคิดและโครงสร้าง

ในการออกแบบเชิงวัตถุนั้นเน้นการกระจายพฤติกรรมของอ๊อบเจกต์ ส่งผลให้เกิดโครงสร้างอ๊อบเจกต์จำนวนมากมีการขึ้นต่อกัน และเป็นไปได้ว่าทุก ๆ อ๊อบเจกต์นั้นมีความรู้เกี่ยวกับอ๊อบเจกต์อื่น ๆ ซึ่งถือว่าเป็นกรณีที่เลวร้ายที่สุด แม้ว่าการกระทำเช่นนี้จะช่วยเพิ่มความสามารถของการนำอ๊อบเจกต์เดิมกลับมาใช้ซ้ำ แต่ก็เป็นการยากเมื่อจะทำการเปลี่ยนแปลงพฤติกรรมของอ๊อบเจกต์ที่ต้องการ หรืออาจจำเป็นต้องกำหนดซับคลาสเพิ่มเป็นจำนวนมากในการปรับแต่งพฤติกรรมของอ๊อบเจกต์

แพทเทิร์นตัวกลางสามารถช่วยแก้ไขปัญหาดังกล่าวได้ด้วยการปกป้องวัตถุที่เป็นพฤติกรรมรวม (Collective behavior) ในอ๊อบเจกต์ตัวกลางที่แยกออกมา อ๊อบเจกต์ตัวกลางจะเป็นผู้รับผิดชอบในการควบคุมและประสานงานในการโต้ตอบกันของกลุ่มอ๊อบเจกต์ ซึ่งเสมือนว่าตัวเองเป็นตัวกลางในการอ้างอิงกันระหว่างอ๊อบเจกต์ในกลุ่ม ทำให้สามารถลดจำนวนของการเชื่อมโยงกันระหว่างอ๊อบเจกต์ได้ เพราะจะมีแต่อ๊อบเจกต์เท่านั้นที่รู้จักตัวกลาง

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวกลางดังแสดงในรูปที่ 2.43



รูปที่ 2.43 โครงสร้างแพทเทิร์นตัวกลาง

6. แพทเทิร์นที่ระลึก

แพทเทิร์นที่ระลึก (Memento pattern) มีวัตถุประสงค์หลัก เพื่อให้อ๊อบเจกต์สามารถย้อนกลับไปยังสถานะก่อนหน้าได้ โดยไม่ให้ละเมิดกฎการห่อหุ้มด้วยการเก็บสถานะภายในของอ๊อบเจกต์ให้อยู่ในคลาสอื่น

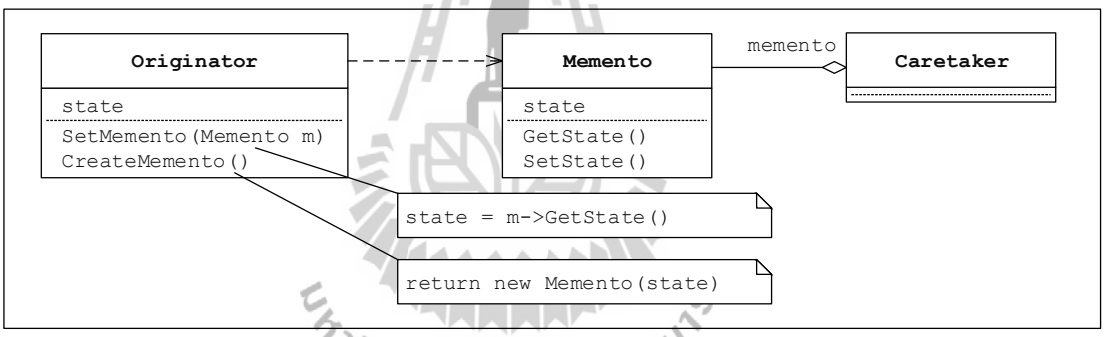
แนวคิดและโครงสร้าง

เมื่อพิจารณากลไกการย้อนกลับของการดำเนินการเบื้องต้น หรือการกู้คืนจากข้อผิดพลาด ทำให้จำเป็นที่จะต้องมีการบันทึกสถานะภายในของอ๊อบเจกต์เพื่อให้อ๊อบเจกต์สามารถกลับไปยังสถานะก่อนหน้าได้ โดยปกติแล้วอ๊อบเจกต์จะห่อหุ้มสถานะบางส่วนหรือทั้งหมดของอ๊อบเจกต์ ทำให้ไม่สามารถจัดเก็บสถานะที่ภายนอกได้ และไม่สามารถเข้าถึงอ๊อบเจกต์อื่นได้ ดั้งนั้น

การที่จะเปิดเผยให้เห็นถึงสถานะภายในของอ๊อบเจกต์จึงถือเป็นการละเมิดกฎการห่อหุ้ม พิจารณาเอดิเตอร์กราฟฟิกที่สนับสนุนการเชื่อมต่อกันของอ๊อบเจกต์ การใช้เส้นในการเชื่อมต่อกันของอ๊อบเจกต์สี่เหลี่ยมสองอ๊อบเจกต์ เมื่อมีการย้ายที่ของสี่เหลี่ยมจะต้องมีการเก็บรักษาระยะทางเดิมไว้เพื่อที่ว่าเมื่อมีการยกเลิกการย้ายที่จะสามารถกลับไปยังตำแหน่งเดิมได้

แพทเทิร์นที่ระลึกช่วยสนับสนุนการทำงานของอ๊อบเจกต์ให้สามารถกลับไปยังสถานะก่อนหน้าได้ การอ๊อบเจกต์ที่ระลึกนั้นจะทำการเก็บรวบรวมสถานะภายในของอ๊อบเจกต์อื่น ๆ ในกลไกการย้อนกลับนั้นจะร้องของอ๊อบเจกต์ที่ระลึกจากอ๊อบเจกต์เริ่มต้นเพื่อตรวจสอบสถานะของอ๊อบเจกต์เริ่มต้น และอ๊อบเจกต์เริ่มต้นเท่านั้นที่จะสามารถจัดเก็บและดึงข้อมูลจากอ๊อบเจกต์ที่ระลึกได้ กล่าวคืออ๊อบเจกต์ที่ระลึกจะไม่สามารถเข้าถึงได้ด้วยอ๊อบเจกต์อื่น ๆ

โครงสร้างแผงผังคลาสของแพทเทิร์นที่ระลึกดังแสดงในรูปที่ 2.44



รูปที่ 2.44 โครงสร้างแพทเทิร์นที่ระลึก

7. แพทเทิร์นตัวสังเกตการณ์

แพทเทิร์นตัวสังเกตการณ์ (Observer pattern) มีวัตถุประสงค์หลัก เพื่อกำหนดการพึ่งพากันของอ๊อบเจกต์ให้มีความสัมพันธ์แบบหนึ่งต่อกลุ่ม เพื่อให้เมื่อมีหนึ่งอ๊อบเจกต์ได้รับการเปลี่ยนแปลงสถานะแล้ว จะมีการแจ้งเตือนและการปรับปรุงอ๊อบเจกต์ทั้งหมดที่อยู่ในการพึ่งพากันเกิดขึ้นโดยอัตโนมัติ

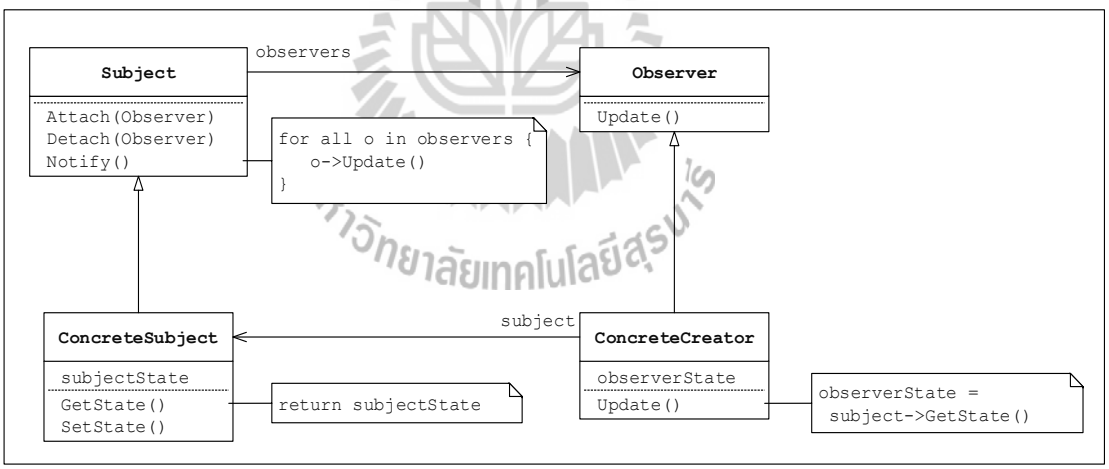
แนวคิดและโครงสร้าง

เมื่อพิจารณาชุดเครื่องมือของส่วนติดต่อกับผู้ใช้ที่เป็นแบบกราฟฟิก สำหรับการ

นำเสนอลักษณะข้อมูลจากโปรแกรมประยุกต์ คลาสที่ใช้ในการกำหนดข้อมูลของโปรแกรมประยุกต์และสามารถนำข้อมูลนั้นมาใช้นำเสนอได้อย่างอิสระ จากอ๊อบเจกต์ข้อมูลของโปรแกรมประยุกต์เดียวสามารถนำเสนอได้ทั้งในรูปแบบอ๊อบเจกต์สเปรดชีต อ๊อบเจกต์แผนภูมิแท่ง อ๊อบเจกต์แผนภูมิวงกลม ฯลฯ ในการนำเสนอข้อมูลผ่านอ๊อบเจกต์ที่แตกต่างกันนั้น ในแต่ละอ๊อบเจกต์จะไม่ทราบอะไรที่เกี่ยวข้องกัน จึงสามารถนำอ๊อบเจกต์นั้นมาใช้ได้เพียงหนึ่งอ๊อบเจกต์ แต่ในทางตรงกันข้ามแล้วเมื่อมีการเปลี่ยนแปลงข้อมูลในโปรแกรมประยุกต์ก็จะทำให้อ๊อบเจกต์ในการนำเสนอข้อมูลมีการเปลี่ยนแปลงในทันที

แพทเทิร์นตัวสังเกตการณ์สามารถใช้อธิบายถึงวิธีการสร้างความสัมพันธ์ดังกล่าว อ๊อบเจกต์ที่สำคัญในแพทเทิร์นนี้คือ อ๊อบเจกต์ของคลาส Subject และ อ๊อบเจกต์ของคลาส Observer ทั้งนี้จำนวนของอ๊อบเจกต์ Subject จะมีจำนวนเท่าใดก็ขึ้นอยู่กับอ๊อบเจกต์ Observer เมื่อใดก็ตามที่หัวข้อมีการเปลี่ยนแปลงสถานะแล้วตัวสังเกตการณ์ทั้งหมดก็จะทำการแจ้งเตือนทันที

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวสังเกตการณ์ดังแสดงในรูปที่ 2.45



รูปที่ 2.45 โครงสร้างแพทเทิร์นตัวสังเกตการณ์

8. แพทเทิร์นสถานะ

แพทเทิร์นสถานะ (State pattern) มีวัตถุประสงค์หลัก เพื่อเมื่อสถานะภายในมีการเปลี่ยนแปลงจะยอมให้อ๊อบเจกต์เปลี่ยนแปลงพฤติกรรม และอ๊อบเจกต์ของสถานะจะปรากฏขึ้น

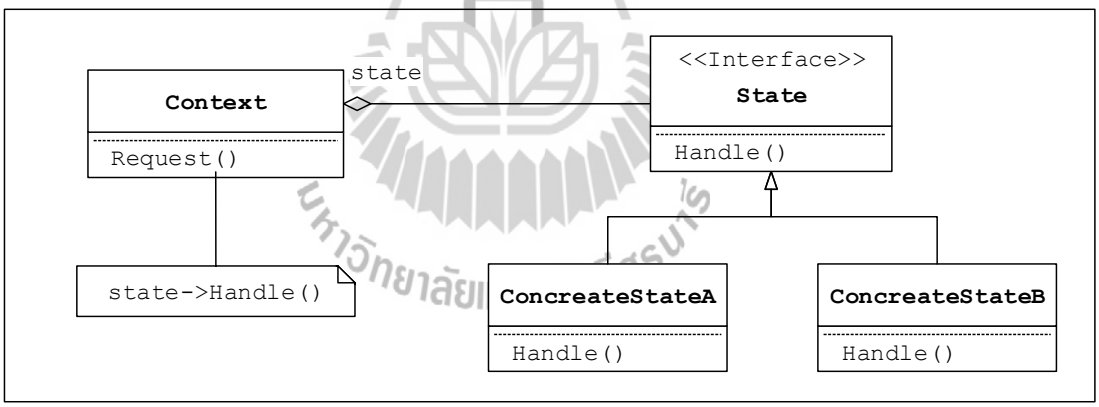
เมื่อมีการเปลี่ยนแปลงคลาส

แนวคิดและโครงสร้าง

เมื่อพิจารณาคลาสสำหรับการเชื่อมต่อเครือข่าย และอ๊อบเจกต์ของคลาสประกอบด้วยสถานะของการเชื่อมต่อ และการหยุดการเชื่อมต่อ เมื่ออ๊อบเจกต์ของคลาสการเชื่อมต่อได้รับการร้องขอจากอ๊อบเจกต์อื่น การตอบสนองไปยังอ๊อบเจกต์นั้นจะแตกต่างกันไปขึ้นอยู่กับสถานะของอ๊อบเจกต์ของคลาสการเชื่อมต่อ เช่น การร้องขอทำให้เกิดการหยุดการเชื่อมต่อ ก็ต่อเมื่อสถานะก่อนหน้าอยู่ในสถานะของการเชื่อมต่อ

แพทเทิร์นสถานะสามารถใช้อธิบายวิธีการเชื่อมต่อเครือข่ายของคลาสการเชื่อมต่อในการแสดงพฤติกรรมที่แตกต่างกันในแต่ละสถานะ แนวคิดของแพทเทิร์นนี้คือการแนะนำคลาสนามธรรมเพื่อเป็นตัวแทนของสถานะในการเชื่อมต่อเครือข่ายในการดำเนินการที่แตกต่างกัน

โครงสร้างแผงผังคลาสของแพทเทิร์นสถานะดังแสดงในรูปที่ 2.46



รูปที่ 2.46 โครงสร้างแพทเทิร์นสถานะ

9. แพทเทิร์นกลยุทธ์

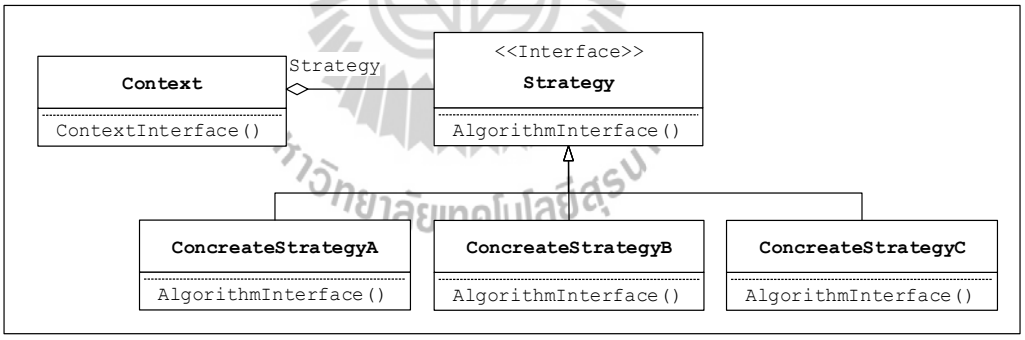
แพทเทิร์นกลยุทธ์ (Strategy pattern) มีวัตถุประสงค์หลัก เพื่อช่วยให้อัลกอริทึมที่แตกต่างกันเป็นอิสระจากไคลเอนต์ที่ต้องการใช้ โดยการห่อหุ้มอัลกอริทึมที่มีการกำหนดขึ้น และสามารถเปลี่ยนแปลงการเลือกใช้อัลกอริทึมเหล่านั้น

แนวคิดและโครงสร้าง

เมื่อพิจารณาอัลกอริทึมสำหรับหยุดข้อความในบรรทัดที่มีอยู่เป็นจำนวนมากนั้น การพัฒนาอัลกอริทึมแบบไม่พึ่งพาอะไรเลย อาจทำให้เกิดปัญหาหลายอย่างคือ ถ้าไคลเอนต์ที่ต้องการใช้อัลกอริทึมแยกบรรทัด (Linebreaking algorithms) มีการเรียกใช้ (Include) โค้ดของอัลกอริทึมไลน์เบรคอาจทำให้ระบบมีความซับซ้อนมากขึ้น และไคลเอนต์มีขนาดใหญ่ขึ้นทำให้ยากที่จะทำการบำรุงรักษา การที่อัลกอริทึมมีอยู่เป็นจำนวนมากนั้น ก็จะเหมาะสมกับการเรียกใช้งานที่แตกต่างกัน จึงไม่จำเป็นที่จะต้องสนับสนุนทุกอัลกอริทึมถ้าหากไม่ได้ใช้ เมื่ออัลกอริทึมเป็นส่วนหนึ่งของไคลเอนต์แล้ว จะทำให้การเพิ่มอัลกอริทึมใหม่และแตกต่างจากอัลกอริทึมที่มีอยู่เป็นเรื่องที่ยากมาก

แพทเทิร์นกลยุทธ์จึงถูกนำมาใช้ในการหลีกเลี่ยงปัญหาดังกล่าว โดยการกำหนดคลาสสำหรับการห่อหุ้มอัลกอริทึมที่แตกต่างกัน ซึ่งก็คือการกำหนดคลาสสำหรับห่อหุ้มอัลกอริทึมไลน์เบรคที่แตกต่างกัน ซึ่งจะทำให้ง่ายต่อการเปลี่ยนแปลงและการขยายอัลกอริทึม

โครงสร้างแผงผังคลาสของแพทเทิร์นกลยุทธ์ดังแสดงในรูปที่ 2.47



รูปที่ 2.47 โครงสร้างแพทเทิร์นกลยุทธ์

10. แพทเทิร์นเมธอดแม่แบบ

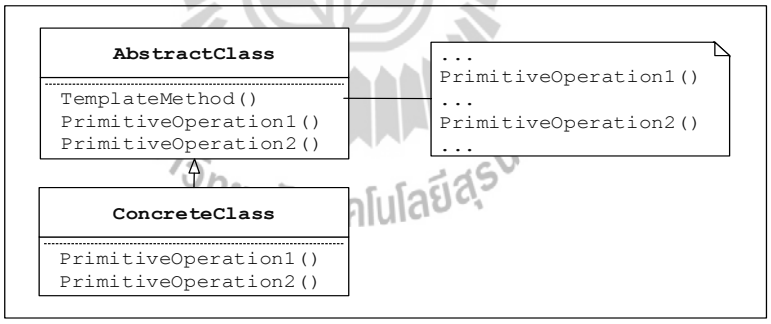
แพทเทิร์นเมธอดแม่แบบ (Template method pattern) มีวัตถุประสงค์หลัก เพื่อกำหนดเค้าโครงของอัลกอริทึม และไม่ต้องกำหนดรายละเอียดของขั้นตอนลงไปในซับคลาส ด้วยคุณสมบัติของเมธอดแม่แบบจะช่วยให้ซับคลาสสามารถกำหนดรายละเอียดที่แน่นอนในอัลกอริทึมโดยที่ไม่ต้องทำการเปลี่ยนแปลงโครงสร้างของอัลกอริทึม

แนวคิดและโครงสร้าง

เมื่อพิจารณาโปรแกรมประยุกต์ที่ประกอบด้วยคลาสสำหรับทำหน้าที่เป็นผู้รับผิดชอบในการเปิดเอกสารที่มีอยู่ที่ถูกเก็บไว้ภายนอก (เช่น ไฟล์) และคลาสสำหรับทำหน้าที่แสดงข้อมูลในเอกสารเมื่อมีการอ่านจากไฟล์ คลาสการเปิดเอกสารจะกำหนดซับคลาสสำหรับการเปิดและอ่านเอกสารตามลำดับด้วยเมธอดสำหรับการเปิดและอ่านเอกสาร โดยจะทำการตรวจสอบว่าถ้าเอกสารสามารถเปิดได้ แล้วจะสร้างอ๊อบเจกต์การแสดงข้อมูล เพื่อทำการแสดงข้อมูลในเอกสารจากไฟล์ เมธอดที่ทำหน้าที่ในการเปิดและอ่านเอกสาร เรียกว่าเมธอดแม่แบบ

เมธอดแม่แบบจะกำหนดอัลกอริทึมในลักษณะของตัวดำเนินการนามธรรม เพื่อให้ซับคลาสทำการโอเวอร์ไรด์พฤติกรรมนั้นแล้วทำให้เป็นรูปธรรม กล่าวคือซับคลาสการเปิดเอกสารจะกำหนดอัลกอริทึมในการตรวจสอบว่าถ้าเอกสารนั้นสามารถเปิดได้และสร้างอ๊อบเจกต์ของการแสดงข้อมูล เพื่อกำหนดขึ้นตอนในการอ่านเอกสาร อีกทั้งเมื่อมีการเปิดเอกสารแล้วยังสามารถที่จะกำหนดให้ซับคลาสของคลาสการเปิดเอกสารทราบได้อีกด้วย

โครงสร้างแผงผังคลาสของแพทเทิร์นเมธอดแม่แบบดังแสดงในรูปที่ 2.48



รูปที่ 2.48 โครงสร้างแพทเทิร์นเมธอดแม่แบบ

11. แพทเทิร์นตัวเยี่ยมเยือน

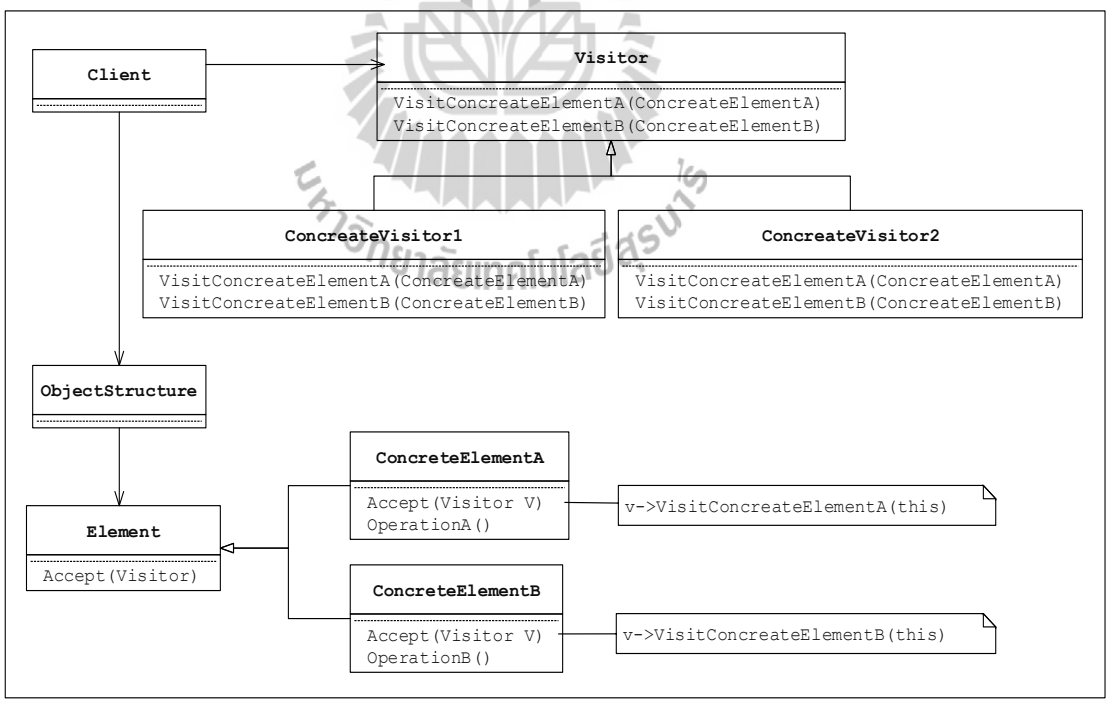
แพทเทิร์นตัวเยี่ยมเยือน (Visitor pattern) มีวัตถุประสงค์หลัก เพื่อให้สามารถกำหนดการดำเนินการใหม่โดยไม่ต้องมีการเปลี่ยนแปลงคลาสขององค์ประกอบที่จะดำเนินการ กล่าวคือตัวเยี่ยมเยือนจะทำหน้าที่แทนการดำเนินการในการที่จะทำให้องค์ประกอบของโครงสร้างอ๊อบเจกต์ดำเนินการได้สำเร็จ

แนวคิดและโครงสร้าง

เมื่อพิจารณาคอมไพเลอร์ที่แสดงในลักษณะต้นไม้วากยสัมพันธ์เชิงนามธรรม ที่จำเป็นต้องมีการสร้างโค้ด การตรวจสอบชนิดของตัวแปร การเพิ่มประสิทธิภาพให้กับโค้ด การวิเคราะห์กระบวนการทำงาน ในการดำเนินการดังกล่าวส่วนใหญ่จำเป็นต้องมีการเก็บรักษาโหนดคำสั่งที่ได้รับมอบหมาย ทำให้เกิดคลาสลำดับชั้นของโหนดถ้าหากมีการกระจายการดำเนินการทั้งหมดในคลาสลำดับชั้นของโหนดแล้ว จะทำให้ระบบยากที่จะเข้าใจ เปลี่ยนแปลงและบำรุงรักษายาก อีกทั้งการเพิ่มการทำงานใหม่ก็มักจะต้องคอมไพล์ทุกคลาสใหม่

แพทเทิร์นตัวเยี่ยมเยือนจะช่วยให้สามารถกำหนดลำดับชั้นออกเป็น 2 คลาส คลาสหนึ่งสำหรับ Visitor เพื่อให้กำหนดการดำเนินการบนองค์ประกอบ และอีกคลาสหนึ่งสำหรับ Element ที่จะนำไปดำเนินการ ทำให้การเพิ่มการดำเนินการใหม่นั้นทำได้ง่ายโดยการเพิ่มซับคลาสใหม่ไปยังคลาส Visitor

โครงสร้างแผงผังคลาสของแพทเทิร์นตัวเยี่ยมเยือนดังแสดงในรูปที่ 2.49



รูปที่ 2.49 โครงสร้างแพทเทิร์นตัวเยี่ยมเยือน

2.3 ตัวทดสอบประสิทธิภาพ Deltablue

Deltablue เป็นวิธีการหนึ่งในการแก้ไขปัญหาเรื่องข้อจำกัด ถูกพัฒนาขึ้นโดย John Maloney และ Mario Wolczko มีการพัฒนา Deltablue ครั้งแรกในภาษาเขียนโปรแกรมภาษาสมอลล์ทอล์ก (Smalltalk) โดยมีจุดสนใจหลักของตัวทดสอบ Deltablue อยู่ในการมีหลายรูปแบบ และการเขียนโปรแกรมเชิงวัตถุ (The Dart Team, 2014) DeltaBlue Benchmark เป็นมาตรฐานในการทดสอบประสิทธิภาพทั่วไปสำหรับการเปรียบเทียบภาษาเขียนโปรแกรมเชิงวัตถุ (Hemel, 2013)

2.4 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับการเปรียบเทียบและการประเมินผลภาษาเขียนโปรแกรมมีเป็นจำนวนมาก การศึกษาการเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยดีไซน์แพทเทิร์น ผู้วิจัยได้ทำการศึกษาค้นคว้างานวิจัยที่เกี่ยวข้องกับการประเมินผลภาษาที่ใช้ในการเขียนโปรแกรมที่น่าสนใจ โดยสรุปไว้ในตารางที่ 2.3 และมีรายละเอียดดังต่อไปนี้

1. งานวิจัยเรื่อง “Evaluating a new programming language” ของ Steven Clarke (2001) ได้นำเสนอการประเมินภาษาที่ใช้ในการเขียนโปรแกรมด้วยการเปรียบเทียบข้อมูลที่ได้จากแบบสอบถามเพื่อรวบรวมข้อมูลของภาษาและการประเมินภาษาจากห้องปฏิบัติการ ภาษาที่ได้ทำรับการประเมินในงานวิจัยนี้คือภาษาซีชาร์ป (C#) ที่ได้รับการพัฒนาโดย Microsoft Corp., โดยการใช้แบบสอบถามที่ชื่อว่า “Cognitive Dimensions” เป็นแบบสอบถามที่ได้รับการยอมรับในการวิเคราะห์สัญลักษณ์หรือภาษาเขียนโปรแกรมที่เกิดขึ้นใหม่ ในงานวิจัยจะมีปัญหาเรื่องของประสบการณ์ในการเขียนโปรแกรมและความถูกต้องที่ได้จากแบบสอบถาม จึงได้แก้ปัญหาโดยการทำการประเมินจากห้องปฏิบัติการก่อน แล้วจึงสร้างแบบสอบถาม ในการเปรียบเทียบข้อมูลโดยการอภิปรายที่ได้จากทั้ง 2 แหล่งข้อมูล งานวิจัยนี้มุ่งเน้น 3 ประเด็นดังต่อไปนี้

1) ข้อสังเกตจากห้องปฏิบัติการกับแบบสอบถาม ซ้อนทับกัน

2) ข้อสังเกตที่ไม่ได้รับรายงานจากแบบสอบถาม

3) ข้อสังเกตที่ไม่มีในห้องปฏิบัติการแต่ได้รับรายงานจากแบบสอบถาม

2. งานวิจัยเรื่อง “A Comparison of Object – Oriented Languages in Software Engineering” ของ Mohammad Reza Nami (2008) เป็นงานวิจัยที่นำเสนอการเปรียบเทียบภาษาที่ใช้ในการเขียนโปรแกรมเชิงวัตถุในงานวิศวกรรมซอฟต์แวร์ รวมถึงการนำเสนอการเปรียบเทียบคุณสมบัติของภาษาจาวา ภาษาซีพลัสพลัส (C++) ภาษาไอเฟล (Eiffel) และภาษาสมอลล์ทอล์ก โดยทำการเปรียบเทียบภาษาเขียนโปรแกรมเชิงวัตถุจากคุณสมบัติดังต่อไปนี้

1) แบบชนิดเชิงสถิตย์ (Static Typing)

2) เทคโนโลยีการคอมไพล์ (Compilation Technology)

3) ความสัมพันธ์ระหว่างภาษาเขียนโปรแกรมเชิงวัตถุและปัจจัยที่มีผลต่อคุณภาพ

4) การจัดทำเอกสารอัตโนมัติ (Automatic Documentation)

5) การสืบทอดจากหลายคลาส (Multiple-Inheritance)

6) การสร้างโครงสร้างพื้นฐาน (Building Infrastructures)

3. งานวิจัยเรื่อง “A COMPARISON OF JAVA AND C#” ของ Shymal Suhana Chandra และ Kailash Chandra (2005) เป็นการเปรียบเทียบภาษาจาวา และภาษาซีชาร์ปซึ่งเป็นภาษาที่ได้รับการพัฒนาขึ้นมาใหม่ในช่วงนั้น (ค.ศ. 2004) โดยงานวิจัยนี้ได้มุ่งเน้นไปที่แนวคิดเบื้องต้นในการเขียนโปรแกรมของแต่ละภาษา การพัฒนาโปรแกรมอย่างง่าย และผลลัพธ์ที่ได้ และหวังว่างานวิจัยนี้จะเป็นประโยชน์ในการเลือกภาษาที่จะใช้ในการเขียนโปรแกรมเป็นภาษาแรก โดยได้นำเสนอการเปรียบเทียบทั้งสองภาษาตามหัวข้อดังต่อไปนี้

1) การเขียนโปรแกรมอย่างง่าย

2) ตัวบ่งชี้ และคำสงวนของภาษา

3) อ๊อบเจกต์และประเภทของข้อมูล

4) ตัวดำเนินการ

5) การโอเวอร์โหลดโอเปอร์เรเตอร์

6) การสร้างคลาส

7) อะเรย์

8) อะเรย์แบบเชิงพลวัต

9) สตริง

10) การเขียนเมธอดแลกเปลี่ยน

11) การสืบทอด

12) อินเตอร์เฟซ

4. งานวิจัยเรื่อง “GoHotDraw : Evaluating the Go Programming Language with Design Patterns” ของ Frank Schmager, Nicholas Cameron และ James Noble (2010) เป็นงานวิจัยที่นำเสนอการประเมินผลภาษาโก (Go) ซึ่งเป็นภาษาได้รับการสนับสนุนจากกูเกิล ในงานวิจัยทำการประเมินภาษาโกด้วยดีไซน์แพทเทิร์นของ GoF ทั้ง 23 แพทเทิร์นและทำการพัฒนากรอบงาน HotDraw ด้วยภาษาโก มีการแสดงให้เห็นถึงคุณลักษณะของภาษาโกที่มีผลต่อการพัฒนาของดีไซน์

แพทเทิร์น การระบุถึงบางอย่างที่อาจเกิดขึ้นได้จากการพัฒนาด้วยภาษาโก และการแสดงให้เห็นถึงวิธีการในการศึกษาดีไซน์แพทเทิร์นว่าสามารถนำมาใช้ในการประเมินผลของภาษาที่ใช้ในการเขียนโปรแกรมได้

จากการศึกษาปริทัศน์ วรรณกรรมและงานวิจัยที่เกี่ยวข้อง ได้แสดงให้เห็นว่า งานวิจัยส่วนในการเปรียบเทียบและการประเมินผลของภาษาที่ใช้ในการเขียนโปรแกรมนั้น มุ่งเน้นให้เห็นถึงความแตกต่างกันของแต่ละภาษา การศึกษาถึงคุณสมบัติของภาษาที่ได้นำมาใช้ในงานวิจัย การนำเสนอวิธีการที่จะนำมาใช้ในการเปรียบเทียบและประเมินภาษาที่ใช้เขียนโปรแกรม ปัจจุบันยังคงมีภาษาที่ใช้ในการเขียนโปรแกรมถูกพัฒนาขึ้นอย่างต่อเนื่องและสมควรที่จะได้รับการเปรียบเทียบหรือการประเมินผลของภาษา ผู้วิจัยจึงได้นำแนวทางในการเปรียบเทียบและการประเมินภาษาที่ใช้ในการเขียนโปรแกรมจากงานวิจัยที่ได้กล่าวมาในข้างต้น เพื่อทำการประเมินภาษาดาร์ทด้วยดีไซน์แพทเทิร์น โดยทำการสรุปลักษณะและวิธีการที่ใช้ในการประเมินภาษาของงานวิจัยที่เกี่ยวข้องไว้ในตารางที่ 2.3

ก. แทนงานวิจัยเรื่อง “Evaluating a new programming language” ของ Steven Clarke

ข. แทนงานวิจัยเรื่อง “A Comparison of Object - Oriented Languages in Software Engineering” ของ Mohammad Reza Nami

ค. แทนงานวิจัยเรื่อง “A COMPARISON OF JAVA AND C#” ของ Shymal Suhana Chandra และ Kailash Chandra

ง. แทนงานวิจัยเรื่อง “GoHotDraw: Evaluating the Go Programming Language with Design Patterns” ของ Frank Schmager, Nicholas Cameron และ James Noble

จ. แทนงานวิจัยเรื่อง “การเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยดีไซน์แพทเทิร์น” (งานวิจัยของวิทยานิพนธ์ฉบับนี้)

ตารางที่ 2.3 สรุปเปรียบเทียบงานวิจัยที่เกี่ยวข้องกับการเปรียบเทียบและการประเมิลผลของภาษาที่ใช้ในการเขียนโปรแกรม

กระบวนการทำงาน	งานวิจัยที่เกี่ยวข้อง				
	ก	ข	ค	ง	จ
วัตถุประสงค์ของการวิจัย					
เพื่อการประเมินผลของภาษา	✓			✓	✓
เพื่อการเปรียบเทียบภาษาเชิงวัตถุ		✓			

ตารางที่ 2.3 สรุปเปรียบเทียบงานวิจัยที่เกี่ยวข้องกับการเปรียบเทียบและการประเมินผลของภาษาที่ใช้ในการเขียนโปรแกรม (ต่อ)

กระบวนการทำงาน	งานวิจัยที่เกี่ยวข้อง				
	ก	ข	ค	ง	จ
เพื่อเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรม			✓		✓
เพื่อศึกษาประสิทธิภาพการทำงานของภาษา					✓
ภาษาที่ใช้ในการเปรียบเทียบ/การประเมิน					
C#	✓		✓		
C++		✓			
Dart					✓
Eiffel		✓			
Go				✓	
Java		✓	✓		✓
Smalltalk		✓			
ลักษณะของการเปรียบเทียบ/ประเมิน					
การใช้แบบสอบถาม	✓				
การเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรม			✓		✓
การเปรียบเทียบด้วย LOC (Line Of Code)			✓		✓
การประเมินผลจากห้องปฏิบัติการ	✓				
การประเมินผลด้วยดีไซน์แพทเทิร์น				✓	✓
กรอบงาน HotDraw					
พัฒนากรอบงานด้วยภาษาที่ใช้ในงานวิจัย				✓	
เครื่องมือสำหรับทดสอบประสิทธิภาพของเวลาที่ใช้ประมวลผล					
DeltaBlue Benchmark					✓

บทที่ 3
วิธีดำเนินการวิจัย

ในบทนี้จะเป็นการนำเสนอวิธีการดำเนินการวิจัย โดยในหัวข้อที่ 3.1 เป็นการนำเสนอระเบียบวิธีในการดำเนินการวิจัย หัวข้อที่ 3.2 เป็นการนำเสนอการออกแบบและพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นทั้ง 23 แพทเทิร์น หัวข้อที่ 3.3 นำเสนอวิธีการดำเนินการเปรียบเทียบ การประเมิน และการทดสอบประสิทธิภาพของเวลาที่ใช้ในการประมวลผล และหัวข้อที่ 3.4 กล่าวถึงเครื่องมือที่ใช้ในการวิจัย โดยมีรายละเอียดของแต่ละหัวข้อดังนี้

3.1 ระเบียบวิธีวิจัย

1. ศึกษาค้นคว้าและรวบรวมงานวิจัยที่เกี่ยวข้องกับการเปรียบเทียบและการประเมินผลภาษาที่ใช้ในการเขียนโปรแกรม
2. ศึกษาค้นคว้าการเขียนโปรแกรมด้วยภาษาดาร์ท
3. ศึกษาค้นคว้าวัตถุประสงค์ แนวคิด โครงสร้าง และผลที่ตามมาจากดีไซด์แพทเทิร์นของ GoF ทั้ง 23 แพทเทิร์น
4. ออกแบบแผงผังคลาสจากโครงสร้างหลักและพัฒนาตัวอย่างการใช้งานแพทเทิร์นของดีไซน์แพทเทิร์นทั้ง 23 แพทเทิร์น ด้วยภาษาจาวา และภาษาดาร์ท
5. เปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรม ประเมินคุณสมบัติ คุณลักษณะของภาษาดาร์ท และทดสอบประสิทธิภาพในลักษณะของเวลาที่ใช้ในการประมวลผลภาษาดาร์ทด้วยชุดทดสอบมาตรฐาน
6. วิเคราะห์ และสรุปผลการทำวิจัย

3.2 การออกแบบและพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์น

ในขั้นตอนของการออกแบบผู้วิจัยได้ออกแบบโครงสร้างแผงผังคลาสตามสัญลักษณ์ UML ให้สอดคล้องกับโครงสร้างของแต่ละแพทเทิร์นตามที่ได้นำเสนอไว้ในบทที่ 2 หัวข้อที่ 2.2 เรื่องดีไซน์แพทเทิร์น การพัฒนาตัวอย่างการใช้งานในแต่ละดีไซน์แพทเทิร์นจะทำการพัฒนาตามโครงสร้างแผงผังคลาสที่จะทำการออกแบบต่อไปนี้ในภาษาจาวา และภาษาดาร์ท โดยภาษาดาร์ทจะถูกพัฒนาขึ้นจากการปรับปรุงโครงสร้างของแพทเทิร์นที่สามารถปรับปรุงให้มีการนำซอร์สโค้ดกลับไปใช้ซ้ำได้

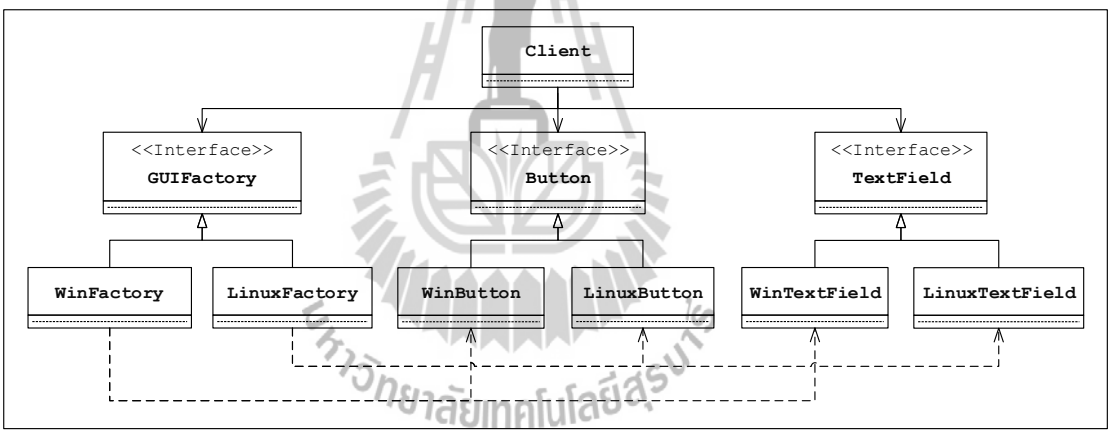
การออกแบบโครงสร้างแผงผังคลาสสำหรับการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นทั้ง 23 แพทเทิร์น จะทำการออกแบบโดยการแบ่งตามวัตถุประสงค์ที่ได้กำหนดไว้โดย GoF ดังต่อไปนี้

3.2.1 แพตเทิร์นเชิงการสร้าง

แพตเทิร์นการสร้างอ็อบเจกต์เป็นดีไซน์แพทเทิร์นสำหรับเชิงการสร้าง และการแก้ไขปัญหาในการสร้างอ๊อบเจกต์

1. แพทเทิร์นโรงงานเชิงนามธรรม

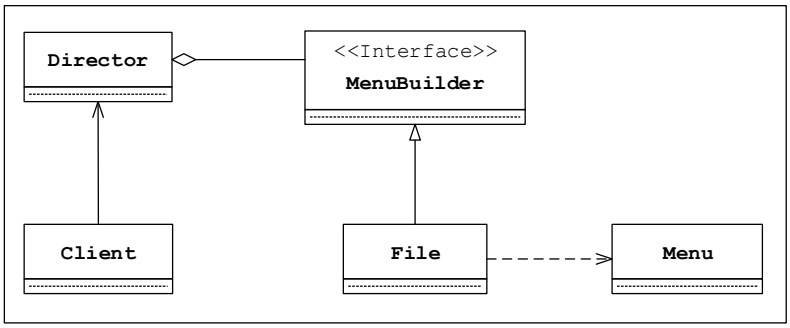
เป็นแพทเทิร์นสำหรับการประกาศอินเตอร์เฟซ เพื่อสร้างอ๊อบเจกต์ผลิตภัณฑ์ด้วย แผงผังคลาสแพทเทิร์นโรงงานเชิงนามธรรมได้ทำการออกแบบดังแสดงในรูปที่ 3.1



รูปที่ 3.1 แผงผังคลาสแพทเทิร์นโรงงานเชิงนามธรรม

2. แพทเทิร์นตัวสร้าง

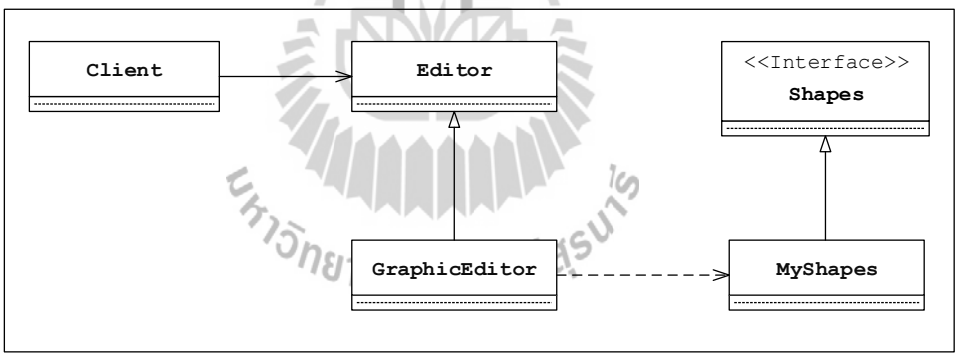
เป็นแพทเทิร์นที่นำเสนอวิธีการที่จะสร้างอ๊อบเจกต์องค์ประกอบ (Composite object) แผงผังคลาสแพทเทิร์นตัวสร้างได้ทำการออกแบบดังแสดงในรูปที่ 3.2



รูปที่ 3.2 แผงผังคลาสแพทเทิร์นตัวสร้าง

3. แพทเทิร์นเมธอดโรงงาน

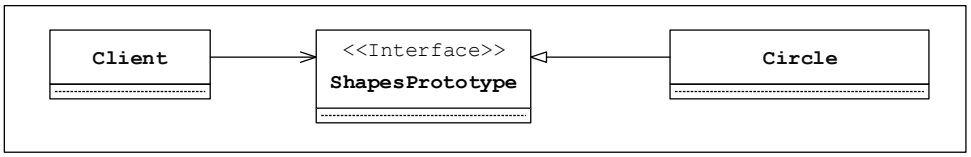
เป็นแพทเทิร์นที่กำหนดให้ซับคลาสตัดสินใจเองว่าจะสร้างอ๊อบเจกต์จากคลาสใด แผงผังคลาสแพทเทิร์นเมธอดโรงงานได้ทำการออกแบบดังแสดงในรูปที่ 3.3



รูปที่ 3.3 แผงผังคลาสแพทเทิร์นเมธอดโรงงาน

4. แพทเทิร์นต้นแบบ

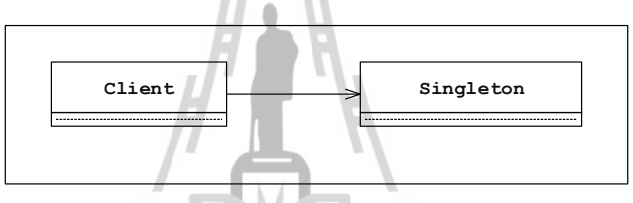
เป็นแพทเทิร์นที่ใช้สร้างอ๊อบเจกต์โดยการโคลนจากอ๊อบเจกต์แผงผังคลาสแพทเทิร์นต้นแบบได้ทำการออกแบบดังแสดงในรูปที่ 3.4



รูปที่ 3.4 แผงผังคลาสแพทเทิร์นต้นแบบ

5. แพทเทิร์นเดี่ยว

เป็นแพทเทิร์นที่กำหนดให้ในหนึ่งคลาสมีเพียงหนึ่งอินสแตนซ์เท่านั้น แผงผังคลาสแพทเทิร์นเดี่ยวได้ทำการออกแบบดังแสดงในรูปที่ 3.5



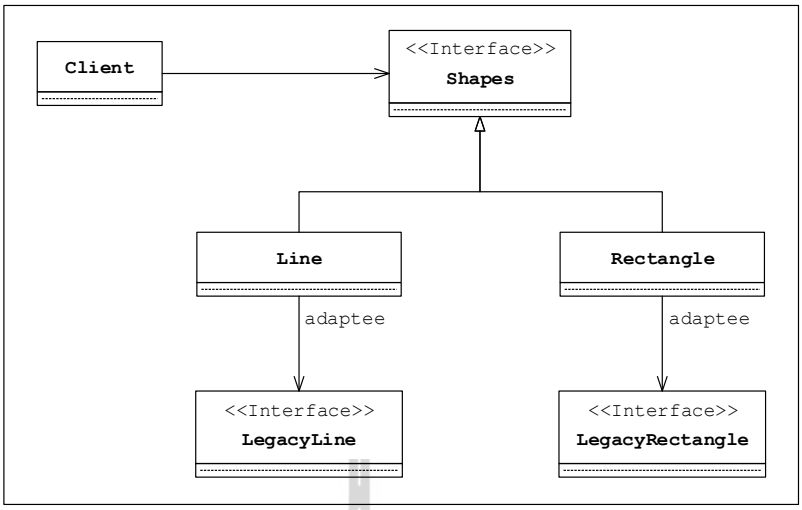
รูปที่ 3.5 แผงผังคลาสแพทเทิร์นเดี่ยว

3.2.2 แพตเทิร์นโครงสร้าง

แพตเทิร์นโครงสร้าง เป็นดีไซน์แพทเทิร์นสำหรับใช้ในการแก้ไขปัญหาเกี่ยวกับโครงสร้างและความสัมพันธ์ระหว่างอ๊อบเจกต์

1. แพทเทิร์นตัวแปลง

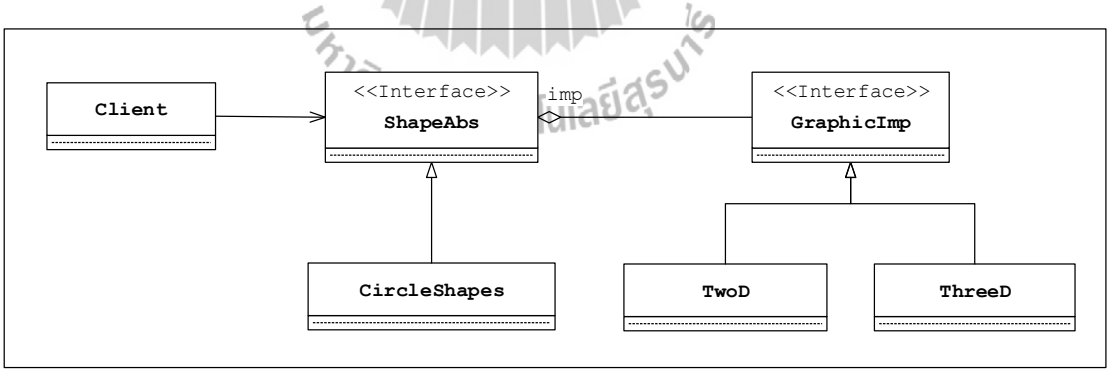
แพทเทิร์นสำหรับแปลงอินเตอร์เฟซของอ๊อบเจกต์ แผงผังคลาสแพทเทิร์นตัวแปลงได้ทำการออกแบบดังแสดงในรูปที่ 3.6



รูปที่ 3.6 แผงผังคลาสแพทเทิร์นตัวแปลง

2. แพทเทิร์นบริดจ์

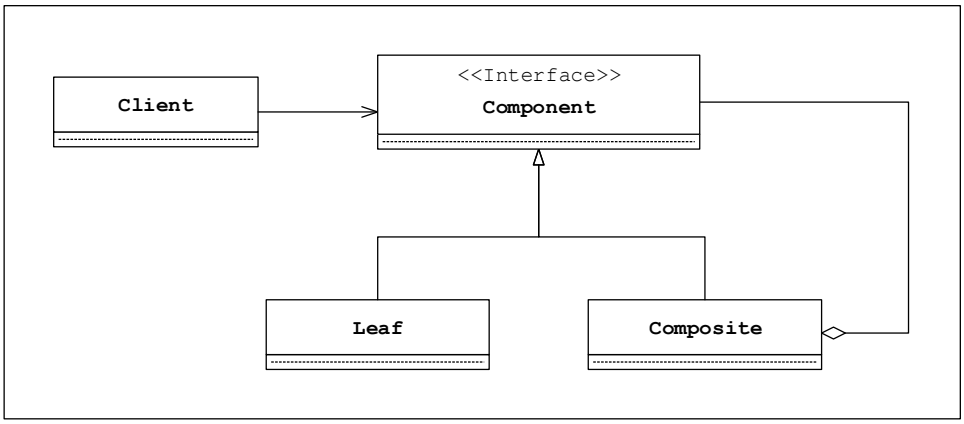
เป็นแพทเทิร์นที่ใช้สร้างอ๊อบเจกต์โดยกำหนดให้คลาสนามธรรมและคลาสทำให้เกิดผลเป็นอิสระต่อกัน แผงผังคลาสแพทเทิร์นบริดจ์ได้ทำการออกแบบดังแสดงในรูปที่ 3.7



รูปที่ 3.7 แผงผังคลาสแพทเทิร์นบริดจ์

3. แพทเทิร์นเชิงประกอบ

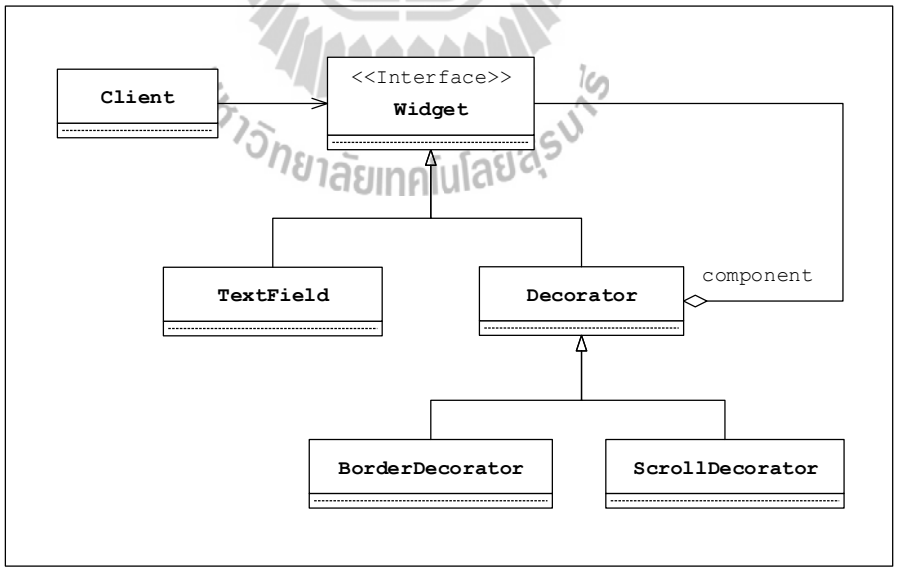
เป็นแพทเทิร์นที่ทำให้สามารถจัดการกับอ๊อบเจกต์ที่มีโครงสร้างแบบต้นไม้ได้ด้วยวิธีเดียวกัน แผงผังคลาสแพทเทิร์นเชิงประกอบได้ทำการออกแบบดังแสดงในรูปที่ 3.8



รูปที่ 3.8 แผงผังคลาสแพทเทิร์นเชิงประกอบ

4. แพทเทิร์นตัวตกแต่ง

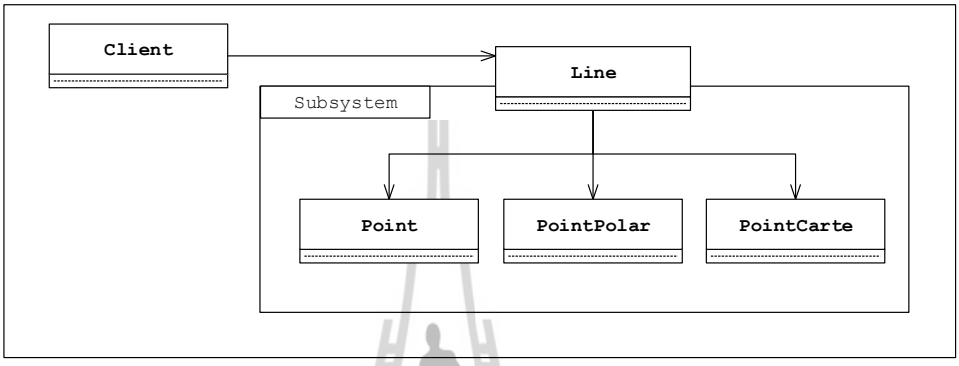
เป็นแพทเทิร์นสำหรับการเพิ่มพฤติกรรมแบบเชิงพลวัตไปยังอ๊อบเจกต์ แผงผังคลาสแพทเทิร์นตัวตกแต่งได้ทำการออกแบบดังแสดงในรูปที่ 3.9



รูปที่ 3.9 แผงผังคลาสแพทเทิร์นตัวตกแต่ง

5. แพทเทิร์นส่วนหน้า

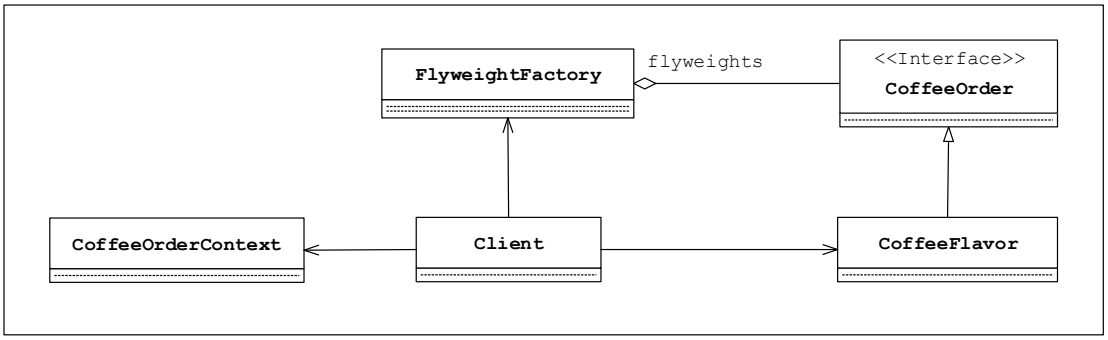
เป็นแพทเทิร์นที่ทำหน้าที่กำหนดอินเตอร์เฟสสำหรับการเข้าใช้ระบบย่อย แผงผังคลาสแพทเทิร์นส่วนหน้าได้ทำการออกแบบดังแสดงในรูปที่ 3.10



รูปที่ 3.10 แผงผังคลาสแพทเทิร์นส่วนหน้า

6. แพทเทิร์นน้ำหนักเบา

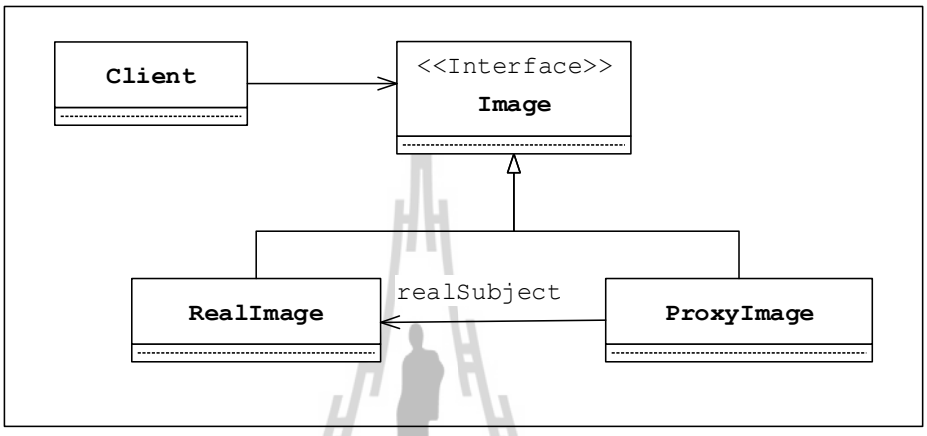
เป็นแพทเทิร์นที่ใช้ลักษณะของการใช้ทรัพยากรร่วมกัน เพื่อช่วยลดค่าใช้จ่ายในการจัดเก็บอ๊อบเจกต์ แผงผังคลาสแพทเทิร์นน้ำหนักเบาได้ทำการออกแบบดังแสดงในรูปที่ 3.11



รูปที่ 3.11 แผงผังคลาสแพทเทิร์นน้ำหนักเบา

7. แพทเทิร์นตัวแทน

เป็นแพทเทิร์นที่จะสร้างอ๊อบเจกต์ตัวแทนในการเข้าถึงอ๊อบเจกต์จริง แผงผังคลาสแพทเทิร์นตัวแทนได้ทำการออกแบบดังแสดงในรูปที่ 3.12



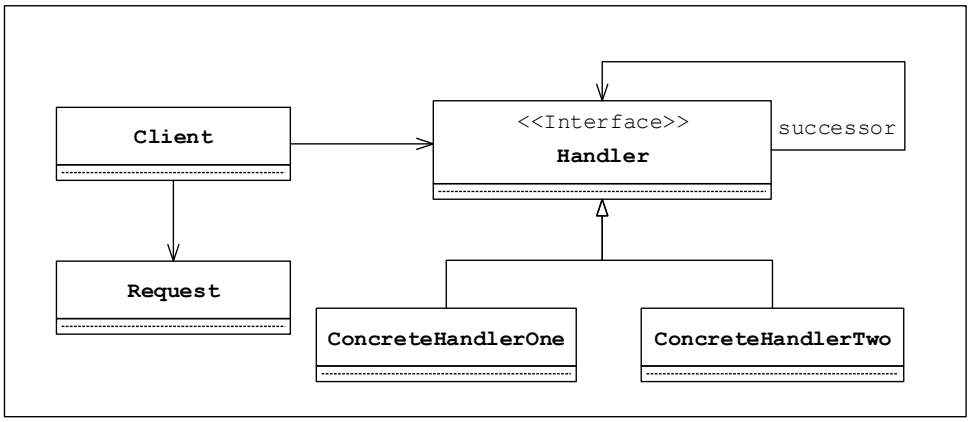
รูปที่ 3.12 แผงผังคลาสแพทเทิร์นตัวแทน

3.2.3 แพตเทิร์นพฤติกรรม

แพตเทิร์นพฤติกรรม เป็นดีไซน์แพทเทิร์นสำหรับการแก้ไขปัญหาที่เกี่ยวกับพฤติกรรมของอ๊อบเจกต์ และการติดต่อสื่อสารกันระหว่างอ๊อบเจกต์

1. แพทเทิร์นห่วงโซ่ความรับผิดชอบ

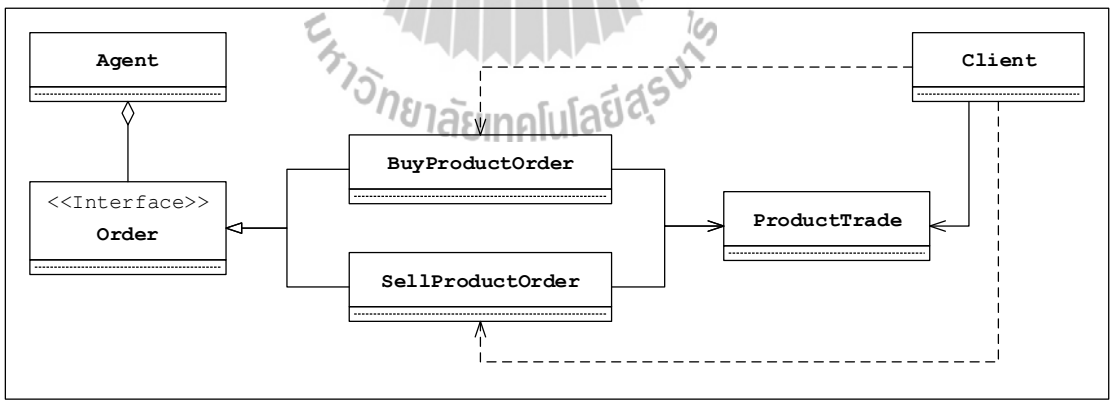
เป็นแพทเทิร์นที่กำหนดอ๊อบเจกต์ให้สามารถทำการตอบสนองการร้องขอได้สมบูรณ์ แผงผังคลาสแพทเทิร์นห่วงโซ่ความรับผิดชอบได้ทำการออกแบบดังแสดงในรูปที่ 3.13



รูปที่ 3.13 แผงผังคลาสแพทเทิร์นห่วงโซ่ความรับผิดชอบ

2. แพทเทิร์นคำสั่ง

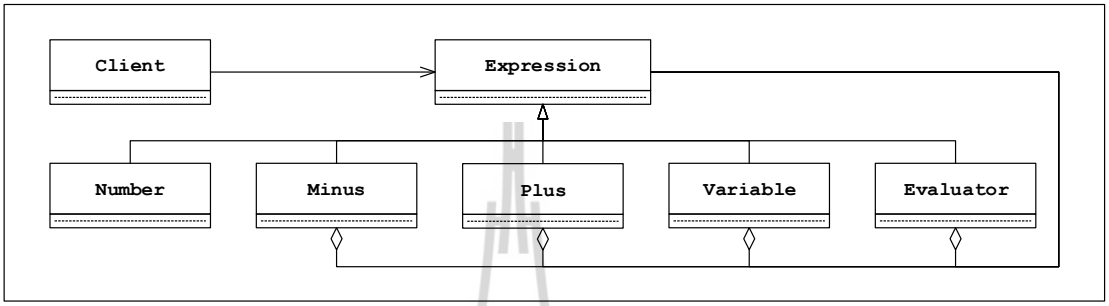
เป็นแพทเทิร์นที่จะทำการปกป้องการร้องขอให้เป็นอ๊อบเจกต์ เพื่อให้เมื่อมีการร้องขอได้รับการจัดการจนเสร็จสิ้น แผงผังคลาสแพทเทิร์นคำสั่งได้ทำการออกแบบดังแสดงในรูปที่ 3.14



รูปที่ 3.14 แผงผังคลาสแพทเทิร์นคำสั่ง

3. แพทเทิร์นตัวแปล

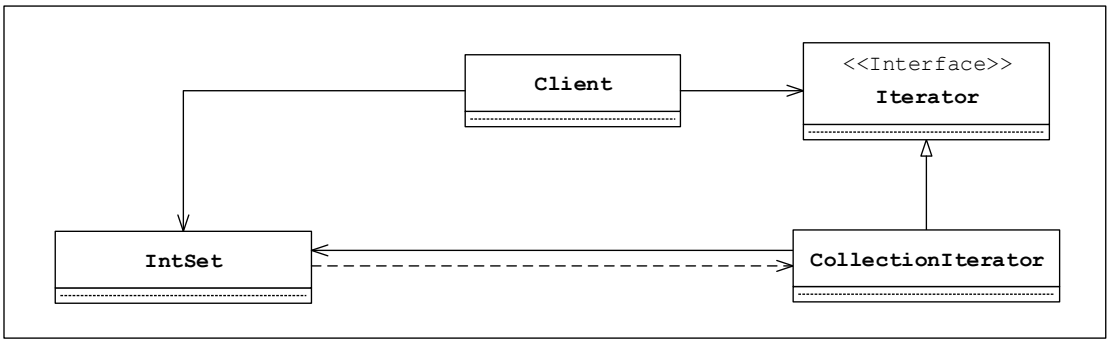
เป็นแพทเทิร์นที่กำหนดตัวแทนสำหรับไวยากรณ์และการตีความของภาษา แผงผังคลาสแพทเทิร์นตัวแปลได้ทำการออกแบบดังแสดงในรูปที่ 3.15



รูปที่ 3.15 แผงผังคลาสแพทเทิร์นตัวแปล

4. แพทเทิร์นตัววนซ้ำ

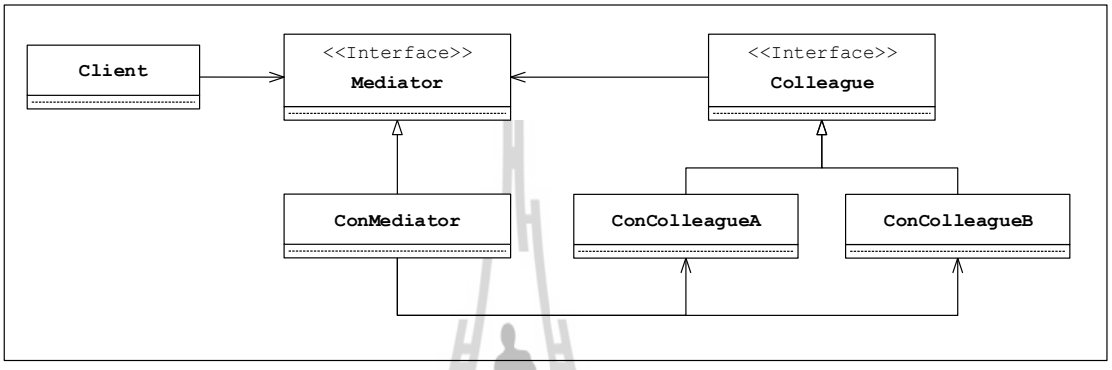
เป็นแพทเทิร์นที่นำเสนอวิธีการในการเข้าถึงองค์ประกอบของอ๊อบเจกต์ แผงผังคลาสแพทเทิร์นตัววนซ้ำได้ทำการออกแบบดังแสดงในรูปที่ 3.16



รูปที่ 3.16 แผงผังคลาสแพทเทิร์นตัววนซ้ำ

5. แพทเทิร์นตัวกลาง

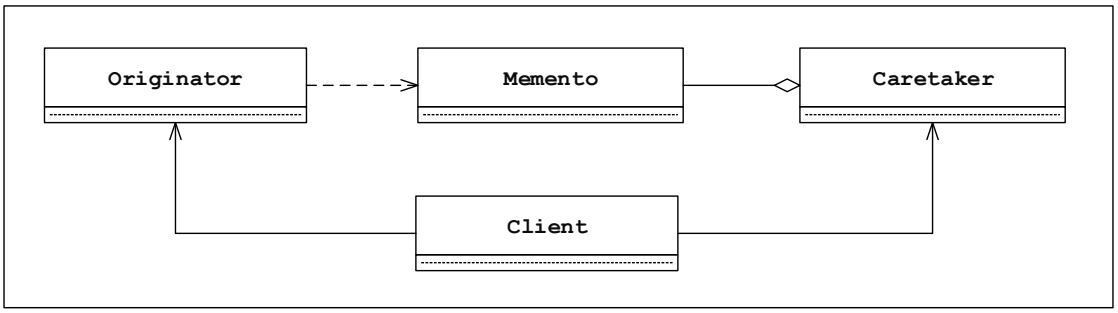
เป็นแพทเทิร์นนำเสนอวิธีการในการติดต่อสื่อสารกับอ๊อบเจกต์อื่น ๆ ด้วยการใช้ตัวกลาง แผงผังคลาสแพทเทิร์นตัวกลางได้ทำการออกแบบดังแสดงในรูปที่ 3.17



รูปที่ 3.17 แผงผังคลาสแพทเทิร์นตัวกลาง

6. แพทเทิร์นที่ระลึก

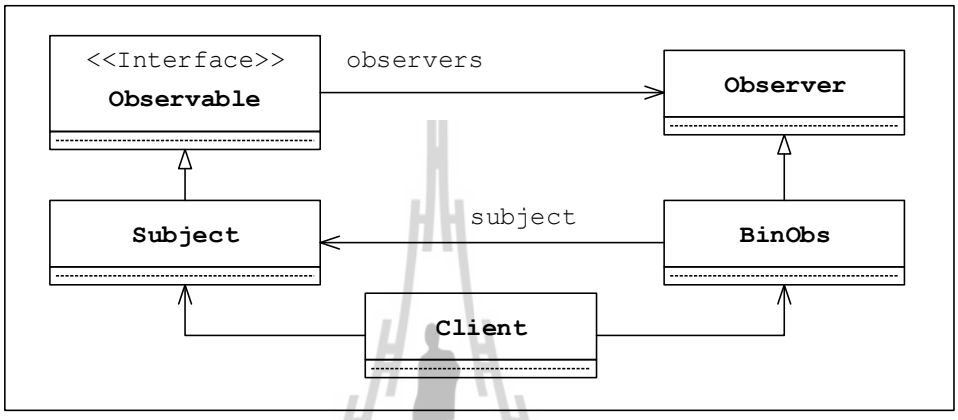
เป็นแพทเทิร์นที่จะทำการเก็บสถานะภายในไว้ที่อ๊อบเจกต์ภายนอก แผงผังคลาสแพทเทิร์นที่ระลึกได้ทำการออกแบบดังแสดงในรูปที่ 3.18



รูปที่ 3.18 แผงผังคลาสแพทเทิร์นที่ระลึก

7. แพทเทิร์นตัวสังเกตการณ์

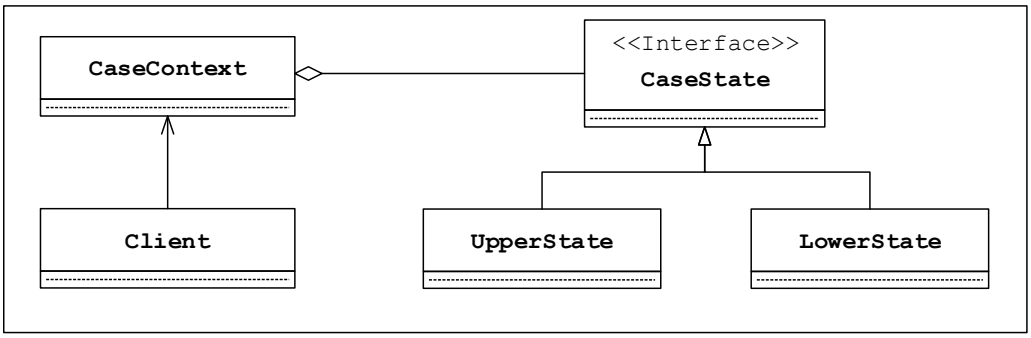
เป็นแพทเทิร์นจะกำหนดอ๊อบเจกต์ให้มีความสัมพันธ์แบบหนึ่งต่อกลุ่ม แผงผังคลาสแพทเทิร์นตัวสังเกตการณ์ได้ทำการออกแบบดังแสดงในรูปที่ 3.19



รูปที่ 3.19 แผงผังคลาสแพทเทิร์นตัวสังเกตการณ์

8. แพทเทิร์นสถานะ

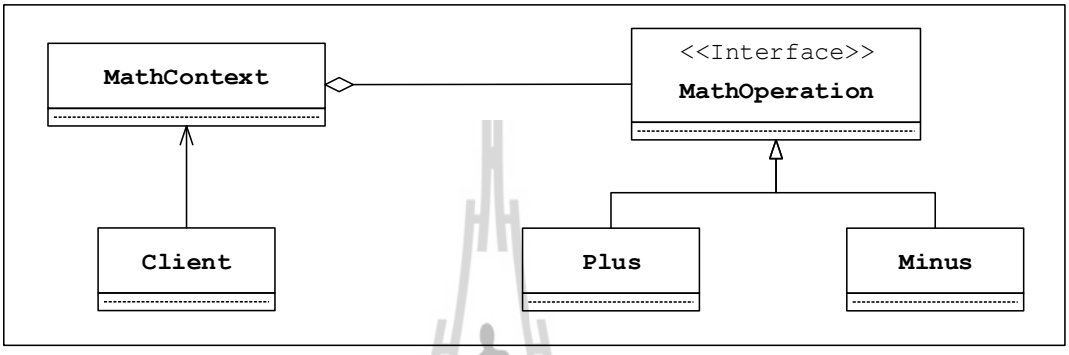
เป็นแพทเทิร์นที่จะยอมให้พฤติกรรมของอ๊อบเจกต์เปลี่ยนแปลง ก็ต่อเมื่อมีการเปลี่ยนแปลงสถานะภายใน แผงผังคลาสแพทเทิร์นสถานะได้ทำการออกแบบดังแสดงในรูปที่ 3.20



รูปที่ 3.20 แผงผังคลาสแพทเทิร์นสถานะ

9. แพทเทิร์นกลยุทธ์

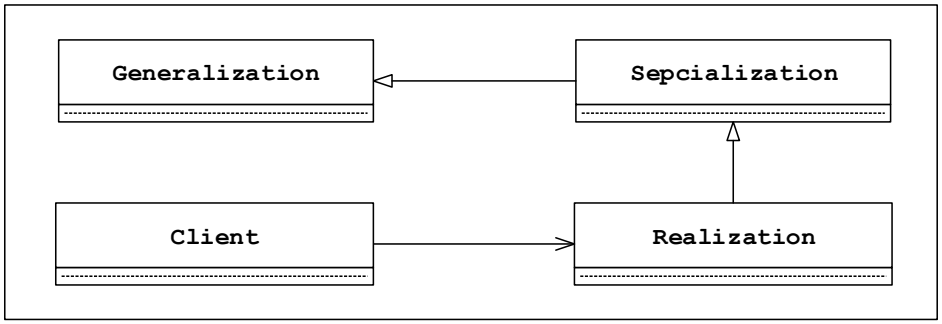
เป็นแพทเทิร์นที่กำหนดอัลกอริทึมและทำการปกป้องทุกอัลกอริทึม แผงผังคลาสแพทเทิร์นกลยุทธ์ได้ทำการออกแบบดังแสดงในรูปที่ 3.21



รูปที่ 3.21 แผงผังคลาสแพทเทิร์นกลยุทธ์

10. แพทเทิร์นเมธอดแม่แบบ

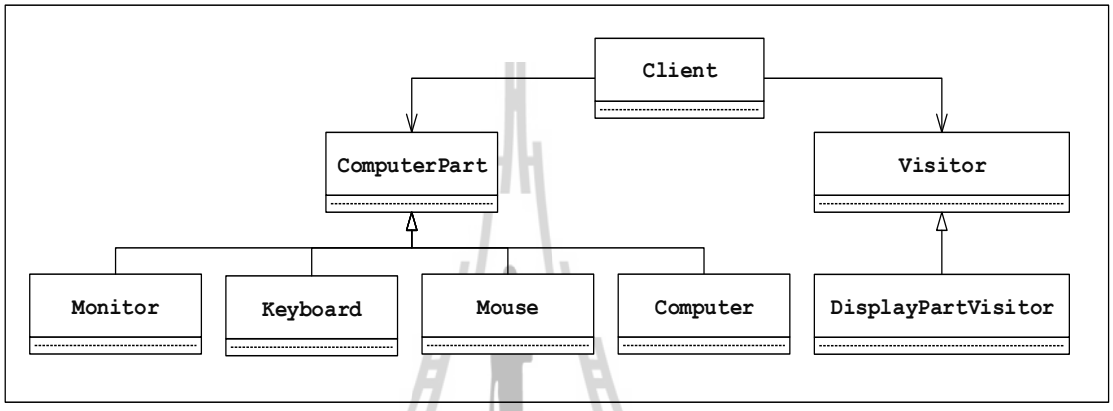
เป็นแพทเทิร์นสำหรับการกำหนดขั้นตอนของอัลกอริทึม แผงผังคลาสแพทเทิร์นเมธอดแม่แบบได้ทำการออกแบบดังแสดงในรูปที่ 3.22



รูปที่ 3.22 แผงผังคลาสแพทเทิร์นเมธอดแม่แบบ

11. แพทเทิร์นตัวเยี่ยมเยือน

เป็นแพทเทิร์นที่สามารถทำให้การดำเนินงานที่จะนำไปใช้กับอ๊อบเจกต์ โดยที่ไม่จำเป็นต้องเปลี่ยนแปลงคลาส แผงผังคลาสแพทเทิร์นตัวเยี่ยมเยือนได้ทำการออกแบบดังแสดงในรูปที่ 3.23



รูปที่ 3.23 แผงผังคลาสแพทเทิร์นตัวเยี่ยมเยือน

3.3 การเปรียบเทียบแนวคิด การประเมินผล และการทดสอบประสิทธิภาพ

3.3.1 การเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรม

ในการศึกษาแนวคิดเบื้องต้นในการเขียนโปรแกรมที่จะนำมาเปรียบเทียบในงานวิจัยนี้ จะทำการศึกษาในลักษณะของการเขียนโปรแกรมอย่างง่าย เช่น การสร้างคลาส การใช้คุณสมบัติการสืบทอด และการเปรียบเทียบคุณลักษณะต่าง ๆ เช่น การแสดงผล คำสงวน ชนิดตัวแปร ตัวดำเนินการ เป็นต้น

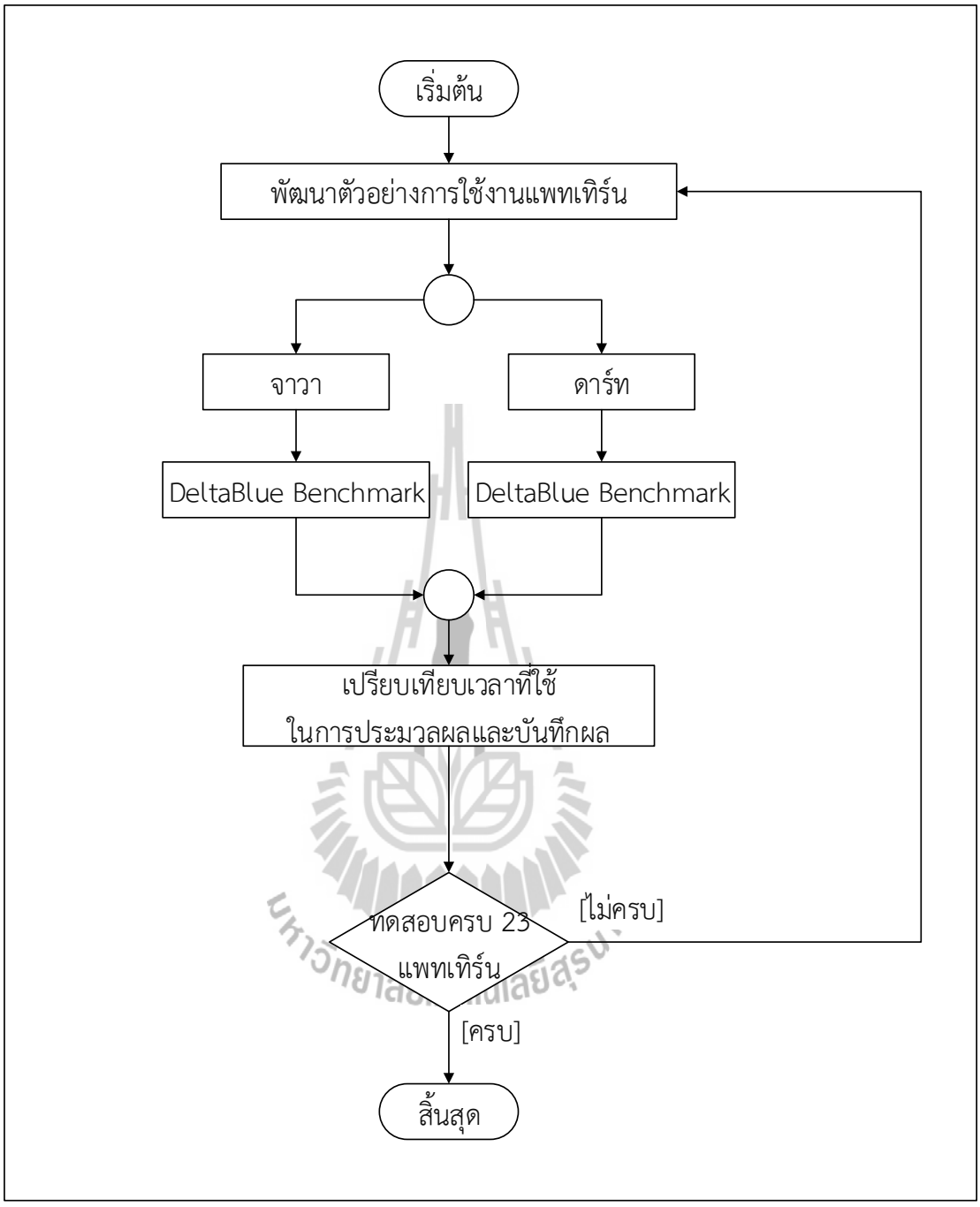
3.3.2 การประเมินคุณสมบัติและคุณลักษณะของภาษาดาร์ท

การประเมินคุณสมบัติและคุณลักษณะของภาษาดาร์ทจะประเมินในลักษณะของการเปรียบเทียบคุณสมบัติและคุณลักษณะของภาษาดาร์ทเทียบภาษาจาวา โดยจะศึกษาผ่านทางดีไซน์แพทเทิร์น เพื่อทำให้เห็นถึงความแตกต่างของทั้งสองภาษา ที่ได้ทำการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์น

การศึกษาจะทำในลักษณะของการเปรียบเทียบความสามารถของภาษาดาร์ทว่าสามารถเขียนโค้ดได้อย่างไรเช่น การแสดงผลออกทางบรรทัดคำสั่ง การนำโค้ดไปอิมพลีเมนต์ในภาษาจาวาต้องประกาศเป็น Interface แต่ในภาษาดาร์ทไม่มีการใช้ Interface แต่ก็สามารถทำการอิมพลีเมนต์จาก abstract class ได้ และการเปรียบเทียบโค้ดบรรทัดต่อบรรทัด เป็นต้น

3.3.3 การทดสอบประสิทธิภาพภาษาดาร์ท

การทดสอบประสิทธิภาพของภาษาในการวิจัยครั้งนี้จะศึกษาเฉพาะในส่วนของเวลาที่ใช้ในการประมวลผลของแต่ละแพทเทิร์นโดยใช้ DeltaBlue Benchmark สำหรับภาษาดาร์ท (Johnmccutchan, 2014) และภาษาจาวา (Botev, 2013) โดยศึกษาเวลาที่ใช้ในการประมวลผลเพื่อทำการเปรียบเทียบ จะทำการศึกษาจากตัวอย่างการใช้งานจากทั้ง 2 ภาษา ซึ่งประกอบไปด้วยรหัสต้นฉบับจำนวน 2 ชุดต่อหนึ่งแพทเทิร์นจากการออกแบบและพัฒนาในข้างต้น ทำการเปรียบเทียบครั้งละแพทเทิร์นให้ครบทั้ง 23 แพทเทิร์น ในการทดสอบจะทำโดยการนำรหัสต้นฉบับชุดที่ 1 รหัสต้นฉบับชุดที่ 2 เปรียบเทียบเวลาที่ใช้ในการประมวลผลแต่ละแพทเทิร์นเพื่อทดสอบประสิทธิภาพของภาษาดาร์ท ขั้นตอนในศึกษาเวลาที่ใช้ในการประมวลผลสำหรับการทดสอบประสิทธิภาพดังแสดงในรูปที่ 3.24



รูปที่ 3.24 ขั้นตอนการเปรียบเทียบประสิทธิภาพ

3.4 เครื่องมือที่ใช้ในการวิจัย

เครื่องมือและอุปกรณ์ที่ใช้สำหรับการวิจัยคือเครื่องคอมพิวเตอร์ส่วนบุคคลสำหรับการศึกษาการเปรียบเทียบและการประเมินผลการเขียนโปรแกรมภาษาดาร์ทด้วยดีไซด์แพทเทิร์น โดยรายละเอียดมีดังนี้

- หน่วยประมวลผลกลาง : Intel® Core™ i5-3230M CPU@2.6GHz
- หน่วยความจำสำรอง : 500 GB 5400 RPM
- หน่วยความจำหลัก : 4.00 GB DDR3
- ระบบปฏิบัติการ : Windows 8 Pro 64-bit Operating System
- เครื่องมือที่ใช้ในการพัฒนา : DartEditor 1.3.6 และ NetBeans IDE 7.3.1
- อุปกรณ์อื่น ๆ เช่น เมาส์ แป้นพิมพ์ เป็นต้น
- ภาษาดาร์ท รุ่นที่ 1.2
- ภาษาจาวา รุ่นที่ 1.7.0_40

บทที่ 4
ผลการวิจัยและอภิปรายผล

ในบทนี้เป็นการนำเสนอผลการวิจัยและการอภิปรายผล โดยในหัวข้อที่ 4.1 เป็นการนำเสนอการเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรมของภาษาดาร์ทและภาษาจาวา หัวข้อที่ 4.2 เป็นการนำเสนอผลการประเมินคุณสมบัติและคุณลักษณะของภาษาดาร์ท และหัวข้อที่ 4.3 เป็นการนำเสนอการทดสอบประสิทธิภาพของภาษาดาร์ทเมื่อเทียบกับภาษาจาวา โดยมีรายละเอียดของแต่ละหัวข้อดังนี้

4.1 การเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรม

ในการเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรม ผู้วิจัยได้นำเสนอการเปรียบเทียบในส่วนของการเขียนโปรแกรมอย่างง่าย คำสงวน แบบชนิดของข้อมูล และตัวดำเนินการ

4.1.1 การเขียนโปรแกรมอย่างง่าย

ในการเขียนโปรแกรมอย่างง่ายเพื่อนำเสนอในการศึกษาครั้งนี้ ผู้วิจัยเลือกการนำเสนอเพียงการแสดงผลเท่านั้น จากรูปที่ 4.1 เป็นการเขียนโปรแกรมเพื่อให้แสดงข้อความ "Hello World!!" โดยการเริ่มเขียนโปรแกรมจาวาจะใช้คำสงวนว่า “package” ในการมาใช้ในการจัดระเบียบคลาสและอินเตอร์เฟซ ในขณะที่ดาร์ทใช้คำสงวนว่า “library” จาวากำหนดให้ทุกเมธอดต้องอยู่ภายในคลาส รวมถึงเมธอด main() ด้วย ส่วนดาร์ทไม่จำเป็นต้องกำหนดให้ทุกเมธอดอยู่ภายในคลาส โดยเฉพาะเมธอด main() จะไม่สามารถอยู่ในคลาสได้ ส่วนการแสดงผลทางบรรทัดคำสั่งนั้น จาวาจะใช้คำสั่ง System.out.println() ซึ่งเป็นคำสั่งให้แสดงผลแล้วทำการขึ้นบรรทัดใหม่ ขณะที่ดาร์ทใช้คำสั่ง print() ซึ่งให้ผลลัพธ์เช่นเดียวกันกับจาวา

Java	Dart

package HelloWorld;
class HelloWorld {
  public static void main(String[] args){
  System.out.println("Hello World!!");
  }
}
	library HelloWorld;
	
	void main() {
	  print("Hello World!!");
	}

Output :
  Hello World!!
	Output :
	  Hello World!!

รูปที่ 4.1 การเขียนโปรแกรมอย่างง่าย

4.1.2 คำสงวน

เนื่องจากดาร์ทเป็นภาษาเขียนโปรแกรมที่มีจุดประสงค์หนึ่งในการออกแบบให้เป็นภาษาที่ง่ายต่อการเรียนรู้ ผลที่ได้จากจุดประสงค์ในการออกแบบดังกล่าวทำให้วากยสัมพันธ์ของภาษาดาร์ทใกล้เคียงกับภาษาเขียนโปรแกรมอื่น ๆ มาก โดยเฉพาะภาษาจาวา ทำให้คำสงวน (Oracle, 2014a ; Walrath and et al., 2013c) ทั้ง 2 ภาษาโดยส่วนใหญ่แล้วเหมือนกัน ดังแสดงในตารางที่ 4.1

ตารางที่ 4.1 เปรียบเทียบคำสงวนในภาษาจาวาและภาษาดาร์ท

จาวา	ดาร์ท	จาวา	ดาร์ท	จาวา	ดาร์ท
abstract	abstract	final	final	public	
	as	finally	finally		rethrow
assert	assert	float		return	return
boolean		for	for		set
break	break		get	short	
byte		goto*		static	static
case	case	if	if	strictfp	
catch	catch	implements	implements	super	super
char		import	import	switch	switch
class	class		in	synchronized	
const*	const	instanceof		this	this
continue	continue	int		throw	throw

ตารางที่ 4.1 เปรียบเทียบคำสงวนในภาษาจาวาและภาษาดาร์ท (ต่อ)

จาวา	ดาร์ท	จาวา	ดาร์ท	จาวา	ดาร์ท
default	default	interface		throws	
do	do		is	transient	
double		long		true	true
	dynamic	native		try	try
else	else	new	new		typedef
enum	enum	null	null		var
	export		operator	void	void
	external	package	library	volatile	
extends	extends		part	while	while
	factory	private			with
false	false	protected			

* ปัจจุบันไม่ได้มีการนำไปใช้งานแล้ว

4.1.3 แบบชนิดของข้อมูล

ภาษาดาร์ทเป็นภาษาที่ถูกพัฒนาให้มีการกำหนดแบบชนิดเชิงทางเลือก แต่ก็มีการสนับสนุนแบบชนิดเชิงสถิตย์เช่นเดียวกัน ทำให้มีแบบชนิดเพียงบางส่วนที่คล้ายกับจาวา ดังแสดงในตารางที่ 4.2 ซึ่งเป็นแบบชนิดของข้อมูลทั้ง 2 ภาษา (Oracle, 2014b ; Walrath and et al., 2013c)

ตารางที่ 4.2 แบบชนิดในภาษาจาวาและภาษาดาร์ท

จาวา	ดาร์ท	รายละเอียด
byte		มีขนาด 8 บิต
short		มีขนาด 16 บิต
	int	ขนาดจะขึ้นอยู่กับเครื่องจักรเสมือน
int		มีขนาด 32 บิต
long		มีขนาด 64 บิต
float		มีขนาด 32-bit (IEEE 754 floating point)
double	double	มีขนาด 64-bit (IEEE 754 floating point)
	num	อ๊อบเจกต์สำหรับเก็บค่าตัวเลข

ตารางที่ 4.2 แบบชนิดในภาษาจาวาและภาษาดาร์ท (ต่อ)

จาวา	ดาร์ท	รายละเอียด
boolean	bool	มีค่าเป็น true และ false
char		ตัวอักษรเดี่ยวขนาด 16-bit (Unicode character)
String	String	รหัสสตริงแบบ UTF-16
	list	เก็บข้อมูลเป็นกลุ่ม
	map	อ๊อบเจกต์สำหรับเก็บตัวกำกับและค่า
	symbols	การสร้างสัญลักษณ์ให้กับการประกาศ (คลาส ขอบเขต ฯลฯ)

4.1.4 ตัวดำเนินการ

ตัวดำเนินการ (Oracle, 2014c ; Walrath and et al., 2013c) ทั้งในภาษาดาร์ทและภาษาจาวาส่วนมากแล้วจะมีความใกล้เคียงกันมาก มีเพียงบางส่วนของตัวดำเนินการเท่านั้นที่ทั้ง 2 ภาษาแตกต่างกัน ดังแสดงในตารางที่ 4.3

ตารางที่ 4.3 ตัวดำเนินการในภาษาจาวาและภาษาดาร์ท

ตัวดำเนินการ	จาวา	ดาร์ท	รายละเอียด
คณิตศาสตร์	+	+	บวก
	-	-	ลบ
	*	*	คูณ
	/		หาร (ผลลัพธ์ขึ้นอยู่กับการกำหนดชนิดของตัวแปร)
		/	หาร (ผลลัพธ์จะเป็นเลขที่มีจุดทศนิยม : double)
		~/	หาร (ผลลัพธ์จะเป็นเลขจำนวนเต็ม)
	%	%	หารเพื่อเก็บเศษเหลือ (Modulo)
การดำเนินการชนิดเอกภาค (Unary Operators)	+	+	ตัวดำเนินการชนิดเอกภาคที่เป็นบวก
	-	-	ตัวดำเนินการชนิดเอกภาคที่เป็นลบ
	++	++	การเพิ่มค่าทีละ 1
	--	--	การลดค่าทีละ 1
ความเท่าเทียมและความสัมพันธ์	==	==	เท่ากับ
	!=	!=	ไม่เท่ากับ
	>	>	มากกว่า

ตารางที่ 4.3 ตัวดำเนินการในภาษาจาวาและภาษาดาร์ท (ต่อ)

ตัวดำเนินการ	จาวา	ดาร์ท	รายละเอียด
ความเท่าเทียม และความสัมพันธ์ (ต่อ)	>=	>=	มากกว่า หรือ เท่ากับ
	<	<	น้อยกว่า
	<=	<=	น้อยกว่า หรือ เท่ากับ
เงื่อนไข (Conditional)	? :	? :	คำสั่ง if-than-else อย่างย่อ
ตัวทดสอบแบบชนิด (Type Test)		as	ตรวจสอบชนิดของตัวแปร
	instanceof		เปรียบเทียบว่าเป็นอ๊อบเจกต์ที่ระบุหรือไม่
		is	เป็นจริง (True) ถ้าเป็นอ๊อบเจกต์เดียวกับที่ระบุ
		is!	เป็นเท็จ (False) ถ้าเป็นอ๊อบเจกต์เดียวกับที่ระบุ
ตรรกะ (Logical)		!	จะเป็นค่าตรงข้ามกับเงื่อนไขที่กำหนด
	||	||	ถ้าทั้งสองเงื่อนไขเป็นเท็จทั้งคู่ ผลที่ได้จะเป็นเท็จ นอกนั้นเป็นจริงทั้งหมด
	&&	&&	ถ้าทั้งสองเงื่อนไขเป็นจริงทั้งคู่ ผลที่ได้จะเป็นจริง นอกนั้นเป็นเท็จทั้งหมด
การกำหนดค่า	=	=	การกำหนดค่าอย่างง่าย (Simple Assignment)
การกำหนดและการเลื่อนค่าระดับบิต (Bitwise and Shift)	&	&	และ (AND)
	|	|	หรือ (OR)
	^	^	ถ้าเงื่อนไขแรกกับเงื่อนไขหลังเหมือนกัน (จริง-จริง/เท็จ-เท็จ) ผลที่ได้จะเป็นเท็จ นอกนั้นเป็นจริง (XOR)
	~	~	นิเสธ (NOT)
	<<	<<	เลื่อนไปทางซ้าย (Shift left)
	>>	>>	เลื่อนไปทางขวา (Shift right)
	>>>		เลื่อนไปทางขวา ไม่ระบุเครื่องหมาย (Unsigned)

4.2 การประเมินคุณสมบัติและคุณลักษณะของภาษาดาร์ท

ในส่วนนี้จะการศึกษาและประเมินคุณสมบัติและคุณลักษณะของภาษาดาร์ท โดยการเปรียบเทียบจากตัวอย่างดีไซน์แพทเทิร์นที่ได้ทำการพัฒนาขึ้นมาในภาษาจาวาและภาษาดาร์ท

4.2.1 แพตเทิร์นเชิงการสร้าง

1. แพทเทิร์นโรงงานเชิงนามธรรม

จากตารางที่ 4.4 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ของทั้ง 2 ภาษา คือ จาวามีการประกาศ interface GUIFactory ในบรรทัดที่ 2 ขณะที่ดาร์ทประกาศเป็น abstract class GUIFactory ในบรรทัดที่ 2 เนื่องจากในดาร์ทไม่มีคำสงวน interface แต่มีการใช้ abstract class แทนซึ่งดาร์ทสามารถนำไปใช้ได้ทั้ง implements และ extends

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นโรงงานเชิงนามธรรมในภาษาจาวาใช้ 69 บรรทัด ภาษาดาร์ทใช้เพียง 42 บรรทัด

ตารางที่ 4.4 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นโรงงานเชิงนามธรรม

จาวา		ดาร์ท	
1	package AbstractFactory;	1	library AbstractFactory;
2	interface GUIFactory {	2	abstract class GUIFactory {
3	Button createButtons();	3	Button createButtons();
4	TextField createTextFields();	4	TextField createTextFields();
5	}	5	}
6	interface Button {	6	abstract class Button {
7	void draw();	7	void draw();
8	}	8	}
9	interface TextField {	9	abstract class TextField {
10	void draw();	10	void draw();
11	}	11	}
12	class WinFactory implements GUIFactory {	12	class WinFactory implements GUIFactory {
13	@Override	13	Button createButtons() => new WinButton();
14	public Button createButtons() {	14	TextField createTextFields() => new WinTextField();
15	return new WinButton();	15	}
16	}		
17	@Override		
18	public TextField createTextFields() {		
19	return new WinTextField();		
20	}		
21	}		
22	class LinuxFactory implements GUIFactory {	16	class LinuxFactory implements GUIFactory {
23	@Override	17	Button createButtons() => new LinuxButton();

ตารางที่ 4.4 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นโรงงานเชิงนามธรรม (ต่อ)

จาวา		ดาร์ท	
24	public Button createButtons() {	18	TextField createTextFields() => new LinuxTextField();
25	return new LinuxButton();	19	}
26	}		
27	@Override		
28	public TextField createTextFields() {		
29	return new LinuxTextField();		
30	}		
31	}		
32	class WinButton implements Button {	20	class WinButton implements Button {
33	@Override	21	void draw() => print("Win Button");
34	public void draw() {	22	}
35	System.out.println("Win Button");		
36	}		
37	}		
38	class LinuxButton implements Button {	23	class LinuxButton implements Button {
39	@Override	24	void draw() => print("Linux Button");
40	public void draw() {	25	}
41	System.out.println("Ubuntu Button");		
42	}		
43	}		
44	class WinTextField implements TextField {	26	class WinTextField implements TextField {
45	@Override	27	void draw() => print("Win TextField");
46	public void draw() {	28	}
47	System.out.println("Win TextField");		
48	}		
49	}		
50	class LinuxTextField implements TextField {	29	class LinuxTextField implements TextField {
51	@Override	30	void draw() => print("Linux TextField");
52	public void draw() {	31	}
53	System.out.println("Ubuntu TextField");		
54	}		
55	}		
56	public class Client {		
57	public static void main(String[] args) {	32	void main(var x) {
58	GUIFactory os = createSpecOsFactory();	33	GUIFactory os = createSpecOsFactory();
59	os.createButtons().draw();	34	os.createButtons().draw();

ตารางที่ 4.4 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นโรงงานเชิงนามธรรม (ต่อ)

จาวา		ดาร์ท	
60	os.createTextFields().draw();	35	os.createTextFields().draw();
61	}	36	}
62	public static GUIFactory createSpecOsFactory() {	37	GUIFactory createSpecOsFactory() {
63	String os = "win";	38	String os = "win";
64	if ("win".equals(os))	39	if(os == "win")
65	return new WinFactory();	40	return new WinFactory();
66	else	41	else
67	return new LinuxFactory();	42	return new LinuxFactory();
68	}	43	}
69	}		

2. แพทเทิร์นตัวสร้าง

จากตารางที่ 4.5 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ของทั้ง 2 ภาษา คือ จาวามีการประกาศ abstract class MenuBuilder ในบรรทัดที่ 2 ซึ่งภายในประกอบด้วยเมธอดนามธรรม 2 เมธอด ดาร์ทก็มีการประกาศเช่นเดียวกันในบรรทัดที่ 2 แต่ลักษณะของวากยสัมพันธ์ของเมธอดนามธรรมทั้ง 2 ภาษาต่างกัน ในจาวาการประกาศเมธอดนามธรรมจะต้องมีคำสงวนว่า abstract กำกับไว้ด้วย เช่น public abstract void buildName(String name)ในบรรทัดที่ 3 ส่วนในดาร์ทไม่จำเป็นต้องมีการกำกับคำสงวนดังกล่าว เช่น void buildName(String name) ในบรรทัดที่ 3 จากตัวอย่างจะเห็นว่าจาวามีการระบุคำสงวน public ในขณะที่ดาร์ทไม่ได้ทำการระบุไว้ เพราะในดาร์ทหากไม่มีการระบุว่าเป็น private แล้ว ถือว่าเป็น public ทั้งหมด

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวสร้างในภาษาจาวาใช้ 60 บรรทัด ภาษาดาร์ทใช้เพียง 44 บรรทัด

ตารางที่ 4.5 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวสร้าง

จาวา		ดาร์ท	
1	package Builder;	1	library Builder;
2	abstract class MenuBuilder {	2	abstract class MenuBuilder {
3	public abstract void buildName(String name);	3	void buildName(String name);
4	public abstract void buildKeyShortcut(String key);	4	void buildKeyShortcut(String key);

ตารางที่ 4.5 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวสร้าง (ต่อ)

จาวา		ดาร์ท	
5	}	5	}
6	class Menus {	6	class Menus {
7	private String name;	7	String _name;
8	private String key;	8	String _key;
9	public void setName(String name) {	9	set Name (String name) => _name = name;
10	this.name = name;	10	set KeyShortcus (String key) => _key = key;
11	}	11	void showMenu() => print(this._name + "\t\t" + this._key);
12	public void setKeyShortcut(String key) {	12	}
13	this.key = key;		
14	}		
15	public void showMenu() {		
16	System.out.println(this.name +"\t\t" + this.key);		
17	}		
18	}	13	class Files extends MenuBuilder {
19	class Files extends MenuBuilder {	14	Menus _menu;
20	private Menus menu;	15	Files(){
21	Files(){	16	print("File");
22	System.out.println("File");	17	}
23	}	18	void buildName(String name) {
24	@Override	19	this._menu.Name = name;
25	public void buildName(String name) {	20	}
26	this.menu.setName(name);	21	void buildKeyShortcut(String key) {
27	}	22	this._menu.KeyShortcus = key;
28	@Override	23	}
29	public void buildKeyShortcut(String key) {	24	void createNewMenu() {
30	this.menu.setKeyShortcut(key);	25	this._menu = new Menus();
31	}	26	}
32	public void createNewMenu() {	27	Menus get Menu => this._menu;
33	this.menu = new Menus();	28	}
34	}		
35	public Menus getMenu() {		
36	return this.menu;		
37	}		
38	}	29	class Director {
39	class Director {	30	Files _files;
40	private Files file;	31	Menus get Menu => this._files.Menu;

ตารางที่ 4.5 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวสร้าง (ต่อ)

จาวา	ดาร์ท

41 public Menus getMenu() {
42 return this.file.getMenu();
43 }
44 public void setFile(Files file) {
45 this.file = file;
46 }
47 public void constructFile() {
48 this.file.createNewMenu();
49 this.file.buildName("New file...");
50 this.file.buildKeyShortcut("Ctrl + N");
51 }
52 }
53 public class Client {
54 public static void main(String args[]) {
55 Director director = new Director();
56 director.setFile(new Files());
57 director.constructFile();
58 director.getMenu().showMenu();
59 }
60 }
	32 set File(Files files) => _files = files;
	33 void constructFile() {
	34 this._files.createNewMenu();
	35 this._files.buildName("New file...");
	36 this._files.buildKeyShortcut("Ctrl + N");
	37 }
	38 }
	
	39 void main() {
	40 Director director = new Director();
	41 director.File = new Files();
	42 director.constructFile();
	43 director.Menu.showMenu();
	44 }

3. แพทเทิร์นเมธอดโรงงาน

จากตารางที่ 4.6 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ของทั้ง 2 ภาษา คือ เมธอด createShapes() ที่อยู่ภายในคลาสนามธรรม Editor ในบรรทัดที่ 6 ในจาวาประกาศโดยระบุให้มีการเข้าถึงแบบ protected แต่ในดาร์ทมีการกำหนดให้มีการเข้าถึงเพียง 2 ชนิดเท่านั้น นั่นคือ public และ private จึงได้มีการกำหนดการเข้าถึงเมธอด createShapes()ในดาร์ทเป็นแบบ private ดาร์ทไม่ได้ใช้คำสงวนดังกล่าวในการระบุการเข้าถึง แต่ใช้ในรูปแบบของเครื่องหมาย “_” (Underscore) เช่น _createShapes() ในบรรทัดที่ 12

ในส่วนของการแสดงผลบนบรรทัดคำสั่งของทั้ง 2 ภาษา มีข้อแตกต่างกันคือ สังเกตใน quadrate(int n) ที่อยู่ภายในคลาส MyShapes ในจาวาจะมีคำสั่งที่ใช้ในการแสดงผล System.out.print() คือเมื่อแสดงผลเสร็จแล้วไม่ต้องทำการขึ้นบรรทัดใหม่ และ System.out.println()

คือเมื่อแสดงผลเสร็จแล้วให้ทำการขึ้นบรรทัดใหม่ ในขณะที่ดาร์ทมีเพียง print() ซึ่งเมื่อแสดงผลเสร็จแล้วจะทำการขึ้นบรรทัดใหม่ทันที จึงต้องทำการเก็บค่าที่ต้องการแสดงผลไว้ในตัวแปรก่อน เพื่อให้ผลลัพธ์ที่แสดงเป็นไปตามความต้องการ

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นเมธอดโรงงานในภาษาจาวาใช้ 47 บรรทัด ภาษาดาร์ทใช้เพียง 42 บรรทัด

ตารางที่ 4.6 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นเมธอดโรงงาน

จาวา

```
1  package FactoryMethod;
2  interface Shapes {
3    public void quadrate(int n);
4    public void triangle(int n);
5  }
6  abstract class Editor {
7    public void operation() {
8      Shapes product = createShapes();
9      product.quadrate(5);
10     product.triangle(5);
11   }
12   protected abstract Shapes createShapes();
13 }
14 class MyShapes implements Shapes {
15   @Override
16   public void quadrate(int n) {
17     for (int i = 0; i < n; i++) {
18       for (int j = 0; j < n; j++) {
19         System.out.print("* ");
20       }
21       System.out.println();
22     }
23   }
24   @Override
25   public void triangle(int n) {
26     for (int i = 1; i <= n; i++) {
27       for (int j = i; j < n; j++) {
28         System.out.print(" ");
29       }
```

ดาร์ท

```
1  library FactorMethod;
2  abstract class Shapes {
3    void quadrate(int n);
4    void triangle(int n);
5  }
6  abstract class Editor {
7    void operation() {
8      Shapes product = _createShapes();
9      product.quadrate(5);
10     product.triangle(5);
11   }
12   Shapes _createShapes();
13 }
14 class MyShapes implements Shapes {
15   void quadrate(int n) {
16     String q = "";
17     for (int i = 0; i < n; i++) {
18       for (int j = 0; j < n; j++) {
19         q += "* ";
20       }
21       q += "\n";
22     }
23     print(q);
24   }
25   void triangle(int n) {
26     String t = "";
27       for (int i = 1; i <= n; i++) {
28         for (int j = i; j < n; j++) {
29           t += " ";
```

ตารางที่ 4.6 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นเมธอดโรงงาน (ต่อ)

จาวา		ดาร์ท	
30	for (int k = 0; k < i; k++) {	30	}
31	System.out.print("* ");	31	for (int k = 0; k < i; k++) {
32	}	32	t += "* ";
33	System.out.println();	33	}
34	}	34	t += "\n";
35	}	35	}
36	}	36	print(t);
37	class GraphicEditor extends Editor {	37	}
38	@Override	38	}
39	protected Shapes createShapes() {	39	class GraphicEditor extends Editor {
40	return new MyShapes();	40	Shapes _createShapes() => new MyShapes();
41	}	41	}
42	}		
43	public class Client {	42	void main() => new GraphicEditor().operation();
44	public static void main(String arg[]) {		
45	new GraphicEditor().operation();		
46	}		
47	}		

4. แพทเทิร์นต้นแบบ

จากตารางที่ 4.7 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีลักษณะในการพัฒนาที่แตกต่างกัน เนื่องจากในดาร์ทยังไม่มีเมธอดสำหรับการคัดลอกที่รันบนบรรทัดคำสั่ง จึงได้ทำการพัฒนาคลาส Cloneable ในบรรทัดที่ 19 สำหรับสร้างเมธอดในการคัดลอกอ๊อบเจกต์ ในขณะที่จาวามีการฝังเมธอดสำหรับคัดลอกอ๊อบเจกต์ไว้ในเครื่องจักรเสมือนแล้ว การพัฒนาคลาส Cloneable ของดาร์ทที่ภายในประกอบด้วยเมธอด clone() ในการพัฒนาดังกล่าวอ้างอิงจากเอกสารของภาษาจาวา โดยเป็นการคัดลอกที่มีลักษณะเป็นการคัดลอกแบบตื้น (Shallow copy) ซึ่งคลาส Cloneable เป็นคลาสที่สามารถนำไปใช้ซ้ำได้ทันทีเมื่อต้องการคัดลอกอ๊อบเจกต์

ในการใช้งานคลาส Cloneable จะใช้คุณสมบัติมิกซ์-อิน (Mixins) (Bracha, 2013) ของภาษาดาร์ท การใช้งานคุณสมบัติไมนิกทำให้คลาสที่ต้องการโอเวอร์ไรด์เมธอด clone()จะต้องมีการสืบทอดคลาสอย่างน้อยหนึ่งคลาส หากคลาสดังกล่าวไม่ได้มีการสืบทอดคลาสใด ๆ ให้คลาสนั้นทำการสืบทอดคลาส Object ดังตัวอย่างในบรรทัดที่ 3 และการเรียกใช้เมธอด clone() จะเรียกใช้

ผ่าน super และจำเป็นต้องส่งอินสแตนซ์ของคลาสที่ทำการเรียกด้วย เช่น super.clone(this) ดังตัวอย่างในบรรทัดที่ 9 ในการใช้งานของจาวาจะแตกต่างกันออกไป คือการเรียกใช้เมธอด clone() โดยเรียกใช้ผ่าน super โดยไม่ต้องส่งอินสแตนซ์ของคลาสที่ทำการเรียก เช่น super.clone()ดังตัวอย่างในบรรทัดที่ 13 และคลาสที่ทำการโอเวอร์ไรน์เมธอด `clone()` จะต้องทำการอิมพลีเมนต์คลาส Cloneable ด้วย

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นต้นแบบในภาษาจาวาใช้ 25 บรรทัด ภาษาดาร์ทใช้ 30 บรรทัด เมื่อนับบรรทัดในส่วนของคลาส Cloneable รวมด้วย หากไม่นับรวมแล้ว ดาร์ทจะใช้เพียง 18 บรรทัด

ตารางที่ 4.7 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นต้นแบบ

จาวา

```
1  package Prototype;
2  interface ShapesPrototype {
3    public void circle(int n);
4    public Circle clone()
           throws CloneNotSupportedException;
5  }
6  class Circle implements ShapesPrototype, Cloneable {
7    @Override
8    public void circle(int r) {
9      System.out.println("Area of Circle : " + (22/7) * r * r);
10   }
11   @Override
12   public Circle clone() throws
               CloneNotSupportedException {
13     return (Circle) super.clone();
14   }
15 }
16 public class Client {
17   public static void main(String arg[]) throws
               CloneNotSupportedException {
18     Circle obj1 = new Circle();
19     ShapesPrototype obj2 = obj1.clone();
20     System.out.println("obj1 = " + obj1);
```

ดาร์ท

```
1  library Prototype;
2  import 'dart:mirrors';
3  abstract class ShapesPrototype extends Object
                        with Cloneable {
4    void circle(int n);
5    Circle clone();
6  }
7  class Circle extends ShapesPrototype {
8    void circle(int r) => print("Area of Circle: ${(22/7)*r*r}");
9    Circle clone()=> super.clone(this);
10 }
11 void main() {
12   Circle obj1 = new Circle();
13   ShapesPrototype obj2 = obj1.clone();
14   print("obj1 = $obj1");
15   print("obj2 =  $obj2");
16   obj1.circle(3);
17   obj2.circle(2);
18 }
19 class Cloneable{
20   Object clone(Object myClass){
21     InstanceMirror im = reflect(myClass);
22     ClassMirror MyClassMirror = im.type;
23     InstanceMirror im2 = MyClassMirror.newInstance(
                    const Symbol(''), []);
```

ตารางที่ 4.7 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นต้นแบบ (ต่อ)

จาวา		ดาร์ท	
21	System.out.println("obj2 = " + obj2);	24	Iterable<DeclarationMirror> fileds =
22	obj1.circle(3);		MyClassMirror.declarations.values.where(
23	obj2.circle(2);		(dm) => dm is VariableMirror);
24	}	25	for(VariableMirror vm in fileds){
25	}	26	im2.setField(vm.simpleName, im.getField(
			vm.simpleName).reflectee);
		27	}
		28	return im2.reflectee;
		29	}
		30	}

5. แพทเทิร์นเดี่ยว

จากตารางที่ 4.8 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีความแตกต่างกันที่ชัดเจนมาก พิจารณาคลาส Singleton ในจาวามีการกำหนดคอนสตรัคเตอร์ 1 คอนสตรัคเตอร์ โดยกำหนดให้เป็น private เพื่อไม่ให้สามารถเข้าถึงได้จากคลาสอื่น และใช้คลาสซ้อนคลาสโดยมีการระบุคำสงวน static (static nested class) สำหรับการสร้างอินสแตนซ์ของคลาส Singleton และกำหนดให้เก็บไว้ในตัวแปร INSTANCE ที่กำหนดไว้เป็นตัวแปรแบบ private และ final และมีเมธอด getInstance() สำหรับการส่งค่าตัวแปร INSTANCE กลับเมื่อมีการเรียกใช้

ในดาร์ทคลาส Singleton มีการกำหนดคอนสตรัคเตอร์ 2 คอนสตรัคเตอร์ คือ Singleton._inner()ซึ่งเป็นคอนสตรัคเตอร์แบบระบุชื่อตามคุณสมบัติของดาร์ท และ Singleton() สำหรับ Singleton._inner() เป็นคอนสตรัคเตอร์ที่มีไว้สำหรับสร้างอ๊อบเจกต์เพื่อเก็บไว้ในตัวแปร _INSTANCE ซึ่งเป็นตัวแปรแบบ private และกำหนดให้เป็น final เช่นเดียวกับจาวา และ Singleton() ที่กำหนดให้เป็นแฟกทอรีคอนสรัคเตอร์เพื่อให้คอนสตรัคเตอร์สามารถส่งค่าจากตัวแปร _INSTANCE กลับได้

สำหรับการเรียกใช้คลาส Singleton ในจาวาการเรียกใช้อินสแตนซ์ของคลาส Singleton ทำได้โดยการเรียกผ่าน getInstance()เท่านั้น ไม่สามารถทำการสร้างอินสแตนซ์ของคลาสโดยการ new อ๊อบเจกต์ของคลาส Singleton ได้ ดังตัวอย่างในบรรทัดที่ 13 ในทางตรงกันข้ามดาร์ทสามารถเรียกใช้อินสแตนซ์ของคลาส Singleton ได้ด้วยการ new อ๊อบเจกต์ได้ ดังตัวอย่างในบรรทัดที่ 8

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นเดี่ยวในภาษาจาวาใช้ 16 บรรทัด ภาษาดาร์ทใช้เพียง 11 บรรทัด

ตารางที่ 4.8 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นเดี่ยว

จาวา	ดาร์ท

1 package Singleton;
2 class Singleton {
3 private Singleton() {}
4 private static class SingleHolder {
5 private static final Singleton INSTANCE =
new Singleton();
6 }
7 public static Singleton getInstance() {
8 return SingleHolder.INSTANCE;
9 }
10 class Client {
11 public static void main(String[] args) {
12 Singleton s1 = Singleton.getInstance();
13 Singleton s2 = Singleton.getInstance();
14 System.out.println(s1.equals(s2));
15 }
16 }
	1 library Singleton;
	2 class Singleton {
	3 Singleton._inner();
	4 static final Singleton _INSTANCE =
	new Singleton._inner();
	5 factory Singleton() => _INSTANCE;
	6 }
	7 void main() {
	8 Singleton s1 = new Singleton();
	9 Singleton s2 = new Singleton();
	10 print(identical(s1, s2));
	11 }

4.2.2 แพตเทิร์นเชิงโครงสร้าง

1. แพทเทิร์นตัวแปลง

จากตารางที่ 4.9 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีลักษณะในการพัฒนาที่แตกต่างกัน เมื่อพิจารณา Shapes ในจาวาถูกกำหนดให้เป็น interface ภายในอินเตอร์เฟซมีเพียงเมธอด draw() แต่ในดาร์ท Shapes ถูกกำหนดให้เป็น abstract class เนื่องจากในดาร์ทไม่มี interface คลาสนามธรรม Shapes ของดาร์ท จะมีการอิมพลีเมนต์คลาส 2 คลาส คือ คลาส LegacyRectangles และ LegacyCircles ซึ่งภายในของทั้ง 2 คลาส มีเมธอด draw() แต่ภายในตัวของคลาสนามธรรม Shapes ไม่ปรากฏเมธอดใด ๆ ดังตัวอย่างในบรรทัดที่ 13 เมื่อคลาส RectangleAdept และคลาส CircleAdept ได้ทำการอิมพลีเมนต์คลาสนามธรรม Shapes จะสามารถทำการโอเวอร์ไรด์เมธอด draw() ได้ แต่ในกรณีที่คลาส LegacyRectangles และคลาส

LegacyCircles มีชื่อของเมธอดภายในที่แตกต่างกันจะไม่สามารถใช้ลักษณะดังกล่าวได้ คลาสนามธรรมของดาร์ทสามารถนำไป extends ได้เช่นเดียวกับคลาสนามธรรมในจาวา และคลาสนามธรรมสามารถทำการอิมพลีเมนต์คลาสทั่วไปได้ ในขณะที่จาวาไม่สามารถทำเช่นนั้นได้

สำหรับ main()ในจาวามีการประกาศตัวแปร shapes เป็นแบบอะเรย์ ซึ่งมีแบบชนิดเป็น Shapes คือ Shape[] shapes = {new Line(), new Rectangle()}ส่วนในดาร์ทไม่มีอะเรย์แต่มีลิสต์ ซึ่งเป็นแบบชนิดที่มีอยู่ในตัว และมีคุณสมบัติเหมือนกับอะเรย์ในจาวา การใช้งานจะมีลักษณะต่างกันเล็กน้อย คือ List<Shape> shapes = [new Line(), new Rectangle()]

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวแปลงในภาษาจาวาใช้ 38 บรรทัด ภาษาดาร์ทใช้เพียง 27 บรรทัด

ตารางที่ 4.9 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแปลง

จาวา

```
1  package Adapter;
2  class LegacyLine {
3    public void draw(int x1, int y1, int x2, int y2) {
4      System.out.println("Line from (" + x1 + ' , ' + y1 + " ) to
                    (" + x2 + ' , ' + y2 + ' ) ' );
5    }
6  }
7  class LegacyRectangle {
8    public void draw(int x, int y, int w, int h) {
9      System.out.println("Rectangle at ("+x+','+ y + ") with
             width " + w + " and height " + h);
10   }
11 }
12 interface Shape {
13   void draw(int x1, int y1, int x2, int y2);
14 }
15 class Line implements Shape {
16   private LegacyLine adaptee = new LegacyLine();
17   @Override
18   public void draw(int x1, int y1, int x2, int y2) {
19     adaptee.draw(x1, y1, x2, y2);
20   }
21 }
```

ดาร์ท

```
1  library Adapter;
2  import 'dart:math' as math;
3  class LegacyLine {
4    void draw(int x1, int y1, int x2, int y2) {
5      print("Line from ($x1, $y1) to ($x2, $y2)");
6    }
7  }
8  class LegacyRectangle {
9    void draw(int x, int y, int w, int h) {
10     print("Rectangle at ($x, $y) with width $w and height
                    $h");
11   }
12 }



13 abstract class Shape implements LegacyLine,
                        LegacyRectangle {}

14 class Line implements Shape {
15   LegacyLine _adaptee = new LegacyLine();
16   void draw(int x1, int y1, int x2, int y2)
           => _adaptee.draw(x1, y1, x2, y2);
17 }
```

ตารางที่ 4.9 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแปลง (ต่อ)

จาวา	ดาร์ท

22 class Rectangle implements Shape {
23 private LegacyRectangle adaptee =
new LegacyRectangle();
24 @Override
25 public void draw(int x1, int y1, int x2, int y2) {
26 adaptee.draw(Math.min(x1, x2), Math.min(y1, y2),
Math.abs(x2- x1),Math.abs(y2 - y1));
27 }
28 }
29 public class Client {
30 public static void main(String[] args) {
31 Shape[] shapes = {new Line(), new Rectangle()};
32 int x1 = 10, y1 = 20;
33 int x2 = 30, y2 = 60;
34 for (Shape shape : shapes) {
35 shape.draw(x1, y1, x2, y2);
36 }
37 }
38 }
	18 class Rectangle implements Shape {
	19 LegacyRectangle _adaptee = new LegacyRectangle();
	20 void draw(int x1, int y1, int x2, int y2)
	=> _adaptee.draw(math.min(x1, x2), math.min(y1,
	y2), (x2 - x1).abs(), (y2 - y1).abs());
	21 }
	
	22 void main() {
	23 List<Shape> shapes = [new Line(), new Rectangle()];
	24 int x1 = 10, y1 = 20;
	25 int x2 = 30, y2 = 60;
	26 shapes.forEach((Shape shape){shape.draw
	(x1, y1, x2, y2);});
	27 }

2. แพทเทิร์นบริดจ์

จากตารางที่ 4.10 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ของทั้ง 2 ภาษา คือการเริ่มต้นเขตของข้อมูล และการใช้ฟังก์ชันลูกศร

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นบริดจ์ในภาษาจาวาใช้ 37 บรรทัด ภาษาดาร์ทใช้เพียง 24 บรรทัด

ตารางที่ 4.10 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นบริดจ์

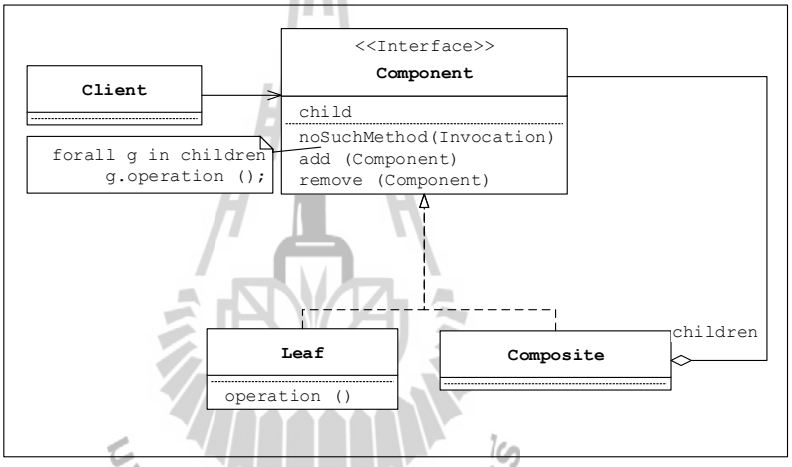
จาวา	ดาร์ท
1 package Bridge; 2 interface ShapesAbs { 3 void drawShapes();	1 library Bridge; 2 abstract class ShapesAbs { 3 void drawShapes();

ตารางที่ 4.10 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นบริดจ์ (ต่อ)

จาวา	ดาร์ท
}	}
interface GraphicImp {	abstract class GraphicImp {
void drawGraphic();	void drawGraphic();
}	}
class CircleShapes implements ShapesAbs {	class CircleShapes implements ShapesAbs {
protected GraphicImp imp;	GraphicImp _imp;
protected CircleShapes(GraphicImp imp) {	CircleShapes(this._imp);
this. imp = imp;	void drawShapes() => _imp.drawGraphic();
}	}
@Override	
public void drawShapes() {	
imp.drawGraphic();	
}	
}	
class TwoD implements GraphicImp {	class TwoD implements GraphicImp {
@Override	void drawGraphic() => print("Drawing 2D");
public void drawGraphic() {	}
System.out.println("Drawing 2D");	
}	
}	
class ThreeD implements GraphicImp {	class ThreeD implements GraphicImp {
@Override	void drawGraphic() => print("Drawing 3D");
public void drawGraphic() {	}
System.out.println("Drawing 3D");	
}	
}	
public class Client {	void main() {
public static void main(String[] args) {	List<ShapesAbs> shapes = [
ShapesAbs[] shapes = {new CircleShapes(new TwoD()), new CircleShapes(new ThreeD())};	new CircleShapes(new TwoD()), new CircleShapes(new ThreeD())];
for(ShapesAbs shape : shapes){	for(ShapesAbs shape in shapes){
shape.drawShapes();	shape.drawShapes();
}	}
}	}
}	

3. แพทเทิร์นเชิงประกอบ

แพทเทิร์นเชิงประกอบผู้วิจัยได้ทำการปรับปรุงโครงสร้างของแพทเทิร์น โดยการเปลี่ยนแปลงที่อินเตอร์เฟซ Component และคลาส Composite ดังแสดงในรูปที่ 4.2 จากโครงสร้างเดิมของแพทเทิร์นที่กำหนดให้อินเตอร์เฟซ Component มีเพียงเมธอดนามธรรมและให้คลาส Composite ทำการสืบทอดและอิมพลีเมนต์เมธอดที่ได้ทำการโอเวอร์ไรด์ได้ปรับปรุงโครงสร้างโดยกำหนดให้อินเตอร์เฟซ Component เป็นคลาสนามธรรมและได้ทำการอิมพลีเมนต์เมธอดในคลาส Component ทั้งหมด ส่วนคลาส Composite ให้ทำการสืบทอดตามโครงสร้างเดิมแต่ไม่ต้องทำการอิมพลีเมนต์



รูปที่ 4.2 โครงสร้างแพทเทิร์นเชิงประกอบที่ได้ทำการปรับปรุง

จากตารางที่ 4.11 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีลักษณะในการพัฒนาที่แตกต่างกัน เนื่องมาจากการปรับปรุงโครงสร้างของแพทเทิร์น แต่การใช้งานในเมธอด main() นั้นเหมือนกัน จากการปรับปรุงทำให้สามารถนำแพทเทิร์นเชิงประกอบไปซ้ำได้ทันที โดยการเปลี่ยนแปลงที่คลาส Leaf เท่านั้น จากโครงสร้างของแพทเทิร์นเดิม จะเห็นว่าเมธอดในคลาส Leaf จำเป็นที่จะต้องมีชื่อเดียวกันกับเมธอดที่ใช้ดำเนินการในคลาส Component ซึ่งก็คือเมธอด Operation() แต่ด้วยคุณสมบัติของภาษาดาร์ทคลาส Leaf สามารถเปลี่ยนชื่อของเมธอดสำหรับดำเนินการได้ตามลักษณะของงานที่ต้องการใช้ได้ทันทีโดยที่ไม่ต้องทำการอิมพลีเมนต์เพิ่มเติมหรือเปลี่ยนแปลงคลาส Component

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งาน

แพทเทิร์นเชิงประกอบในภาษาจาวาใช้ 45 บรรทัด ภาษาดาร์ทใช้เพียง 33 บรรทัด

ตารางที่ 4.11 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นเชิงประกอบ

จาวา

```
1  package Composite;
2  import java.util.ArrayList;
3  import java.util.List;
4  interface Component {
5    public void operation();
6  }
7  class Composite implements Component {
8    private List<Component> childGraphics =
                        new ArrayList<>();
9    @Override
10   public void operation() {
11     for (Component graphic : childGraphics) {
12       graphic.operation();
13     }
14   }
15   public void add(Component graphic) {
16     childGraphics.add(graphic);
17   }
18   public void remove(Component graphic) {
19     childGraphics.remove(graphic);
20   }
21 }
22 class Leaf implements Component {
23   @Override
24   public void operation() {
25     System.out.println("Leaf");
26   }
27 }
28 public class Client {
29   public static void main(String[] args) {
30     Leaf l1 = new Leaf();
31     Leaf l2 = new Leaf();
32     Leaf l3 = new Leaf();
33     Leaf l4 = new Leaf();
```

ดาร์ท

```
1  library Composite;
2  import 'dart:mirrors';
3  abstract class Component{
4    List<Component> _child = new List();
5    noSuchMethod(Invocation iv) {
6      for(Component c in _child){
7        reflect(c).invoke( iv.memberName,
                    iv.positionalArguments,
                    iv.namedArguments
8        );
9      }
10   }
11   add(Component c) => _child.add(c);
12   remove(Component c) => _child.remove(c);
13 }
14 class Composite extends Component{}
15 class Leaf extends Component{
16   void operation()=> print("Leaf");
17 }
18 void main() {
19   Leaf l1 = new Leaf();
20   Leaf l2 = new Leaf();
21   Leaf l3 = new Leaf();
22   Leaf l4 = new Leaf();
23   Composite c1  = new Composite();
```

ตารางที่ 4.11 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นเชิงประกอบ (ต่อ)

จาวา		ดาร์ท	
34	Composite c1 = new Composite();	24	Composite c2 = new Composite();
35	Composite c2 = new Composite();	25	Composite c3 = new Composite();
36	Composite c3 = new Composite();	26	c2.add(l1);
37	c2.add(l1);	27	c2.add(l2);
38	c2.add(l2);	28	c2.add(l3);
39	c2.add(l3);	29	c3.add(l4);
40	c3.add(l4);	30	c1.add(c2);
41	c1.add(c2);	31	c1.add(c3);
42	c1.add(c3);	32	c1.operation();
43	c1.operation();	33	}
44	}		
45	}		

4. แพทเทิร์นตัวตกแต่ง

จากตารางที่ 4.12 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ของทั้ง 2 ภาษา คือ การเรียกใช้ super ในคอนสตรัคเตอร์ของคลาสลูก ตัวอย่างการเรียกใช้ super ในจาวา

```
class BorderDecorator extends Decorator {
  public BorderDecorator(Widget w) {
    super(w);
  }
  /* Source code*/
}
```

ตัวอย่างการเรียกใช้ super ในดาร์ท

```
class BorderDecorator extends Decorator {
  BorderDecorator(Widget w) : super(w);
  /* Source code*/
}
```

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวตกแต่งในภาษาจาวาใช้ 53 บรรทัด ภาษาดาร์ทใช้เพียง 34 บรรทัด

ตารางที่ 4.12 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวตกแต่ง

จาวา

```
1  package Decorator;
2  interface Widget {
3    void draw();
4  }
5  class TextField implements Widget {
6    private int width, height;
7    public TextField(int width, int height) {
8      this.width = width;
9      this.height = height;
10   }
11   @Override
12   public void draw() {
13     System.out.println("Text Field size " + width +
                    " * " + height);
14   }
15 }
16 abstract class Decorator implements Widget {
17   private Widget w;
18   public Decorator(Widget w) {
19     this.w = w;
20   }
21   @Override
22   public void draw() {
23     w.draw();
24   }
25 }
26 class BorderDecorator extends Decorator {
27   public BorderDecorator(Widget w) {
28     super(w);
29   }
30   @Override
31   public void draw() {
32     super.draw();
```

ดาร์ท

```
1  library Decorator;
2  abstract class Widget {
3    void draw();
4  }
5  class TextField implements Widget {
6    int _width, _height;
7    TextField(this._width, this._height);
8    void draw() =>print("Text Field size ${this._width}
                          * ${this._height}");
9  }
10 abstract class Decorator implements Widget {
11   Widget _w;
12   Decorator(this._w);
13   void draw() =>  _w.draw();
14 }
15 class BorderDecorator extends Decorator {
16   BorderDecorator(Widget w) : super(w);
17   void draw() {
18     super.draw();
19     print("  Border Decorator");
20   }
21 }
```

ตารางที่ 4.12 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวตกแต่ง (ต่อ)

จาวา	ดาร์ท

33 System.out.println(" Border Decorator");
34 }
35 }
36 class ScrollDecorator extends Decorator {
37 public ScrollDecorator(Widget w) {
38 super(w);
39 }
40 @Override
41 public void draw() {
42 super.draw();
43 System.out.println(" Scroll Decorator");
44 }
45 }
46 public class Client {
47 public static void main(String[] args) {
48 Widget w = new BorderDecorator(
49 new ScrollDecorator(
50 new TextField(60, 20)));
51 w.draw();
52 }
53 }
	22 class ScrollDecorator extends Decorator {
	23 ScrollDecorator(Widget w) : super(w);
	24 void draw() {
	25 super.draw();
	26 print(" Scroll Decorator");
	27 }
	28 }
	29 void main() {
	30 Widget w = new BorderDecorator(
	31 new ScrollDecorator(
	32 new TextField(60, 20)));
	33 w.draw();
	34 }

5. แพทเทิร์นส่วนหน้า

จากตารางที่ 4.13 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ทมีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ที่พบในแพทเทิร์นนี้ คือการแปลงสตริงให้เป็นตัวเลขจำนวนจริง ในจาวาการแปลงสตริงให้เป็นจำนวนจริงจะต้องเขียนดังนี้ Double.parseDouble(“123”) ส่วนในดาร์ทการแปลงสตริงให้เป็นจำนวนจริงเขียนเพียง double.parse(“123”)และการจัดรูปแบบของตัวเลขในจาวาจะต้องทำการสร้างอินสแตนซ์ของคลาส DecimalFormat แล้วทำการระบุรูปแบบที่ต้องการ เช่น new DecimalFormat(“0.00”) ในขณะที่ดาร์ทต้องทำการสร้างอินสแตนซ์ของคลาส NumberFormat แล้วทำการระบุรูปแบบที่ต้องการ และสามารถระบุสถานที่เฉพาะ (Locale-specific) ได้ เช่น new NumberFormat(“0.00”, “en_US”)

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งาน

แพทเทิร์นส่วนหน้าในภาษาจาวาใช้ 94 บรรทัด ภาษาดาร์ทใช้เพียง 64 บรรทัด

ตารางที่ 4.13 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นส่วนหน้า

จาวา

```
1  package Facade;
2  import java.text.DecimalFormat;
3  class PointCarte {
4    DecimalFormat d = new DecimalFormat("0.00");
5    private double x, y;
6    public PointCarte(double x, double y) {
7      this.x = x;
8      this.y = y;
9    }
10   public void move(int mx, int my) {
11     this.x += mx;
12     this.y += my;
13   }
14   @Override
15   public String toString() {
16     return "(" + d.format(this.x) + "," +d.format(this.y) + ")";
17   }
18   public double getX() {
19     return this.x;
20   }
21   public double getY() {
22     return this.y;
23   }
24 }
25 class PointPolar {
26   DecimalFormat d = new DecimalFormat("0.00");
27   private double radius, angle;
28   public PointPolar(double r, double a) {
29     this.radius = r;
30     this.angle = a;
31   }
32   public void rotate(int ang) {
33     this.angle += ang % 360;
34   }
```

ดาร์ท

```
1  library Facade;
2  import 'dart:math' as math;
3  import 'package:intl/intl.dart';
4  class PointCarte {
5    NumberFormat d = new NumberFormat("0.00", "en_US");
6    double _x, _y;
7    PointCarte(this._x, this._y);
8    void move(int mx, int my) {
9      this._x += mx;
10     this._y += my;
11   }
12   String toString() => "(${d.format(this._x)},
                          ${d.format(this._y)})";
13   double get X => this._x;
14   double get Y => this._y;
15 }
16 class PointPolar {
17   NumberFormat d = new NumberFormat("0.00", "en_US");
18   double _radius, _angle;
19   PointPolar(this._radius, this._angle);
20   void rotate(int ang){
21     this._angle += ang % 360;
22   }
23   String toString() => "[${d.format(this._radius)}
                          @${d.format(this._angle)}]";
24 }
```

ตารางที่ 4.13 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นส่วนหน้า (ต่อ)

จาวา		ดาร์ท	
35	@Override		
36	public String toString() {		
37	return "[" + d.format(this.radius) + "@" + d.format(this.angle)+"]";		
38	}		
39	}		
40	class Point {	25	class Point {
41	private PointCarte pc;	26	PointCarte _pc;
42	public Point(double x, double y) {	27	Point(double x, double y) {
43	this.pc = new PointCarte(x, y);	28	this._pc = new PointCarte(x, y);
44	}	29	}
45	@Override	30	String toString() => this._pc.toString();
46	public String toString() {	31	void move(int mx, int my) => this._pc.move(mx, my);
47	return this.pc.toString();	32	void rotate(int angle, Point o) {
48	}	33	double x = this._pc.X - o._pc.X;
49	public void move(int mx, int my) {	34	double y = this._pc.Y - o._pc.Y;
50	this.pc.move(mx, my);	35	PointPolar pp = new PointPolar(math.sqrt(x * x + y * y), math.atan2(y, x) * 180 / math.PI);
51	}	36	pp.rotate(angle);
52	public void rotate(int angle, Point o) {	37	print(" PointPolar is $pp");
53	double x = this.pc.getX() - o.pc.getX();	38	String str = pp.toString();
54	double y = this.pc.getY() - o.pc.getY();	39	int i = str.indexOf('@');
55	PointPolar pp = new PointPolar(Math.sqrt(x * x + y * y), Math.atan2(y, x) * 180 / Math.PI);	40	double r = double.parse(str.substring(1, i));
56	pp.rotate(angle);	41	double a = double.parse(str.substring(i+ 1, str.length - 1));
57	System.out.println(" PointPolar is "+pp);	42	this._pc = new PointCarte(r * math.cos(a * math.PI/180) + o._pc.X, r *math.sin(a*math.PI/180) +o._pc.Y);
58	String str = pp.toString();	43	}
59	int i = str.indexOf('@');	44	}
60	double r = Double.parseDouble(str.substring(1, i));		
61	double a = Double.parseDouble(str.substring(i+1,str.length()-1));		
62	this.pc = new PointCarte(r*Math.cos(a * Math.PI/180) + o.pc.getX(), r*Math.sin(a*Math.PI/180)+o.pc.getY());		
63	}		
64	}		
65	class Line {	45	class Line {
66	private Point ori, end;	46	Point _ori, _end;

ตารางที่ 4.13 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นส่วนหน้า (ต่อ)

จาวา	ดาร์ท
67 public Line(Point ori, Point end) {	47 Line(this._ori, this._end);
68 this.ori = ori;	48 void move(int mx, int my) {
69 this.end = end;	49 this._ori.move(mx, my);
70 }	50 this._end.move(mx, my);
71 public void move(int mx, int my) {	51 }
72 this.ori.move(mx, my);	52 void rotate(int angle)
73 this.end.move(mx, my);	=> this._end.rotate(angle, this._ori);
74 }	53 String toString() => "origin is ${this._ori},
75 public void rotate(int angle) {	end is ${this._end}";
76 this.end.rotate(angle, this.ori);	54 }
77 }	
78 @Override	
79 public String toString() {	
80 return "origin is " + this.ori + ", end is " + this.end;	
81 }	
82 }	
83 class Client {	
84 public static void main(String[] args) {	55 void main() {
85 Line line1 = new Line(new Point(2, 4), new Point(5, 7));	56 Line line1 = new Line(new Point(2.0, 4.0),
86 line1.move(-2, -4);	new Point(5.0, 7.0));
87 System.out.println("After move: "+ line1);	57 line1.move(-2, -4);
88 line1.rotate(45);	58 print("After move: $line1");
89 System.out.println("After rotate:"+line1);	59 line1.rotate(45);
90 Line line2 = new Line(new Point(2, 1),	60 print("After rotate: $line1");
new Point(2.5, 1.5));	61 Line line2 = new Line(new Point(2.0, 1.0),
91 line2.rotate(30);	new Point(2.5, 1.5));
92 System.out.println("30 degrees to 60 degrees: " + line2);	62 line2.rotate(30);
93 }	63 print("30 degrees to 60 degrees: $line2");
94 }	64 }

6. แพทเทิร์นน้ำหนักเบา

จากตารางที่ 4.14 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ยกเว้นในส่วนของคลาส FlyweightFactory ได้พัฒนาให้สามารถนำไปใช้ซ้ำได้ แต่ยังจำเป็นต้องเปลี่ยนในส่วนของคลาสที่นำมาสร้างอินสแตนซ์สำหรับ

การดำเนินการ จากตัวอย่างในบรรทัดที่ 22 มีการสร้างอินสแตนซ์ของคลาส CoffeeFlavor การนำไปใช้จะต้องเปลี่ยนในส่วนของชื่อคลาสนี้ไปเป็นชื่อคลาสที่ต้องการ

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นน้ำหนักเบาในภาษาจาวาใช้ 64 บรรทัด ภาษาดาร์ทใช้เพียง 45 บรรทัด

ตารางที่ 4.14 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นน้ำหนักเบา

จาวา

```
1  package Flyweight;
2  import java.util.HashMap;
3  import java.util.Map;
4  interface CoffeeOrder{
5   public void serveCoffee(CoffeeOrderContext context);
6  }
7  class CoffeeFlavor implements CoffeeOrder{
8   private final String flavor;
9   public CoffeeFlavor(String newFlavor){
10    this.flavor = newFlavor;
11  }
12  public String getFlavor(){
13    return this.flavor;
14  }
15  @Override
16  public void serveCoffee(CoffeeOrderContext context){
17   System.out.println("Serving coffee flavor "+ flavor +"
                          to table number "+context.getTable());
18  }
19 }
20 class CoffeeOrderContext{
21   private final int tableNumber;
22   public CoffeeOrderContext(int tableNumber){
23     this.tableNumber = tableNumber;
24   }
25   public int getTable(){
26     return this.tableNumber;
27   }
28 }
29 class FlyweightFactory{
```

ดาร์ท

```
1  library Flyweight;
2  import "dart:collection";

3  abstract class CoffeeOrder{
4   void serveCoffee(CoffeeOrderContext context);
5  }
6  class CoffeeFlavor implements CoffeeOrder{
7   final String _flavor;
8   CoffeeFlavor(this._flavor);
9   String get Flavor => this._flavor;
10  void serveCoffee(CoffeeOrderContext context)
        => print("Serving coffee flavor ${_flavor} to table
                  number ${context.Table}");
11 }




12 class CoffeeOrderContext{
13   final int _tableNumber;
14   CoffeeOrderContext(this._tableNumber);
15   int get Table => this._tableNumber;
16 }




17 class FlyweightFactory{
```

ตารางที่ 4.14 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นน้ำหนักเบา (ต่อ)

จาวา

```
30  private Map<String, CoffeeFlavor> flavors =
31                      new HashMap<>();
32  public CoffeeFlavor getCofFv(String flavorName){
33    CoffeeFlavor flavor = flavors.get(flavorName);
34    if(flavor == null){
35      flavor = new CoffeeFlavor(flavorName);
36      flavors.put(flavorName, flavor);
37    }
38    return flavor;
39  }
40  public int getTotalCoffeeFlavorsMade(){
41    return flavors.size();
42  }
43 }
44 public class Client {
45  private static CoffeeFlavor[] flavors =
                    new CoffeeFlavor[4000];
46  private static CoffeeOrderContext[] tables =
                new CoffeeOrderContext[4000];
47  private static int ordersMade = 0;
48  private static FlyweightFactory flavorFactory
                      = new FlyweightFactory();
49  public static void takeOrders(String flavorIn, int table){
50    flavors[ordersMade] =
                flavorFactory.getCofFv(flavorIn);
51    tables[ordersMade++] =
                new CoffeeOrderContext(table);
52  }
53  public static void main(String[] args) {
54    takeOrders("Cappuccino", 2);
55    takeOrders("Cappuccino", 2);
56    takeOrders("Espresso", 97);
57    takeOrders("Espresso", 88);
58    takeOrders("Moccaccino", 121);
59    for(int i = 0 ; i < ordersMade ; ++i){
60      flavors[i].serveCoffee(tables[i]);
```

ดาร์ท

```
18  Map<String, Object> _m = new HashMap();
19  noSuchMethod(Invocation iv){
20    Object o = _m[iv.positionalArguments.first];
21    if(o == null){
    // Change name class CoffeeFlavor
22      o = new CoffeeFlavor(
                  iv.positionalArguments.first);
23      _m.putIfAbsent(
            iv.positionalArguments.first, () => o);
24    }
25    return o;
26  }
27 }

28 void main() {
29  List<CoffeeFlavor> _flavors = new List();
30  List<CoffeeOrderContext> _tables = new List();
31  FlyweightFactory _flavorFactory =
                    new FlyweightFactory();
32  void takeOrders(String flavorIn, int table){
33    _flavors.add(_flavorFactory.getCofFv(flavorIn));
34    _tables.add(new CoffeeOrderContext(table));
35  }
36  takeOrders("Cappuccino", 2);
37  takeOrders("Cappuccino", 2);
38  takeOrders("Espresso", 97);
39  takeOrders("Espresso", 88);
40  takeOrders("Moccaccino", 121);
41  for(int i = 0 ; i < _flavors.length ; ++i){
42    _flavors[i].serveCoffee(_tables[i]);
43  }
44  print("\nTotal CoffeeFlavor objects made:
            ${_flavorFactory._m.length}");
45 }
```

ตารางที่ 4.14 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นน้ำหนักเบา (ต่อ)

จาวา		ดาร์ท	
61	} System.out.println("\nTotal CoffeeFlavor objects made: " + flavorFactory. getTotalCoffeeFlavorsMade()); }		
62			
63			
64	}		

7. แพทเทิร์นตัวแทน

จากตารางที่ 4.15 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ที่พบในแพทเทิร์นนี้ ได้กล่าวไปแล้วในแต่ละแพทเทิร์นที่ผ่านมา สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวแทนในภาษาจาวาใช้ 43 บรรทัด ภาษาดาร์ทใช้เพียง 33 บรรทัด

ตารางที่ 4.15 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแทน

จาวา		ดาร์ท	
1	package Proxy;	1	library Proxy;
2	interface Image {	2	abstract class Image {
3	public void display();	3	void display();
4	}	4	}
5	class RealImage implements Image {	5	class RealImage implements Image {
6	private String filename = null;	6	String _filename = null;
7	public RealImage(final String filename) {	7	RealImage(final String filename) {
8	this.filename = filename;	8	this._filename = filename;
9	loadImageFromDisk();	9	_loadImageFromDisk();
10	}	10	}
11	private void loadImageFromDisk() {	11	void _loadImageFromDisk()
12	System.out.println("Loading " + this.filename);		=> print("Loading ${this._filename}");
13	}	12	void display()
14	@Override		=> print("Displaying ${this._filename}");
15	public void display() {	13	}
16	System.out.println("Displaying "+ this.filename);		
17	}		
18	}		
19	class ProxyImage implements Image {	14	class ProxyImage implements Image {

ตารางที่ 4.15 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแทน (ต่อ)

จาวา	ดาร์ท

20 private RealImage image = null;
21 private String filename = null;
22 public ProxyImage(final String filename) {
23 this.filename = filename;
24 }
25 @Override
26 public void display() {
27 if (this.image == null) {
28 this.image = new RealImage(this.filename);
29 }
30 this.image.display();
31 }
32 }
33 public class Client {
34 public static void main(String[] args) {
35 final Image im1 = new ProxyImage("
HiRes_10MB_Photo1");
36 final Image im2 = new ProxyImage("
HiRes_10MB_Photo2");
37 im1.display();
38 im1.display();
39 System.out.println();
40 im2.display();
41 im2.display();
42 }
43 }
	15 RealImage _image = null;
	16 String _filename = null;
	17 ProxyImage(final this._filename);
	18 void display() {
	19 if (this._image == null) {
	20 this._image = new RealImage(_filename);
	21 }
	22 this._image.display();
	23 }
	24 }
	
	25 void main() {
	26 final Image im1 = new ProxyImage("
	HiRes_10MB_Photo1");
	27 final Image im2 = new ProxyImage("
	HiRes_10MB_Photo2");
	28 im1.display();
	29 im1.display();
	30 print("");
	31 im2.display();
	32 im2.display();
	33 }

4.2.3 แพตเทิร์นเชิงพฤติกรรม

1. แพทเทิร์นห่วงโซ่ความรับผิดชอบ

จากตารางที่ 4.16 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน มีวากยสัมพันธ์ที่แตกต่างกันในแพทเทิร์นนี้ ได้กล่าวไปแล้วในแต่ละแพทเทิร์นที่ผ่านมา โดยมีการพัฒนาในส่วนของคลาส Handler ให้สามารถนำไปใช้ซ้ำได้ทั้งในจาวา และดาร์ท สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งาน

แพทเทิร์นห่วงโซ่ความรับผิดชอบในภาษาจาวาใช้ 70 บรรทัด ภาษาดาร์ทใช้เพียง 60 บรรทัด

ตารางที่ 4.16 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นห่วงโซ่ความรับผิดชอบ

จาวา

```
1  package ChainOfResponsibility;
2  class Handler {
3    private static java.util.Random s_rn =
                    new java.util.Random();
4    private static int s_next = 1;
5    private int m_id = s_next++;
6    private Handler m_next;
7    public void add(Handler next) {
8      if (this.m_next == null) {
9        this.m_next = next;
10     } else {
11       this.m_next.add(next);
12     }
13   }
14   public void wrap_around(Handler root) {
15     if (this.m_next == null) {
16       this.m_next = root;
17     } else {
18       this.m_next.wrap_around(root);
19     }
20   }
21   public void handle(Request num) {
22     System.out.print(this.m_id + ": busy");
23     this.m_next.handle(num);
24   }
25 }
26 class ConcreteHandlerOne extends Handler {
27   @Override
28   public void handle(Request request) {
29     if (request.getValue() < 0) {
30       System.out.println("- Negative values are handled by
             ConcreteHandlerOne:" + request.getDescription()
           + request.getValue());
31     } else {
```

ดาร์ท

```
1  library ChainOfResponsibility;
2  import 'dart:math' show Random;
3  class Handler {
4    static Random _s_rn = new Random();
5    static int _s_next = 1;
6    int _m_id = _s_next++;
7    Handler _m_next;
8    void add(Handler next) {
9      if (this._m_next == null) {
10       this._m_next = next;
11     } else {
12       this._m_next.add(next);
13     }
14   }
15   void wrap_around(Handler root) {
16     if (this._m_next == null) {
17       this._m_next = root;
18     } else {
19       this._m_next.wrap_around(root);
20     }
21   }
22   void handle(Request num) {
23     print("${this._m_id}: busy");
24     this._m_next.handle(num);
25   }
26 }
27 class ConcreteHandlerOne extends Handler {
28   void handle(Request request) {
29     if (request.Value < 0) {
30       print("- Negative values are handled by
             ConcreteHandlerOne: ${request.Description}
             ${request.Value}");
31     } else {
32       super.handle(request);
```

ตารางที่ 4.16 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นห่วงโซ่ความรับผิดชอบ (ต่อ)

จาวา		ดาร์ท	
32	super.handle(request);	33	}
33	}	34	}
34	}	35	}
35	}		
36	class ConcreteHandlerTwo extends Handler {	36	class ConcreteHandlerTwo extends Handler {
37	@Override	37	void handle(Request request) {
38	public void handle(Request request) {	38	if (request.Value > 0) {
39	if (request.getValue() > 0) {	39	print("+ Positive values are handled by ConcreteHandlerTwo: ${request.Description} ${request.Value}");
40	System.out.println("+ Positive values are handled by ConcreteHandlerTwo:" + request.getDescription() + request.getValue());	40	} else {
41	} else {	41	super.handle(request);
42	super.handle(request);	42	}
43	}	43	}
44	}	44	}
45	}		
46	class Request {	45	class Request{
47	private int m_value;	46	int _m_value;
48	private String m_description;	47	String _m_description;
49	public Request(String description, int value) {	48	Request(this._m_description, this._m_value);
50	this.m_description = description;	49	int get Value => this._m_value;
51	this.m_value = value;	50	String get Description => this._m_description;
52	}	51	}
53	public int getValue() {		
54	return this.m_value;		
55	}		
56	public String getDescription() {		
57	return this.m_description;		
58	}		
59	}		
60	class Client {	52	void main() {
61	public static void main(String[] args) {	53	Handler chainRoot = new Handler();
62	Handler chainRoot = new Handler();	54	chainRoot.add(new ConcreteHandlerOne());
63	chainRoot.add(new ConcreteHandlerOne());	55	chainRoot.add(new ConcreteHandlerTwo());
64	chainRoot.add(new ConcreteHandlerTwo());	56	chainRoot.wrap_around(chainRoot);
65	chainRoot.wrap_around(chainRoot);	57	chainRoot.handle(new Request("Negative Value ", -1));

ตารางที่ 4.16 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นห่วงโซ่ความรับผิดชอบ (ต่อ)

จาวา		ดาร์ท	
66	chainRoot.handle(new Request("Negative Value ", -1));	58	chainRoot.handle(new Request("Positive Value ", 1));
67	chainRoot.handle(new Request("Positive Value ", 1));	59	chainRoot.handle(new Request("Positive Value ", 2));
68	chainRoot.handle(new Request("Positive Value ", 2));	60	}
69	}		
70	}		

2. แพทเทิร์นคำสั่ง

จากตารางที่ 4.17 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ที่พบในแพทเทิร์นนี้ ได้กล่าวไปแล้วในแต่ละแพทเทิร์นที่ผ่านมา สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นคำสั่งในภาษาจาวาใช้ 55 บรรทัด ภาษาดาร์ทใช้เพียง 38 บรรทัด

ตารางที่ 4.17 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นคำสั่ง

จาวา		ดาร์ท	
1	package Command;	1	library Command;
2	import java.util.ArrayList;		
3	import java.util.Iterator;		
4	import java.util.List;		
5	interface Order {	2	abstract class Order {
6	public void execute();	3	void execute();
7	}	4	}
8	class Agent {	5	class Agent {
9	private List<Order> ordersQueue = new ArrayList();	6	List<Order> _ordersQueue = new List();
10	void placeOrder(Order[] order) {	7	void placeOrder(List<Order> order) {
11	for (int i = 0; i < order.length; i++) {	8	this._ordersQueue.addAll(order);
12	this.ordersQueue.add(order[i]);	9	for (Iterator it = _ordersQueue.iterator; it.moveNext();) {
13	}	10	it.current.execute();
14	for (Iterator it = ordersQueue.iterator(); it.hasNext();) {	11	}
15	((Order) it.next()).execute();	12	}
16	}	13	}
17	}		
18	}		
19	class BuyProductOrder implements Order {	14	class BuyProductOrder implements Order {

ตารางที่ 4.17 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นคำสั่ง (ต่อ)

จาวา	ดาร์ท

20 private ProductTrade product;
21 public BuyProductOrder(ProductTrade product){
22 this.product = product;
23 }
24 @Override
25 public void execute() {
26 this.product.buy();
27 }
28 }
29 class SellProductOrder implements Order {
30 private ProductTrade product;
31 public SellProductOrder(ProductTrade product) {
32 this.product = product;
33 }
34 @Override
35 public void execute() {
36 this.product.sell();
37 }
38 }
39 class ProductTrade {
40 public void buy() {
41 System.out.println("Buy products");
42 }
43 public void sell() {
44 System.out.println("Sell products ");
45 }
46 }
47 public class Client {
48 public static void main(String[] args) {
49 ProductTrade product = new ProductTrade();
50 Order[] order = {new BuyProductOrder(product),
new SellProductOrder(product)};
51 Agent agent = new Agent();
52 agent.placeOrder(order);
53 }
54 }
	15 ProductTrade _product;
	16 BuyProductOrder(this._product);
	17 void execute() {
	18 this._product.buy();
	19 }
	20 }
	21
	
	22 class SellProductOrder implements Order {
	23 ProductTrade _product;
	24 SellProductOrder(this._product);
	25 void execute() {
	26 this._product.sell();
	27 }
	28 }
	
	29 class ProductTrade {
	30 void buy() => print("Buy products");
	31 void sell() => print("Sell products ");
	32 }
	
	33 void main() {
	34 ProductTrade product = new ProductTrade();
	35 List<Order> order =[new BuyProductOrder(product),
	new SellProductOrder(product)];
	36 Agent agent = new Agent();
	37 agent.placeOrder(order);
	38 }

3. แพทเทิร์นตัวแปล

จากตารางที่ 4.18 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่แตกต่างคือวากยสัมพันธ์ที่พบในแพทเทิร์นนี้ คือ การใช้เงื่อนไข switch-case ในจาวาการกำหนดให้เงื่อนไขใน case ให้สิ้นสุดการทำงานด้วยคำสั่ง break โดยเขียนไว้ในวงเล็บของ case ได้ดังนี้

```
switch(x){
   case “+” : { /* Source code */
            break;
           }
}
```

ในขณะที่ดาร์ทการกำหนดให้เงื่อนไขใน case ให้สิ้นสุดการทำงานด้วยคำสั่ง break จำต้องกำหนด break ไว้นอกวงเล็บของเงื่อนไขของ case ดังนี้

```
switch(x){
   case “+” : { /* Source code */
           }
           break;
}
```

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวแปลในภาษาจาวาใช้ 98 บรรทัด ภาษาดาร์ทใช้เพียง 71 บรรทัด

ตารางที่ 4.18 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแปล

จาวา		ดาร์ท	
1	package Interpreter;	1	library Interpreter;
2	import java.util.HashMap;	2	import "dart:collection";
3	import java.util.Map;		
4	import java.util.Stack;		
5	interface Expression {	3	abstract class Expression {
6	public int interpret(Map<String, Expression> vrb);	4	int interpret(Map<String, Expression> vrb);
7	}	5	}
8	class Number implements Expression {	6	class Number implements Expression {
9	private int number;	7	int _number;

ตารางที่ 4.18 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแปล (ต่อ)

จาวา	ดาร์ท

10 public Number(int number) {
11 this.number = number;
12 }
13 @Override
14 public int interpret(Map<String, Expression> vrb) {
15 return this.number;
16 }
17 }
	8 Number(this._number);
	9 int interpret(Map<String, Expression> vrb)
	=> this._number;
	10 }

18 class Plus implements Expression {
19 Expression lOperand, rOperand;
20 public Plus(Expression left, Expression right) {
21 this.lOperand = left;
22 this.rOperand = right;
23 }
24 @Override
25 public int interpret(Map<String, Expression> vrb) {
26 return this.lOperand.interpret(vrb) +
this.rOperand.interpret(vrb);
27 }
28 }
	11 class Plus implements Expression {
	12 Expression lOperand, rOperand;
	13 Plus(this.lOperand, this.rOperand;
	14 int interpret(Map<String, Expression> vrb)
	=> this.lOperand.interpret(vrb) +
	this.rOperand.interpret(vrb);
	15 }

29 class Minus implements Expression {
30 Expression lOperand, rOperand;
31 public Minus(Expression left, Expression right) {
32 this.lOperand = left;
33 this.rOperand = right;
34 }
35 @Override
36 public int interpret(Map<String, Expression> vrb) {
37 return this.lOperand.interpret(vrb) -
this.rOperand.interpret(vrb);
38 }
39 }
	16 class Minus implements Expression {
	17 Expression lOperand, rOperand;
	18 Minus(this.lOperand, this.rOperand);
	19 int interpret(Map<String, Expression> vrb)
	=> this.lOperand.interpret(vrb) -
	this.rOperand.interpret(vrb);
	20 }

40 class Variable implements Expression {
41 private String name;
42 public Variable(String name) {
43 this.name = name;
	21 class Variable implements Expression {
	22 String _name;
	23 Variable(this._name);
	24 int interpret(Map<String, Expression> vrb) {

ตารางที่ 4.18 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแปล (ต่อ)

จาวา

```
44  }
45  @Override
46  public int interpret(Map<String, Expression> vrb) {
47    if (null == vrb.get(name)) {
48      return 0;
49    }
50    return vrb.get(name).interpret(vrb);
51  }
52 }
53 class Evaluator implements Expression {
54  private Expression syntaxTree;
55  public Evaluator(String expression) {
56    Stack<Expression> expressStack = new Stack<>();
57    for (String token :expression.split(" ")){
58      switch (token) {
59        case "+":
60          {
61          Expression subExpress = new Plus(
                    expressStack.pop(),
                    expressStack.pop());
62          expressStack.push(subExpress);
63          break;
64          }
65        case "-":
66          {
67          Expression right = expressStack.pop();
68          Expression left = expressStack.pop();
69          Expression subExpress =  new Minus(left, right);
70          expressStack.push(subExpress);
71          break;
72              }
73        default:
74          expressStack.push(new Variable(token));
75          break;
76      }
77    }
```

ดาร์ท

```
25    if (null == vrb[_name]) {
26      return 0;
27    }
28    return vrb[_name].interpret(vrb);
29  }
30 }
31 class Evaluator implements Expression {
32  Expression _syntaxTree;
33  Evaluator(String expression) {
34   List<Expression> expressStack = new List();
35   for(String token in expression.split(" ")){
36     switch (token) {
37       case "+":
38         {
39         Expression subExpress = new Plus(
                  expressStack.removeLast(),
                  expressStack.removeLast());
40         expressStack.add(subExpress);
41         }
42         break;
43       case "-":
44          {
45          Expression right = expressStack.removeLast();
46          Expression left =  expressStack.removeLast();
47          Expression subExpress =  new Minus(left, right);
48          expressStack.add(subExpress);
49          }
50          break;
51       default:
52          expressStack.add(new Variable(token));
53          break;
54     }
55   }
```

ตารางที่ 4.18 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวแปล (ต่อ)

จาวา	ดาร์ท

78 this.syntaxTree = expressStack.pop();
79 }
80 @Override
81 public int interpret(Map<String, Expression> context) {
82 return syntaxTree.interpret(context);
83 }
84 }
85 class Client {
86 public static void main(String[] args) {
87 String expression = "x y z - +";
88 Evaluator sentence = new Evaluator(expression);
89 Map<String, Expression> vrb = new HashMap<>();
90 String s1 = "x", s2 = "y", s3 = "z";
91 int n1 = 5, n2 = 10, n3 = 42;
92 vrb.put(s1, new Number(n1));
93 vrb.put(s2, new Number(n2));
94 vrb.put(s3, new Number(n3));
95 int result = sentence.interpret(vrb);
96 System.out.println(s1 + ":" + n1 + ", " + s2 + ":" + n2 +
", " + s3 + ":"+ n3 +"\n" + expression + " = " + result);
97 }
98 }
	56 this._syntaxTree = expressStack.removeLast();
	57 }
	58 int interpret(Map<String, Expression> context) =>
	this._syntaxTree.interpret(context);
	59 }
	
	60 void main() {
	61 String expression = "x y z - +";
	62 Evaluator sentence = new Evaluator(expression);
	63 Map<String, Expression> vrb = new HashMap();
	64 String s1 = "x", s2 = "y", s3 = "z";
	65 int n1 = 5, n2 = 10, n3 = 42;
	66 vrb.putIfAbsent(s1, ()=>new Number(n1));
	67 vrb.putIfAbsent(s2, ()=>new Number(n2));
	68 vrb.putIfAbsent(s3, ()=>new Number(n3));
	69 int result = sentence.interpret(vrb);
	70 print("$s1: $n1, $s2: $n2, $s3: $n3
	\n$expression = $result");
	71 }

4. แพทเทิร์นตัววนซ้ำ

จากตารางที่ 4.19 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ยกเว้นในส่วนของคลาส Iterators ในดาร์ท ได้พัฒนาให้สามารถนำคลาสไปใช้ซ้ำได้ แต่การใช้งานจำเป็นต้องส่งสตริงที่เป็นชื่อของเมธอดที่ต้องการเข้าถึงข้อมูล (เกทเทอร์) ให้กลับคลาส Iterators ด้วย ดังตัวอย่างในบรรทัดที่ 52

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัววนซ้ำในภาษาจาวาใช้ 62 บรรทัด ภาษาดาร์ทใช้เพียง 63 บรรทัด

ตารางที่ 4.19 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัววนซ้ำ

จาวา

```
1  package Iterator;
2  import java.util.*;
3  interface Iterator {
4    public void first();
5    public void next();
6    public boolean isDone();
7    public Object currentItem();
8  }
9  class ConcreteIterator implements Iterator {
10   private IntSet set;
11   private java.util.Iterator it;
12   private Object current;
13   ConcreteIterator(IntSet in) {
14     this.set = in;
15   }
16   public void first() {
17     this.it = set.getData().iterator();
18     next();
19   }
20   public void next() {
21     try {
22         current = this.it.next();
23     } catch (NoSuchElementException e) {
24         current = null;
25     }
26   }
27   public boolean isDone() {
28     return current == null;
29   }
30   public Object currentItem() {
31     return current;
32   }
33 }
34 class IntSet {
35   private TreeSet data = new TreeSet();
36   public void add(int in) {
```

ดาร์ท

```
1  library Iterator;
2  import 'dart:mirrors';
3  import 'package:quiver/collection.dart';
4  abstract class Iterators extends Iterator{
5    Object _set, _c;
6    String _mt;
7    Iterator _it;
8    Iterators(this._set, this._mt);
9    void first() {
10     InstanceMirror im = reflect(_set);
11     ClassMirror MyClassMirror = im.type;
12     Iterable<DeclarationMirror> fileds =
         MyClassMirror.declarations.values.where(
                 (dm) => dm is MethodMirror);
13     for(MethodMirror  vm in fileds){
14       String b = vm.simpleName.toString();
15       String d = (b.substring(8,b.length-2));
16       if ( d == this._mt){
17         this._it = (im.getField(vm.simpleName).reflectee).iterator;
18         break;
19       }
20     }
21     _it.moveNext();
22     next();
23   }
24   void next() {
25     try {
26       this._c = _it.current;
27       _it.moveNext();
28     } catch (e) {
29       _c = null;
30     }
31   }
32   bool isDone() => _c == null;
33   Object currentItem() => _c;
34 }
```

ตารางที่ 4.19 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัววนซ้ำ (ต่อ)

จาวา	ดาร์ท

37 this.data.add(in);
38 }
39 public TreeSet getData() {
40 return this.data;
41 }
42 public ConcreteIterator createIterator() {
43 return new ConcreteIterator(this);
44 }
45 }
 	
46 class Client {
47 public static void main(String[] args) {
48 IntSet set = new IntSet();
49 for (int i = 9; i > 0; --i) {
50 set.add(i);
51 }
52 Iterator it1 = set.createIterator();
53 Iterator it2 = set.createIterator();
54 for (it1.first(); !it1.isDone();it1.next()) {
55 System.out.print(it1.currentItem()+" ");
56 }
57 System.out.println();
58 for(it1.first(),it2.first();!it1.isDone();
it1.next(),it2.next()) {
59 System.out.print(it1.currentItem() + "
" + it2.currentItem() + " ");
60 }
61 }
62 }
	35 class ConcreteIterator extends Iterators {
	36 ConcreteIterator(Object i, String Data) : super(i,Data);
	37 }
	38 class IntSet {
	39 TreeSet _data = new TreeSet();
	40 void add(int i) {
	41 this._data.add(i);
	42 }
	43 TreeSet get Data => _data;
	44 ConcreteIterator createIterator(String Data)
	=> new ConcreteIterator(this, Data);
	45 }
	46 void main() {
	47 IntSet set = new IntSet();
	48 String v = "";
	49 for (int i = 9; i > 0; --i) {
	50 set.add(i);
	51 }
	52 Iterators it1 = set.createIterator("Data");
	53 Iterators it2 = set.createIterator("Data");
	54 for (it1.first();!it1.isDone();it1.next()){
	55 v += "${it1.currentItem()} ";
	56 }
	57 print(v);
	58 v = "";
	59 for (var x = it1.first(), y = it2.first();
	!it1.isDone();it1.next(), it2.next()) {
	60 v += "${it1.currentItem()} ${it2.currentItem()} ";
	61 }
	62 print(v);
	63 }

5. แพทเทิร์นตัวกลาง

จากตารางที่ 4.20 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน มีเพียงวากยสัมพันธ์ที่แตกต่างกันซึ่งได้กล่าวไปแล้วในแต่ละ

แพทเทิร์นที่ผ่านมา ส่วนวากยสัมพันธ์ที่พบในแพทเทิร์นนี้คือข้อจำกัดของการตั้งค่าเริ่มต้นให้เขตข้อมูลโดยอัตโนมัติในคอนสตรัคเตอร์ แม้ว่าดาร์ทจะมีจะออกแบบภาษาให้มีการตั้งค่าเริ่มต้นเขตข้อมูลโดยอัตโนมัติ แต่การจะเขียนโค้ดให้กระทำลักษณะดังกล่าว ตัวแปรที่ใช้กำหนดค่าให้จำเป็นจะต้องประกาศอยู่ในคลาสที่กำหนดให้คอนสตรัคเตอร์เริ่มต้นเขตข้อมูลโดยอัตโนมัติด้วย เช่น

```
class ConColleagueA extends Colleague {
  Mediator mediator;
  ConcolleagueA(this.mediator);
}
```

แต่ถ้าตัวแปรถูกประกาศไว้ในคลาสแม่แล้ว คอนสตรัคเตอร์ของคลาสลูกจะไม่สามารถกำหนดให้มีการเริ่มต้นเขตของข้อมูลโดยอัตโนมัติได้ จะต้องทำการเริ่มต้นเขตของข้อมูลแบบจาวา เช่น

```
abstract class Colleague {
  Mediator mediator;
}
class ConColleagueA extends Colleague {
  ConcolleagueA(Mediator m);
  this.mediator = m;
}
```

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวกลางในภาษาจาวาใช้ 60 บรรทัด ภาษาดาร์ทใช้เพียง 48 บรรทัด

ตารางที่ 4.20 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวกลาง

จาวา		ดาร์ท	
1	package Mediator;	1	library Mediator;
2	interface Mediator {	2	abstract class Mediator{
3	public void fight();	3	void fight();
4	public void talk();	4	void talk();
5	public void registerA(ConColleagueA a);	5	void registerA(ConColleagueA a);
6	public void registerB(ConColleagueB a);	6	void registerB(ConColleagueB a);
7	}	7	}
8	class ConMediator implements Mediator {	8	class ConMediator implements Mediator {

ตารางที่ 4.20 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวกลาง (ต่อ)

จาวา		ดาร์ท	
9	ConColleagueA talk;	9	ConColleagueA _talk;
10	ConColleagueB fight;	10	ConColleagueB _fight;
11	@Override	11	void registerA(ConColleagueA a) {
12	public void registerA(ConColleagueA a) {	12	this._talk = a;
13	talk = a;	13	}
14	}	14	void registerB(ConColleagueB b) {
15	@Override	15	this._fight = b;
16	public void registerB(ConColleagueB b) {	16	}
17	fight = b;	17	void fight()=>print("Mediator is fighting");
18	}	18	void talk() => print("Mediator is talking");
19	@Override	19	}
20	public void fight() {		
21	System.out.println("Mediator is fighting");		
22	}		
23	@Override		
24	public void talk() {		
25	System.out.println("Mediator is talking");		
26	}		
27	}		
28	abstract class Colleague {	20	abstract class Colleague {
29	Mediator mediator;	21	Mediator mediator;
30	public abstract void doSomething();	22	void doSomething();
31	}	23	}
32	class ConColleagueA extends Colleague {	24	class ConColleagueA extends Colleague {
33	public ConColleagueA(Mediator m) {	25	ConColleagueA(Mediator m) {
34	this.mediator = m;	26	this.mediator = m;
35	}	27	}
36	@Override	28	void doSomething() {
37	public void doSomething() {	29	this.mediator.talk();
38	this.mediator.talk();	30	this.mediator.registerA(this);
39	this.mediator.registerA(this);	31	}
40	}	32	}
41	}		
42	class ConColleagueB extends Colleague {	33	class ConColleagueB extends Colleague {
43	public ConColleagueB(Mediator m) {	34	ConColleagueB(Mediator m) {
44	this.mediator = m;	35	this.mediator = m;

ตารางที่ 4.20 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวกลาง (ต่อ)

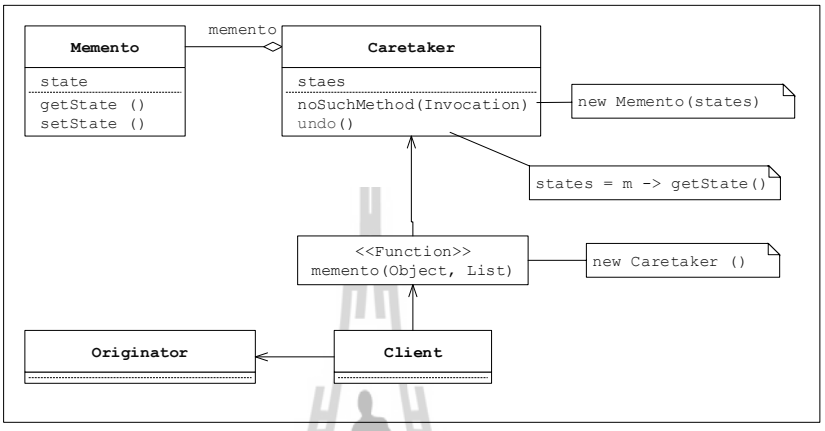
จาวา	ดาร์ท
45 }	36 }
46 @Override	37 void doSomething() {
47 public void doSomething() {	38 this.mediator.fight();
48 this.mediator.fight();	39 this.mediator.registerB(this);
49 this.mediator.registerB(this);	40 }
50 }	41 }
51 }	
52 class Client {	42 void main() {
53 public static void main(String[] args) {	43 Mediator m = new ConMediator();
54 Mediator m = new ConMediator();	44 ConColleagueA talk = new ConColleagueA(m);
55 ConColleagueA talk = new ConColleagueA(m);	45 ConColleagueB fight = new ConColleagueB(m);
56 ConColleagueB fight = new ConColleagueB(m);	46 talk.doSomething();
57 talk.doSomething();	47 fight.doSomething();
58 fight.doSomething();	48 }
59 }	
60 }	

6. แพทเทิร์นที่ระลึก

แพทเทิร์นที่ระลึกได้ทำการปรับปรุงโครงสร้างของแพทเทิร์นดังแสดงในรูปที่ 4.3 จากเดิมโครงสร้างของแพทเทิร์นจะประกอบด้วยคลาส 3 คลาส และการสร้างอินสแตนซ์ของคลาส Memento สำหรับการเก็บสถานะของอ๊อบเจกต์จะถูกสร้างภายในคลาส Originator การปรับปรุงโครงสร้างของแพทเทิร์นการปรับปรุงเกิดขึ้นที่คลาส Caretaker คลาส Originator และมีการเพิ่มฟังก์ชัน memento() โดยไม่ได้ทำการปรับปรุงคลาส Memento สำหรับฟังก์ชัน memento() ถูกกำหนดให้เป็นฟังก์ชันสำหรับให้คลาส Client เรียกใช้เพื่อทำการสร้างอินสแตนซ์ของคลาส Caretaker และการสร้างอินสแตนซ์ของคลาส Memento จะถูกสร้างโดยคลาส Caretaker ทำให้คลาส Originator ซึ่งเป็นคลาสเป้าหมายที่จะทำการเก็บสถานะนั้นสามารถเปลี่ยนแปลงเป็นคลาสอื่นได้ทันทีโดยไม่ต้องคำนึงถึงการสร้างอ๊อบเจกต์เพื่อเก็บสถานะเมื่อมีการเปลี่ยนแปลง คลาส Caretaker จากเดิมที่ภายในจะมีเฉพาะเมธอดสำหรับกลับไปยังสถานะก่อนหน้า ได้มีการเพิ่มเมธอด noSuchMethod() ซึ่งเป็นเมธอดที่จะทำการสร้างอินสแตนซ์ของ Memento เพื่อเก็บสถานะอ๊อบเจกต์ของคลาส Originator

การเรียกใช้งานของ Client จะเรียกใช้งานคลาส Caretaker ผ่านทางฟังก์ชัน

memento() โดยการส่งค่าของอินสแตนซ์ของ Originator และค่าของขอบเขต (Fields) ที่ต้องการเก็บค่าให้กับ memento()แพทเทิร์นที่ระลึกไปใช้สามารถนำไปใช้ซ้ำได้โดยการเปลี่ยนแปลงคลาส Originator



รูปที่ 4.3 โครงสร้างแพทเทิร์นที่ระลึกที่ได้ทำการปรับปรุง

จากตารางที่ 4.21 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีลักษณะในการพัฒนาที่แตกต่างกัน เนื่องมาจากการปรับปรุงโครงสร้างของแพทเทิร์น สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นที่ระลึกในภาษาจาวาใช้ 50 บรรทัด ภาษาดาร์ทใช้ 54 บรรทัด

ตารางที่ 4.21 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นที่ระลึก

จาวา	ดาร์ท

1 package Memento;
2 import java.util.*;
3 class Memento {
4 private String state;
5 public Memento(String stateToSave) {
6 this.state = stateToSave;
7 }
8 public String getSavedState() {
9 return this.state;
10 }
	1 library Memento;
	2 import 'dart:mirrors';
	3 class Memento {
	4 Map<Symbol, dynamic> _state = {};
	5 Memento(this._state);
	6 Map get getState => this._state;
	7 }
	8 class Caretaker {
	9 Object _o;
	10 List<Symbol> _ls = [];

ตารางที่ 4.21 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นที่ระลึก (ต่อ)

จาวา	ดาร์ท
11 }	11 Map<int, Memento> _saveStates = {};
12 class Caretaker {	12 int index = 0;
13 private ArrayList<Memento> savedStates = new ArrayList<>();	13 Caretaker(this._o,this._ls);
14 public void addMemento(Memento m) {	14 noSuchMethod(Invocation iv){
15 savedStates.add(m);	15 Map<Symbol, dynamic> _m = {};
16 }	16 InstanceMirror im = reflect(_o);
17 public Memento getMemento() {	17 _ls.forEach((Symbol s){ _m[s]= im.getField(s).reflectee;});
18 int index = (savedStates.size()) - 1;	18 _saveStates[index++] = new Memento(_m);
19 return savedStates.remove(index - 1);	19 im.invoke(iv.memberName, iv.positionalArguments, iv.namedArguments
20 }	20);
21 }	21 }
22 class Originator {	22 undo(){
23 private String state;	23 if(_saveStates.isNotEmpty){
24 public void set(String state) {	24 Map<Symbol, dynamic> _gm = _saveStates.values.last.getState;
25 System.out.println("Setting state to " + state);	25 _saveStates.remove(_saveStates.length-1);
26 this.state = state;	26 _gm.forEach((Symbol K, dynamic V){ reflect(_o).setField(K, V);});
27 }	27 }else{
28 public Memento saveToMemento() {	28 print("-------State's empty!!!-------");
29 System.out.println("Saving to Memento : "+ state);	29 }
30 return new Memento(state);	30 }
31 }	31 }
32 public void restoreFromMemento(Memento m) {	32 memento(Object o,List l) => new Caretaker(o,l);
33 state = m.getSavedState();	33 class Originator {
34 System.out.println("State after restoring from Memento: " + state);	34 String _state =null;
35 }	35 void m(String state) => this. _state = state;
36 }	36 }
37 class Client {	37 }
38 public static void main(String[] args) {	38 void main() {
39 Caretaker caretaker = new Caretaker();	39 Originator o = new Originator();
40 Originator originator = new Originator();	40 Caretaker c = memento(o,[#str]);
41 originator.set("State0");	41 print("Setting state to State 0");
42 caretaker.addMemento(originator.saveToMemento());	42 c.m('State 0');
43 originator.set("State1");	
44 caretaker.addMemento(originator.saveToMemento());	

ตารางที่ 4.21 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นที่ระลึก (ต่อ)

จาวา		ดาร์ท	
45	originator.set("State2");	43	print("Saving to Memento : State 0");
46	caretaker.addMemento(originator.saveToMemento());	44	print("Setting state to State 1");
47	originator.restoreFromMemento(	45	c.m('State 1');
	caretaker.getMemento());	46	print("Saving to Memento : State 1");
48	originator.restoreFromMemento(	47	print("Setting state to State 2");
	caretaker.getMemento());	48	c.m('State 2');
49	}	49	print("Saving to Memento : State 2");
50	}	50	c.undo();
		51	print("State after restoring from Memento: ${o.str}");
		52	c.undo();
		53	print("State after restoring from Memento: ${o.str}");
		54	}

7. แพทเทิร์นตัวสังเกตการณ์

จากตารางที่ 4.22 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีลักษณะในการพัฒนาที่แตกต่างกัน พิจารณาคลาส BinObs มีการประกาศตัวแปร convertObsrv สำหรับเก็บค่าอินสแตนซ์ของคลาสภายในที่ไม่ระบุชื่อ (Anonymous inner class) ของอินเตอร์เฟซ Observer เพื่อทำการโอเวอร์ไรด์เมธอด update (Observable obs, Object ob) สำหรับเปลี่ยนแปลงข้อมูลเมื่อได้รับการแจ้งเตือน ส่วนในดาร์ทได้พัฒนาฟังก์ชัน binObs(int _dex) ในบรรทัดที่ 2-3 ซึ่งมีค่าเทียบเท่ากับคลาส BinObs ของจาวาในบรรทัดที่ 6-20 ภายในฟังก์ชัน binObs(int _dex)มีการส่งค่ากลับเป็นฟังก์ชัน ซึ่งก็คือฟังก์ชันไม่ระบุชื่อในบรรทัดที่ 3 ของดาร์ทและมีค่าเทียบเท่ากับเมธอด update(Observable obs, Object ob) ในจาวา

คลาส Observable สำหรับตัวแปร observer ที่ใช้ในการเก็บค่าสำหรับการเปลี่ยนแปลงข้อมูลเมื่อได้รับการแจ้งเตือน ในจาวาใช้คลาส Vector เป็นแบบชนิดของตัวแปร ซึ่งจะต้องทำการเก็บค่าของ observer ไว้ในตัวแปรที่เป็นอะเรย์ก่อน ในบรรทัดที่ 30 จะสามารถนำไปใช้ได้ ส่วนในดาร์ทใช้แบบชนิดที่เป็น List<Function> เมื่อจะนำไปใช้จะต้องทำเก็บค่าไว้ในตัวแปรที่มีแบบชนิดเป็น Function ในบรรทัดที่ 13

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวสังเกตการณ์ในภาษาจาวาใช้ 65 บรรทัด ภาษาดาร์ทใช้เพียง 44 บรรทัด

ตารางที่ 4.22 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวสังเกตการณ์

จาวา

```
1  package Observers;
2  import java.util.Vector;
3  interface Observer {
4    void update(Observable o, Object arg);
5  }
6  class BinObs {
7    private int dex;
8    private Observer convertObsrv = new Observer() {
9      @Override
10     public void update(Observable obs, Object ob){
11       System.out.println("Bin : " +
                                  Integer.toBinaryString(dex));
12     }
13   };
14   BinObs(int dex) {
15     this.dex = dex;
16   }
17   public Observer convrtObs() {
18     return this.convertObsrv;
19   }
20 }
21 class Observable {
22   Vector observer = new Vector();
23   public void addObs(Observer o) {
24     observer.addElement(o);
25   }
26   public void deleteObs() {
27     observer.removeAllElements();
28   }
29   public void notifyObs() {
30     Object[] obs = observer.toArray();
31     for (int i = obs.length-1; i >= 0; i--)
32       ((Observer) obs[i]).update(this, null);
33   }
34 }
35 class Subject extends Observable {
```

ดาร์ท

```
1  library Observer;


2  binObs(int _dex) =>
3    (obs,ob)=>print(("Bin: ${_dex.toRadixString(2)}"));


4  class Observable{
5    List<Function> observer = [];
6    void addObs(Function obs => this.observer.add(obs);
7    void deleteObs(){
8      for(int i = observer.length -1 ;i>=0;i--)
9        this.observer.remove(observer[i]);
10   }
11   void notifyObs() {
12     for(int i = observer.length - 1;i>=0;i--){
13       Function fObserver = this.observer[i];
14       fObserver(this, null);
15     }
16   }
17 }
18 class Subject extends Observable{
```

ตารางที่ 4.22 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวสังเกตการณ์ (ต่อ)

จาวา	ดาร์ท

36 private boolean isOpen;
37 public Subject() {
38 this.isOpen = false;
39 }
40 @Override
41 public void notifyObs() {
42 if (isOpen) {
43 super.notifyObs();
44 }
45 }
46 public void convert() {
47 this.isOpen = true;
48 this.notifyObs();
49 }
50 }
51 class Client {
52 private int dex = 15;
53 Subject sj = new Subject();
54 public void test() {
55 BinObs bin = new BinObs(dex);
56 System.out.println("Input Dex: " + dex);
57 sj.addObs(bin.convrtObs());
58 sj.convert();
59 sj.deleteObs();
60 sj.convert();
61 }
62 public static void main(String[] args) {
63 new Client().test();
64 }
65 }
	19 bool _isOpen;
	20 Subject() {
	21 this._isOpen = false;
	22 }
	23 void notifyObs() {
	24 if(_isOpen)
	25 super.notifyObs();
	26 }
	27 void convert() {
	28 this._isOpen = true;
	29 this.notifyObs();
	30 }
	31 }
	32 class Client {
	33 int _dex = 15;
	34 Subject sj = new Subject();
	35 void test(){
	36 Function bObs = binObs(_dex);
	37 print("Input Dex : $_dex");
	38 sj.addObs(bObs);
	39 sj.convert();
	40 sj.deleteObs();
	41 sj.convert();
	42 }
	43 }
	44 void main() => new Client().test();

8. แพทเทิร์นสถานะ

จากตารางที่ 4.23 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน มีเพียงวากยสัมพันธ์ที่แตกต่างกันซึ่งได้กล่าวไปแล้วในแต่ละแพทเทิร์นที่ผ่านมา คุณสมบัติของภาษาดาร์ทที่พบในแพทเทิร์นนี้คือ การระบุคำสงวน final ให้กับ

คลาส จาวาสามารถระบุคำสงวน final ให้กับคลาสได้ ดังตัวอย่างในบรรทัดที่ 19 ที่มีการระบุคำสงวน final ให้กับคลาส Context ในขณะที่คลาส Context ของดาร์ท ในบรรทัดที่ 17 ไม่ได้มีการระบุคำสงวน final ไว้ เนื่องจากดาร์ทไม่สามาถระบุคำสงวน final ให้กับคลาสได้

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นสถานะในภาษาจาวาใช้ 40 บรรทัด ภาษาดาร์ทใช้เพียง 32 บรรทัด

ตารางที่ 4.23 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นสถานะ

จาวา

```
1  package State;
2  interface State {
3    void write(final Context context,final String msg);
4  }
5  class UpperState implements State {
6    @Override
7    public void write(final Context context, final String msg){
8      System.out.println(msg.toUpperCase());
9      context.setState(new LowerState());
10   }
11 }
12 class LowerState implements State {
13   @Override
14   public void write(final Context context, final String msg){
15     System.out.println(msg.toLowerCase());
16     context.setState(new UpperState());
17   }
18 }
19 final class Context {
20   private State nowState;
21   public Context() {
22     setState(new LowerState());
23   }
24   public void setState(final State new_state) {
25     nowState = new_state;
26   }
27   public void write(final String msg) {
28     nowState.write(this, msg);
29   }
```

ดาร์ท

```
1  library State;
2  abstract class CaseState {
3    void write(final Context context,final String msg);
4  }
5  class UpperState implements CaseState {
6    void write(final Context context,final String msg){
7      print(msg.toUpperCase());
8      context.State = new LowerState();
9    }
10 }
11 class LowerState implements CaseState {
12   void write(final Context context,final String msg){
13     print(msg.toLowerCase());
14     context.State = new UpperState();
15   }
16 }
17 class Context{
18   CaseState _nowState;
19   Context() {
20     State = new LowerState();
21   }
22   set State(final CaseState new_state) =>
                                  _nowState = new_state;
23   void write(final String msg) =>
                                  _nowState.write(this, msg);
24 }
```

ตารางที่ 4.23 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นสถานะ (ต่อ)

จาวา		ดาร์ท	
30	}		
31	public class Client {	25	void main() {
32	public static void main(String[] args) {	26	final Context cc = new Context();
33	final Context cc = new Context();	27	cc.write("Monday");
34	cc.write("Monday");	28	cc.write("tuesDAY");
35	cc.write("tuesDAY");	29	cc.write("WEDNESDAY");
36	cc.write("WEDNESDAY");	30	cc.write("thursday");
37	cc.write("thursday");	31	cc.write("FrIdAy");
38	cc.write("FrIdAy");	32	}
39	}		
40	}		

9. แพทเทิร์นกลยุทธ์

จากตารางที่ 4.24 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน ที่มีเพียงวากยสัมพันธ์ที่แตกต่างกันซึ่งได้กล่าวไปแล้วในแต่ละแพทเทิร์นที่ผ่านมา สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นกลยุทธ์ในภาษาจาวาใช้ 37 บรรทัด ภาษาดาร์ทใช้เพียง 25 บรรทัด

ตารางที่ 4.24 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นกลยุทธ์

จาวา		ดาร์ท	
1	package Strategy;	1	library Strategy;
2	interface MathOperation {	2	abstract class MathOperation {
3	double compute(double x, double y);	3	num compute(num x, num y);
4	}	4	}
5	class Plus implements MathOperation {	5	class Plus implements MathOperation {
6	@Override	6	num compute(num x, num y) => x + y;
7	public double compute(double x, double y) {	7	}
8	return x + y;		
9	}		
10	}		
11	class Minus implements MathOperation {	8	class Minus implements MathOperation {
12	@Override	9	num compute(num x, num y) => x - y;
13	public double compute(double x, double y) {	10	}

ตารางที่ 4.24 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นกลยุทธ์ (ต่อ)

จาวา		ดาร์ท	
14	return x - y;		
15	}		
16	}		
17	class MathContext {	11	class MathContext {
18	private MathOperation mathOperation;	12	MathOperation _mathOperation;
19	public MathContext(MathOperation mathOperation){	13	MathContext(this._mathOperation);
20	this.mathOperation = mathOperation;	14	num compute (num x, num y) => this._mathOperation.compute(x, y);
21	}	15	}
22	public double compute (double x, double y) {		
23	return this.mathOperation.compute(x, y);		
24	}		
25	}		
26	public class Client {		
27	public static void main(String[] args) {	16	void main() {
28	MathContext mathContext;	17	MathContext mathContext;
29	double x = 10, y = 3;	18	num x = 10.0, y = 3.0;
30	mathContext = new MathContext(new Plus());	19	mathContext = new MathContext(new Plus());
31	double resultI = mathContext.compute(x, y);	20	num resultI = mathContext.compute(x, y);
32	System.out.println(x+" + "+ y + " = " + resultI);	21	print("$x + $y = $resultI");
33	mathContext = new MathContext(new Minus());	22	mathContext = new MathContext(new Minus());
34	double resultII = mathContext.compute(x, y);	23	num resultII = mathContext.compute(x, y);
35	System.out.println(x+" - "+y + " = " + resultII);	24	print("$x - $y = $resultII");
36	}	25	}
37	}		

10. แพทเทิร์นเมธอดแม่แบบ

จากตารางที่ 4.25 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน มีเพียงวากยสัมพันธ์ที่แตกต่างกันซึ่งได้กล่าวไปแล้วในแต่ละแพทเทิร์นที่ผ่านมา สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นเมธอดแม่แบบในภาษาจาวาใช้ 53 บรรทัด ภาษาดาร์ทใช้เพียง 37 บรรทัด

ตารางที่ 4.25 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นเมธอดแม่แบบ

จาวา	ดาร์ท

1 package TemplateMethod;
2 abstract class Generalization {
3 public void findSolution() {
4 stepOne();
5 stepTwo();
6 stepThree();
7 stepFor();
8 }
9 protected void stepOne() {
10 System.out.println("Generalization.stepOne");
11 }
12 abstract protected void stepTwo();
13 abstract protected void stepThree();
14 protected void stepFor() {
15 System.out.println("Generalization.stepFor");
16 }
17 }
18 abstract class Specialization extends Generalization{
19 @Override
20 protected void stepThree() {
21 step3_1();
22 step3_2();
23 step3_3();
24 }
25 protected void step3_1() {
26 System.out.println("Specialization.step3_1");
27 }
28 abstract protected void step3_2();
29 protected void step3_3() {
30 System.out.println("Specialization.step3_3");
31 }
32 }
33 class Realization extends Specialization {
34 @Override
35 protected void stepTwo() {
36 System.out.println("Realization .stepTwo");
	1 library TemplateMethod;
	2 abstract class Generalization {
	3 void findSolution() {
	4 _stepOne();
	5 _stepTwo();
	6 _stepThree();
	7 _stepFour();
	8 }
	9 void _stepOne() => print("Generalization.stepOne");
	10 void _stepTwo();
	11 void _stepThree();
	12 void _stepFour()=>print("Generalization.stepFour");
	13 }
	14 abstract class Specialization extends Generalization{
	15 void _stepThree() {
	16 _step3_1();
	17 _step3_2();
	18 _step3_3();
	19 }
	20 void _step3_1() => print("Specialization.step3_1");
	21 void _step3_2();
	22 void _step3_3() {
	23 print("Specialization.step3_3");
	24 }
	25 }
	26 class Realization extends Specialization {
	27 void _stepTwo() => print("Realization .stepTwo");
	28 void _step3_2() => print("Realization .step3_2");
	29 void _stepFour() {

ตารางที่ 4.25 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นเมธอดแม่แบบ (ต่อ)

จาวา		ดาร์ท	
37	}	30	print("Realization .stepFour");
38	@Override	31	super._stepFour();
39	protected void step3_2() {	32	}
40	System.out.println("Realization .step3_2");	33	}
41	}		
42	@Override		
43	protected void stepFor() {		
44	System.out.println("Realization .stepFor");		
45	super.stepFor();		
46	}		
47	}		
48	class Client {		
49	public static void main(String[] args) {	34	void main() {
50	Generalization algorithm = new Realization();	35	Generalization algorithm = new Realization();
51	algorithm.findSolution();	36	algorithm.findSolution();
52	}	37	}
53	}		

11. แพทเทิร์นตัวเยี่ยมเยือน

จากตารางที่ 4.26 การพัฒนาตัวอย่างการใช้งานแพทเทิร์นนี้ทั้งจาวาและดาร์ท มีการพัฒนาด้วยลักษณะที่ใกล้เคียงกัน มีเพียงวากยสัมพันธ์ที่แตกต่างกันซึ่งได้กล่าวไปแล้วในแต่ละแพทเทิร์นที่ผ่านมา ส่วนวากยสัมพันธ์ที่พบในแพทเทิร์นนี้คือ การที่ดาร์ทไม่สนับสนุนการโอเวอร์โหลดเมธอด การตั้งชื่อของเมธอดจึงไม่สามารถตั้งชื่อของเมธอดเหมือนกันได้ เช่น

```
abstract class Visitor {
  void visitComputer(Computer computer);
  void visitMouse(Mouse mouse);
  void visitKeyboard(Keyboard keyboard);
  void visitMonitor(Monitor monitor);
}
```

ในขณะที่จาวาเป็นภาษาที่สนับสนุนการโอเวอร์โหลดเมธอด จึงสามารถตั้งชื่อของเมธอดเหมือนกันได้ เช่น

```
interface Visitor {
  public void visit(Computer computer);
  public void visit(Mouse mouse);
  public void visit(Keyboard keyboard);
  public void visit(Monitor monitor);
}
```

สำหรับจำนวนบรรทัดของรหัสต้นฉบับที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นตัวเยี่ยมเยือนในภาษาจาวาใช้ 65 บรรทัด ภาษาดาร์ทใช้เพียง 42 บรรทัด

ตารางที่ 4.26 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวเยี่ยมเยือน

จาวา

```
1  package Visitor;
2  interface ComputerPart {
3    public void accept(Visitor partVisitor);
4  }
5  class Monitor implements ComputerPart {
6    @Override
7    public void accept(Visitor partVisitor) {
8      partVisitor.visit(this);
9    }
10 }
11 class Keyboard implements ComputerPart {
12   @Override
13   public void accept(Visitor partVisitor) {
14     partVisitor.visit(this);
15   }
16 }
17 class Mouse implements ComputerPart {
18   @Override
19   public void accept(Visitor partVisitor) {
20     partVisitor.visit(this);
21   }
22 }
23 class Computer implements ComputerPart {
24   ComputerPart[] parts;
```

ดาร์ท

```
1  library Visitor;
2  abstract class ComputerPart {
3    factory ComputerPart(){}
4    void accept(Visitor partVisitor);
5  }
6  class Monitor implements ComputerPart {
7    void accept(Visitor partVisitor) =>
                    partVisitor.visitMonitor(this);
8  }
9  class Keyboard implements ComputerPart {
10   void accept(Visitor partVisitor) =>
                    partVisitor.visitKeyboard(this);
11 }
12 class Mouse implements ComputerPart {
13   void accept(Visitor partVisitor) =>
                    partVisitor.visitMouse(this);
14 }
15 class Computer implements ComputerPart {
16   List<ComputerPart> parts;
```

ตารางที่ 4.26 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวเยี่ยมเยือน (ต่อ)

จาวา

```
25  public Computer() {
26    parts = new ComputerPart[]{new Mouse(),
                    new Keyboard(),new Monitor()};
27  }
28  @Override
29  public void accept(Visitor partVisitor) {
30    for (int i = 0; i < parts.length; i++) {
31      parts[i].accept(partVisitor);
32    }
33    partVisitor.visit(this);
34  }
35 }
36 interface Visitor {
37  public void visit(Computer computer);
38  public void visit(Mouse mouse);
39  public void visit(Keyboard keyboard);
40  public void visit(Monitor monitor);
41 }
42 class DisplayPartVisitor implements Visitor {
43  @Override
44  public void visit(Computer computer) {
45    System.out.println("Displaying Computer.");
46  }
47  @Override
48  public void visit(Mouse mouse) {
49    System.out.println("Displaying Mouse.");
50  }
51  @Override
52  public void visit(Keyboard keyboard) {
53    System.out.println("Displaying Keyboard.");
54  }
55  @Override
56  public void visit(Monitor monitor) {
57    System.out.println("Displaying Monitor.");
58  }
59 }
```

ดาร์ท

```
17  Computer() {
18    parts = [new Mouse(), new Keyboard(),
            new Monitor()];
19  }
20  void accept(Visitor partVisitor) {
21    for (int i = 0; i < parts.length; i++) {
22      parts[i].accept(partVisitor);
23    }
24    partVisitor.visitComputer(this);
25  }
26 }
27 abstract class Visitor {
28  void visitComputer(Computer computer);
29  void visitMouse(Mouse mouse);
30  void visitKeyboard(Keyboard keyboard);
31  void visitMonitor(Monitor monitor);
32 }
33 class DisplayPartVisitor implements Visitor {
34  void visitComputer(Computer computer) =>
                print("Displaying Computer.");
35  void visitMouse(Mouse mouse) =>
              print("Displaying Mouse.");
36  void visitKeyboard(Keyboard keyboard) =>
                print("Displaying Keyboard.");
37  void visitMonitor(Monitor monitor) =>
                print("Displaying Monitor.");
38 }
```

ตารางที่ 4.26 รหัสต้นฉบับตัวอย่างการใช้งานของแพทเทิร์นตัวเยี่ยมเยือน (ต่อ)

จาวา	ดาร์ท

60 class Client {
61 public static void main(String[] args) {
62 ComputerPart computer = new Computer();
63 computer.accept(new DisplayPartVisitor());
64 }
65 }
	39 void main() {
	40 ComputerPart computer = new Computer();
	41 computer.accept(new DisplayPartVisitor());
	42 }

ตารางที่ 4.27 เปรียบเทียบจำนวนบรรทัดคำสั่งของภาษาจาวาและภาษาดาร์ท

แพทเทิร์น	จำนวนบรรทัด			ลดลง (%)	หมายเหตุ
	จาวา	ดาร์ท	ลดลง		
แพตเทิร์นเชิงการสร้าง					
โรงงานเชิงนามธรรม	69	42	26	38%	
ตัวสร้าง	60	44	16	27%	
เมธอดโรงงาน	47	42	5	11%	
ต้นแบบ	25	30	-5	-20%	นับรวมคลาส Cloneable
	25	18	7	28%	ไม่นับรวมคลาส Cloneable
เดี่ยว	17	11	5	31%	
แพตเทิร์นเชิงโครงสร้าง					
ตัวแปลง	38	27	11	29%	
บริดจ์	37	24	13	35%	
เชิงประกอบ	45	33	12	27%	
ตัวตกแต่ง	53	34	19	36%	
ส่วนหน้า	94	64	30	32%	
น้ำหนักเบา	64	45	19	30%	
ตัวแทน	43	33	10	23%	
แพทเทิร์นเชิงพฤติกรรม					
ห่วงโซ่ความรับผิดชอบ	70	60	10	14%	
คำสั่ง	54	38	16	30%	

ตารางที่ 4.27 เปรียบเทียบจำนวนบรรทัดคำสั่งของภาษาจาวาและภาษาดาร์ท (ต่อ)

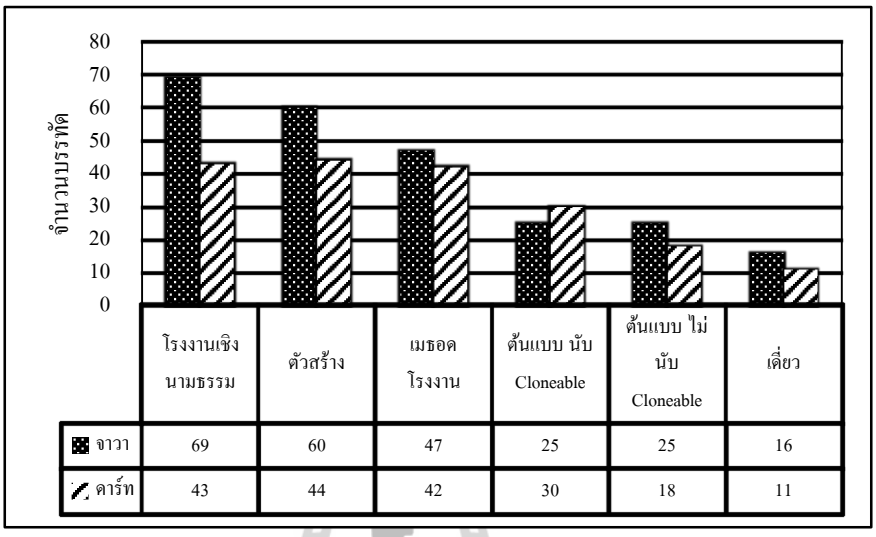
แพทเทิร์น	จำนวนบรรทัด			ลดลง	หมายเหตุ
	จาวา	ดาร์ท	ลดลง	(%)	
ตัวแปล	98	71	27	28%	
ตัววนซ้ำ	62	63	-1	-2%	
ตัวกลาง	60	48	12	20%	
ที่ระลึก	50	54	-4	-8%	
ตัวสังเกตการณ์	65	44	21	32%	
สถานะ	40	32	8	20%	
กลยุทธ์	37	25	12	32%	
เมธอดแม่แบบ	53	37	16	30%	
ตัวเยี่ยมเยือน	65	42	23	35%	

จากตารางที่ 4.27 พบว่าในแพทเทิร์นโรงงานเชิงนามธรรม ใช้จำนวนของบรรทัดคำสั่งลดลงถึง 38% ในขณะที่แพทเทิร์นที่ระลึกเพิ่มขึ้น 8% เนื่องมาจากการปรับปรุงโครงสร้างของแพทเทิร์นเพื่อให้สามารถนำรหัสต้นฉบับของแพทเทิร์นดังกล่าวกลับมาใช้ซ้ำได้ แพทเทิร์นตัววนซ้ำเพิ่มขึ้น 2% เนื่องมาจากการพัฒนาคลาส Iterators ให้สามารถใช้ซ้ำได้ และแพทเทิร์นต้นแบบหากมีการนับรวมคลาส Cloneable ซึ่งเป็นคลาสที่พัฒนาขึ้นเพื่อการคัดลอกอ๊อบเจกต์แล้ว จำนวนของบรรทัดคำสั่งจะเพิ่มขึ้น 20% แต่ถ้าไม่นับรวมแล้วจำนวนของบรรทัดคำสั่งจะลดลง 28%

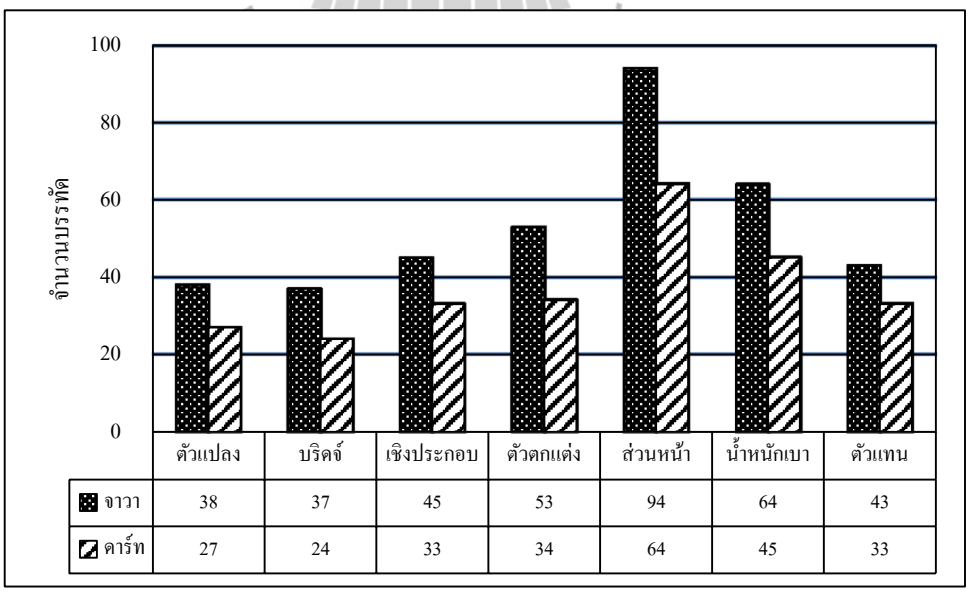
เมื่อนำข้อมูลจากตารางที่ 4.27 มาเปรียบเทียบเป็นกราฟดังแสดงในรูปที่ 4.4 สำหรับแพทเทิร์นเชิงการสร้าง จะพบว่ารหัสต้นฉบับภาษาดาร์ทในแพทเทิร์นเชิงการสร้างนี้ใช้จำนวนบรรทัดคำสั่งน้อยกว่าภาษาจาวามาก ยกเว้นในส่วนของแพทเทิร์นต้นแบบที่มีการนับรวมคลาส Cloneable ซึ่งเป็นคลาสที่พัฒนาขึ้นสำหรับการคัดลอกอ๊อบเจกต์ทำให้มีจำนวนบบรทัดที่มากกว่าจาวา

รูปที่ 4.5 สำหรับแพทเทิร์นเชิงโครงสร้าง จะพบว่ารหัสต้นฉบับในภาษาดาร์ทในแพทเทิร์นเชิงโครงสร้างนี้ใช้จำนวนบรรทัดคำสั่งที่น้อยกว่าภาษาจาวาทั้งหมด และรูปที่ 4.6 สำหรับแพทเทิร์นเชิงพฤติกรรม จะพบว่ารหัสต้นฉบับภาษาดาร์ทในแพทเทิร์นเชิงพฤติกรรมโดยส่วนใหญ่ใช้จำนวนบรรทัดคำสั่งที่น้อยกว่าภาษาจาวา ยกเว้นในส่วนของแพทเทิร์นตัววนซ้ำที่มีการพัฒนาให้

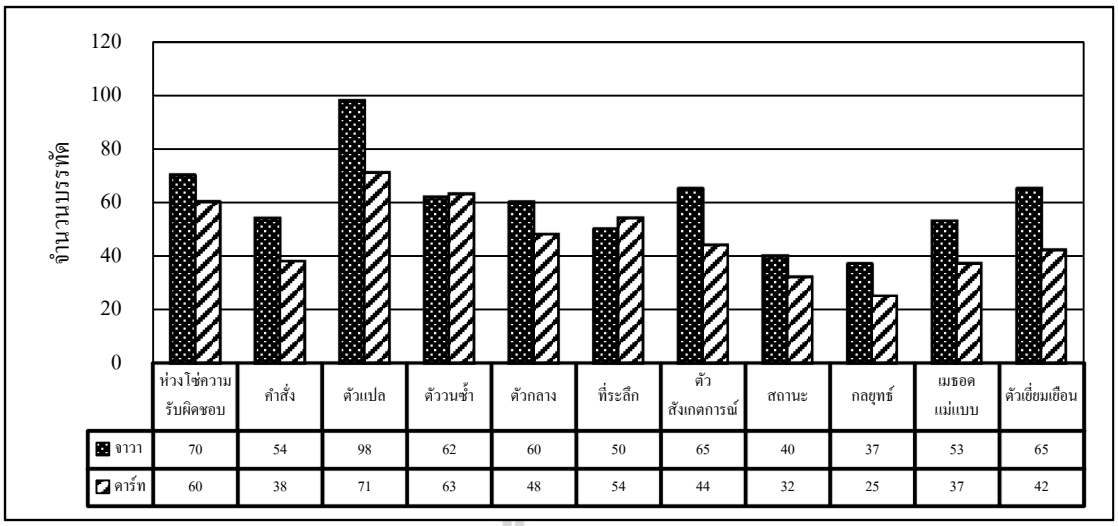
คลาส Iterators สามารถนำกลับไปใช้ซ้ำได้ และแพทเทิร์นที่ระลึกที่มีการปรับปรุงโครงสร้างของแพทเทิร์นเพื่อให้สามารถนำรหัสต้นฉบับที่ได้ทำการพัฒนากลับไปใช้ซ้ำได้ง่าย



รูปที่ 4.4 กราฟแสดงจำนวนบรรทัดคำสั่งในแพทเทิร์นเชิงการสร้าง



รูปที่ 4.5 กราฟแสดงจำนวนบรรทัดคำสั่งในแพทเทิร์นเชิงโครงสร้าง



รูปที่ 4.6 กราฟแสดงจำนวนบรรทัดคำสั่งในแพทเทิร์นเชิงพฤติกรรม

จากการศึกษาดีไซน์แพทเทิร์นทั้ง 23 แพทเทิร์น ทำให้สามารถสรุปความแตกต่างระหว่างภาษาจาวาและภาษาดาร์ทที่พบในระหว่างการศึกษาได้ดังแสดงในตารางที่ 4.28 โดยสามารถจำแนกออกได้ 3 ลักษณะ คือ คอนสตรัคเตอร์ เมธอด วากยสัมพันธ์ทั่วไปโดยมีรายละเอียดดังต่อไปนี้

1) คอนสตรัคเตอร์ของดาร์ทหากภายในตัวคอนสตรัคเตอร์ไม่มีคำสั่งใด ๆ สามารถใช้สัญลักษณ์ ; แทนการใช้สัญลักษณ์ { } ในการเริ่มต้นค่าของตัวแปรสามารถเขียนไว้ในพารามิเตอร์ของคอนสตรัคเตอร์ได้ เช่น BinObs(this.dex); แทนการเขียน BinObs (int dex) { this.dex = dex; } แต่ดาร์ทไม่สนับสนุนการโอเวอร์โหลดทำให้ไม่สามารถสร้างคอนสตรัคเตอร์มากกว่า 1 คอนสรัคเตอร์ได้ การจะสร้างมากกว่า 1 คอนสรัคเตอร์นั้นจึงจำเป็นต้องมีการตั้งชื่อให้กับคอนสรัคเตอร์ เช่น BinObs.first();

2) เมธอด

2.1) การสร้างเมธอดในจาวาเมธอด main() จำเป็นต้องสร้างไว้ภายในคลาส มีการกำหนดชนิดของเมธอดและพารามิเตอร์ที่ใช้รับค่า แต่ดาร์ทจะต้องสร้างเมธอด main() ไว้ภายนอกคลาสซึ่งไม่ต้องกำหนดชนิดของเมธอด และไม่จำเป็นต้องกำหนดพารามิเตอร์ให้กับเมธอด main()

2.2) การเขียนเมธอดทั่วไปที่มีการส่งค่ากลับหรือการพิมพ์ค่าเพื่อแสดงผลดาร์ทสามารถใช้คุณสมบัติของฟังก์ชันลูกศรในการเขียนเมธอดได้ โดยการส่งค่ากลับสามารถเขียนได้โดยไม่ต้องใช้คำสงวน return เช่น String draw() => "Drawing Rectangle"; แทนการเขียนด้วย String draw() { return "Drawing Rectangle"; }

2.3) การเขียนเมธอดที่มีเพียงการแสดงผลทางบรรทัดคำสั่งในดาร์ทก็สามารถใช้คุณสมบัติของฟังก์ชันลูกศรได้ เช่น void draw() => print("Drawing Rectangle");

2.4) การประกาศให้ตัวแปรเป็นตัวแปรแบบ final โดยมีการเริ่มต้นค่าให้กับตัวแปรนั้นในคอนสตรัคเตอร์ ในดาร์ทจะแตกต่างจากจาวา คือดาร์ทจะไม่สามารถทำเช่นนี้ได้

```
class CoffeeOrderContext{
 final int _tableNumber;
 CoffeeOrderContext(int tableNum){
  this._tableNumber = tableNum;   // Exception
 }
}
```

แต่จะสามารถทำได้เมื่อกำหนดให้มีการเริ่มต้นของเขตข้อมูลโดยอัตโนมัติ ดังนี้

```
class CoffeeOrderContext{
 final int _tableNumber;
 CoffeeOrderContext(this._tableNumber);
}
```

2.5) ดาร์ทมีการกำหนดการเข้าถึงและกำหนดเขตของข้อมูลแบบเกทเทอร์ และเซตเทอร์นั้น สามารถเขียนได้เหมือนกับจาวา แต่สามารถเขียนได้ด้วยคุณลักษณะของภาษาดาร์ท ทำให้สามารถเขียนได้ง่ายและสั้นลง สำหรับเกทเทอร์ใช้การระบุคำสงวน get ไว้ เช่น Menus get Menu => this._files.Menu และสำหรับเซตเทอร์ใช้โดยการระบุคำสงวน set ไว้ เช่น set File(Files files) => _files = files

3) วากยสัมพันธ์ทั่วไปจากการศึกษาสามารถสรุปได้ดังนี้

3.1) การเรียกใช้งานเซตเทอร์ในดาร์ท จะเรียกโดยใช้อินสแตนซ์ของคลาส ตามด้วยจุด ตามด้วยชื่อของเซตเทอร์ เครื่องหมายเท่ากับ และค่าที่ต้องการกำหนดเขตข้อมูล เช่น director.File = new Files()

3.2) การแปลงสตริงให้เป็นตัวเลขจำนวนจริงของทั้งสองภาษาใกล้เคียงกัน จาวาเขียนด้วย Double.parseDouble(“123”) ในขณะที่ดาร์ทเขียนเพียง double.parse(“123”)

3.3) ดาร์ทไม่มีการใช้คำสงวน interface เหมือนในจาวาแต่จะใช้ abstract class แทน ซึ่งสามารถกำหนดให้เป็นอินเตอร์เฟซได้โดยการใช้คำสงวน implements และให้เป็นคลาสนามธรรมสำหรับการสืบทอดโดยการใช้คำสงวน extends ทั้งนี้ขึ้นอยู่กับลักษณะของการใช้งาน

3.4) ลูป for จาวาสามารถเขียนให้อยู่ในลักษณะ for (Shapes shape : shapes) { shape.draw(); } ซึ่งดาร์ทสามารถเขียนในลักษณะ for (Shapes shape in shapes) { shape.draw(); } ได้เช่นกัน และยังสามารถเขียนให้อยู่ในลักษณะของ forEach คือ shapes.forEach ((Shapes shape) { shape.draw(); }); ได้อีกด้วย

3.5) การเขียนลูป for ที่มีการเริ่มต้นค่าภายใน 2 ค่า ในจาวาสามารถเขียนได้ง่ายกว่า เช่น

```
for(it1.first(),it2.first(); !it1.isDone(); it1.next(), it2.next()){}
```

ในขณะที่ดาร์ท จะต้องมีการเก็บค่าเริ่มต้นไว้ให้กับตัวแปรก่อน ซึ่งภายหลังไม่ได้มีการนำตัวแปรดังกล่าวมาใช้ เช่น

```
for(var x = it1.first(), y = it2.first(); !it1.isDone(); it1.next(), it2.next()){}
```

3.6) ในจาวาจะรวมคลาสไว้ใน package ส่วนในดาร์ทจะรวมไว้ใน library

3.7) ดาร์ทมีการกำหนดการเข้าถึงเพียง 2 ลักษณะคือ private และ public (ไม่มี protected เหมือนกับจาวา) การกำหนดการเข้าถึงโดยใช้ private จะใช้สัญลักษณ์ “_” เช่น int _dex; การเข้าถึงที่ไม่ได้เป็น private จะเป็น public ทั้งหมดโดยไม่ต้องระบุ public กำกับไว้

3.7) การเรียกใช้คอนสตรัคเตอร์ของคลาสแม่มีการเรียกใช้ผ่าน super ทั้งจาวาและดาร์ทแต่ในดาร์ทใช้เครื่องหมายทวิภาค (:) ขั้นระหว่างคอนสตรัคเตอร์ของคลาสลูกกับ super() เช่น BorderDecorator(Widget w) : super(w)

3.8) ดาร์ทใช้ลิสต์ในการเก็บชุดของตัวแปรที่เป็นชนิดเดียวกัน ซึ่งมีคุณสมบัติคล้ายกับอะเรย์ในจาวา

3.9) สำหรับการใช้ประโยคควบคุม switch ในจาวาสามารถกำหนดให้เงื่อนไขใน case สิ้นสุดการทำงานด้วยคำสั่ง break โดยเขียนไว้ในวงเล็บของ case ได้ดังนี้

```
switch(x){
   case “+” : { break; }
}
```

ในขณะที่ดาร์ทจำต้องกำหนด break ไว้นอกวงเล็บของเงื่อนไขของ case ดังนี้

```
switch(x){
   case “+” : { } break;
}
```

3.6) การพิมพ์ค่าออกทางหน้าจอดาร์ทสามารถเขียนได้สั้นกว่า คือ print("Input Dex : $_dex"); ในขณะที่จาวาเขียนด้วยวากยสัมพันธ์ที่ยาวกว่า คือ System.out.println("Input Dex: " + dex);

ตารางที่ 4.28 สรุปความแตกต่างระหว่างภาษาจาวาและภาษาดาร์ทที่พบใน 23 แพทเทิร์น

จาวา	ดาร์ท
คอนสตรัคเตอร์	
BinObs() {} BinObs(int dex) { this.dex = dex; }	BinObs.first(); BinObs(this.dex);
เมธอด	

class Main {
public static void main(String[] args){}
}
	void main(){}

public String draw(){
return "Drawing Rectangle";
}
	String draw() => "Drawing Rectangle";

public void draw(){
System.out.println("Drawing Rectangle");
}
	void draw() => print("Drawing Rectangle");

class CoffeeOrderContext{
private final int tableNumber;
public CoffeeOrderContext(int tableNum){
this.tableNumber = tableNum;
}
	class CoffeeOrderContext{
	final int _tableNumber;
	CoffeeOrderContext(this._tableNumber);
	}

public Menus getMenu() {
return this.file.getMenu();
}
	Menus get Menu => this._files.Menu;

public void setFile(Files file) {
this.file = file;
}
	set File(Files files) => _files = files;

วากยสัมพันธ์ทั่วไป	
director.setFile(new Files());	director.File = new Files();

ตารางที่ 4.28 สรุปความแตกต่างระหว่างภาษาจาวาและภาษาดาร์ทที่พบใน 23 แพทเทิร์น (ต่อ)

จาวา	ดาร์ท
Double.parseDouble(“123”)	double.parse(“123”)
interface	abstract class

for(Shapes shape : shapes)
  shape.draw();
	for(Shapes shape in shapes)
	  shape.draw(); // หรือ
	shapes.forEach((Shapes shape){shape.draw();});

for (it1.first(), it2.first(); !it1.isDone(); it1.next(),
it2.next()) {
/* Source code */
}
	for (var x = it1.first(), y = it2.first(); !it1.isDone(); it1.next(),
	it2.next()) {
	/* Source code */
	}

package	library
private	_ (underscore)
public	ถ้าไม่มีการระบุว่าเป็น private จะเป็น public ทั้งหมด

class BorderDecorator extends Decorator {
  public BorderDecorator(Widget w) {
  super(w);
  }
}
	class BorderDecorator extends Decorator {
	  BorderDecorator(Widget w) : super(w);
	}

Shapes[] shapes = {};	List<Shapes> shapes = [];

switch(x){
  case “+” : {
  /* Source code */
  break;
  }
}
	switch(x){
	  case “+” : {
	  /* Source code */
	  }
	  break;
	}

System.out.println("Input Dex: " + dex);	print("Input Dex : $_dex");

4.3 การทดสอบประสิทธิภาพภาษาดาร์ท

การทดสอบประสิทธิภาพของภาษาในการวิจัยครั้งนี้จะศึกษาเฉพาะในส่วนของเวลาที่ใช้ในการประมวลผลของแต่ละแพทเทิร์น สำหรับภาษาดาร์ท และภาษาจาวา โดยใช้ DeltaBlue

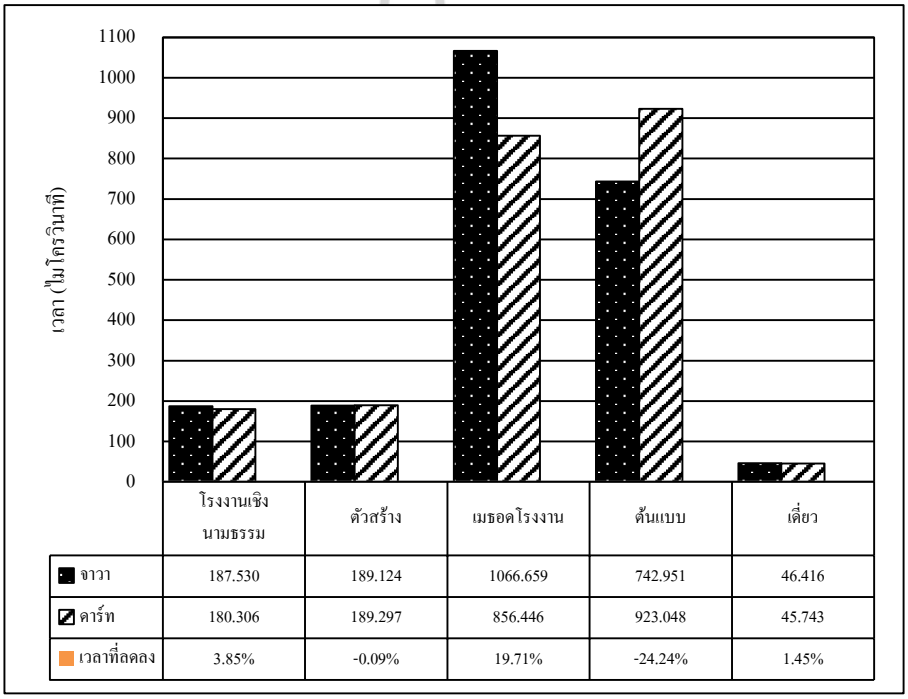
Benchmark ศึกษาจากตัวอย่างการใช้งานจากทั้ง 2 ภาษา ซึ่งประกอบไปด้วยรหัสต้นฉบับจำนวน 2 ชุดต่อหนึ่งแพทเทิร์นจากการออกแบบและพัฒนา ทำการเปรียบเทียบทีละแพทเทิร์นให้ครบทั้ง 23 แพทเทิร์น ในการทดสอบจะทำโดยการนำรหัสต้นฉบับชุดที่ 1 รหัสต้นฉบับชุดที่ 2 เปรียบเทียบเวลาที่ใช้ในการประมวลผลแต่ละแพทเทิร์นเพื่อทดสอบประสิทธิภาพของภาษาดาร์ท ผลที่ได้ดังแสดงในตารางที่ 4.29

ตารางที่ 4.29 เวลาที่ใช้ในการประมวลผลจาวา และภาษาดาร์ท

แพทเทิร์น	**เวลาที่ใช้ในการประมวลผล** (ไมโครวินาที)		**เวลาที่ลดลง**	**ลดลง (%)**
	จาวา	**ดาร์ท**		
แพตเทิร์นเชิงการสร้าง				
โรงงานเชิงนามธรรม	187.830	180.306	7.524	4.01%
ตัวสร้าง	189.124	189.297	-0.173	-0.09%
เมธอดโรงงาน	1066.659	856.446	210.213	19.71%
ต้นแบบ	742.951	923.048	-180.097	-24.24%
เดี่ยว	46.416	45.743	0.673	1.45%
แพตเทิร์นเชิงโครงสร้าง				
ตัวแปลง	532.345	559.951	-27.606	-5.19%
บริดจ์	170.169	173.102	-2.933	-1.72%
เชิงประกอบ	188.392	208.106	-19.714	-10.46%
ตัวตกแต่ง	436.816	433.131	3.685	0.84%
ส่วนหน้า	1984.787	1665.698	319.089	16.08%
น้ำหนักเบา	171071.581	2022.044	169049.537	98.82%
ตัวแทน	1178.577	1174.303	4.274	0.36%
แพทเทิร์นเชิงพฤติกรรม				
ห่วงโซ่ความรับผิดชอบ	1789.905	1843.182	-53.277	-2.98%
คำสั่ง	203.096	204.006	-0.910	-0.45%
ตัวแปล	236.826	260.585	-23.759	-10.03%
ตัววนซ้ำ	531.022	387.172	143.850	27.09%
ตัวกลาง	290.757	285.724	5.033	1.73%
ที่ระลึก	1645.359	1686.413	-41.054	-2.50%

ตารางที่ 4.29 เวลาที่ใช้ในการประมวลผลจาวา และภาษาดาร์ท (ต่อ)

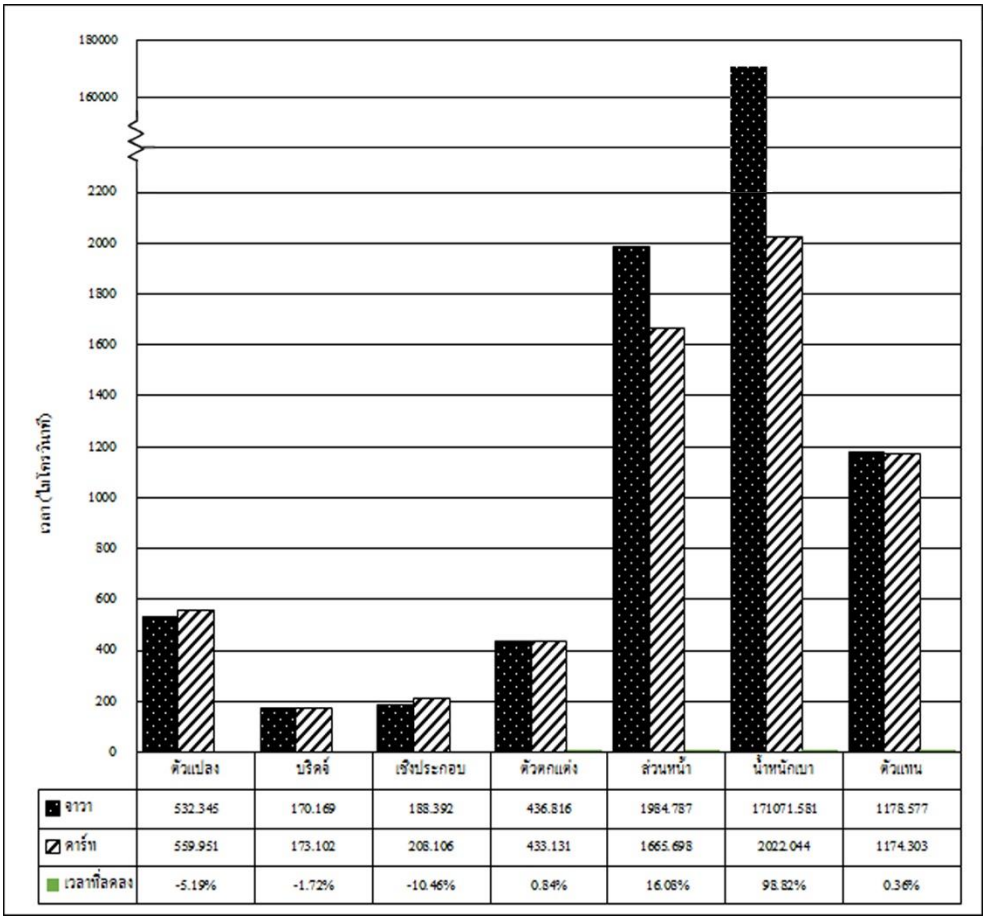
แพทเทิร์น	เวลาที่ใช้ในการประมวลผล (ไมโครวินาที)		เวลาที่ลดลง	ลดลง (%)
	จาวา	ดาร์ท		
ตัวสังเกตการณ์	192.100	183.483	8.617	4.49%
สถานะ	345.590	337.583	8.007	2.32%
กลยุทธ์	263.026	267.180	-4.154	-1.58%
เมธอดแม่แบบ	1128.035	1121.328	6.707	0.59%
ตัวเยี่ยมเยือน	565.669	559.324	6.345	1.12%



รูปที่ 4.7 กราฟแสดงเวลาที่ใช้ในการประมวลผลภาษาจาวา และภาษาดาร์ทของแพทเทิร์นเชิงการสร้าง

เมื่อนำข้อมูลจากตารางที่ 4.29 มาเปรียบเทียบเป็นกราฟสำหรับแพทเทิร์นเชิงการสร้างดังแสดงในรูปที่ 4.7 จะพบว่าสำหรับแพทเทิร์นโรงงานเชิงนามธรรม แพทเทิร์นตัวสร้าง และแพทเทิร์นเดี่ยว ใช้เวลาในการประมวลผลที่ใกล้เคียงกันมากโดยดาร์ทใช้เวลาลดลงเพียง 3.58%

-0.09% และ 1.45% ตามลำดับ ในส่วนของแพทเทิร์นเมธอดโรงงานดาร์ทใช้เวลาในการประมวลผลที่น้อยกว่าจาวามาก โดยเวลาที่ใช้เวลาในการประมวลผลลดลงถึง 19.71% ส่วนแพทเทิร์นต้นแบบดาร์ทใช้เวลาในการประมวลผลมากกว่าจาวา โดยใช้เวลาในการประมวลผลมากกว่าถึง 24.24% เนื่องเมธอดสำหรับคัดลอกในดาร์ทได้ถูกพัฒนาขึ้นมาใหม่ ในขณะที่จาวาถูกฝังไว้ในเครื่องจักรเสมือนแล้ว

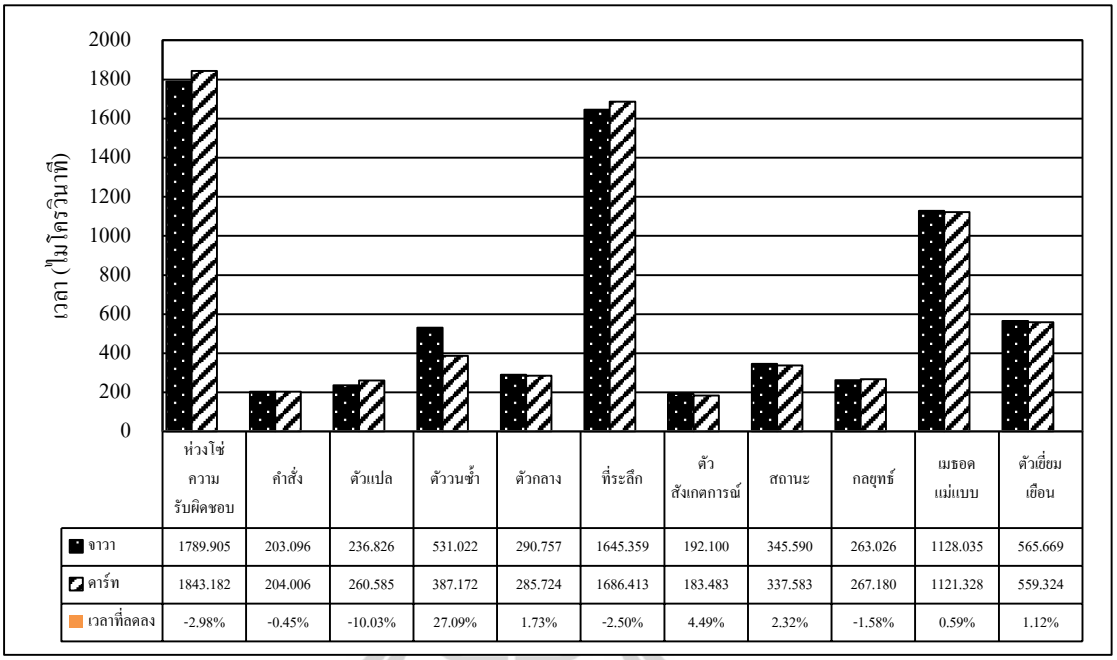


	ตัวแปลง	บริคจ์	เชิงประกอบ	ตัวตกแต่ง	ส่วนหน้า	น้ำหนักเบา	ตัวแทน
จาวา	532.345	170.169	188.392	436.816	1984.787	171071.581	1178.577
ดาร์ท	559.951	173.102	208.106	433.131	1665.698	2022.044	1174.303
เวลาที่ลดลง	-5.19%	-1.72%	-10.46%	0.84%	16.08%	98.82%	0.36%

รูปที่ 4.8 กราฟแสดงเวลาที่ใช้ในการประมวลผลภาษาจาวา และภาษาดาร์ทของแพทเทิร์นเชิงโครงสร้าง

สำหรับแพทเทิร์นเชิงโครงสร้างดังแสดงในรูปที่ 4.8 จะพบว่ามี 3 แพทเทิร์นดาร์ทใช้เวลาในการประมวลผลมากกว่าจาวา คือ แพทเทิร์นตัวแปลง แพทเทิร์นบริจด์ และแพทเทิร์นเชิงประกอบ โดยใช้เวลาในการประมวลผลเพิ่มขึ้น 5.19% 1.72% และ 10.46% ตามลำดับ และอีก 4 แพทเทิร์นที่

ดาร์ทใช้เวลาในการประมวลผลที่น้อยกว่าจาวา คือ แพทเทิร์นตัวตกแต่ง แพทเทิร์นส่วนหน้า แพทเทิร์นน้ำหนักเบา และแพทเทิร์นตัวแทน โดยใช้เวลาในการประมวลผลลดลง 0.84% 16.08% 98.82% และ 0.36% ตามลำดับ



	ห่วงโซ่ความรับผิดชอบ	คำสั่ง	ตัวแปล	ตัววนซ้ำ	ตัวกลาง	ที่ระลึก	ตัวสังเกตการณ์	สถานะ	กลยุทธ์	เมธอดแม่แบบ	ตัวเยี่ยมเยือน
จาวา	1789.905	203.096	236.826	531.022	290.757	1645.359	192.100	345.590	263.026	1128.035	565.669
ดาร์ท	1843.182	204.006	260.585	387.172	285.724	1686.413	183.483	337.583	267.180	1121.328	559.324
เวลาที่ลดลง	-2.98%	-0.45%	-10.03%	27.09%	1.73%	-2.50%	4.49%	2.32%	-1.58%	0.59%	1.12%

รูปที่ 4.9 กราฟแสดงเวลาที่ใช้ในการประมวลผลภาษาจาวา และภาษาดาร์ทของแพทเทิร์นเชิงพฤติกรรม

สำหรับแพทเทิร์นเชิงพฤติกรรม ดังแสดงในรูปที่ 4.9 จะพบว่ามี 5 แพทเทิร์นที่ดาร์ทใช้เวลาในการประมวลผลมากกว่าจาวา คือ แพทเทิร์นห่วงโซ่ความรับผิดชอบ แพทเทิร์นคำสั่ง แพทเทิร์นตัวแปล แพทเทิร์นที่ระลึก และแพทเทิร์นกลยุทธ์ โดยใช้เวลาในการประมวลผลเพิ่มขึ้น 2.98% 0.45% 10.03% 2.05% และ 1.58% ตามลำดับ สำหรับแพทเทิร์นที่ดาร์ทใช้เวลาในการประมวลผลน้อยกว่าจาวามี 6 แพทเทิร์น คือ แพทเทิร์นตัววนซ้ำ แพทเทิร์นตัวกลาง แพทเทิร์นตัวสังเกตการณ์ แพทเทิร์นสถานะ แพทเทิร์นเมธอดแม่แบบ และแพทเทร์นตัวเยี่ยมเยือน โดยใช้เวลาในการประมวลผลลดลง 27.79% 1.73% 4.49% 2.32% 0.59% และ 1.12% ตามลำดับ

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

ในปัจจุบันซอฟต์แวร์มีส่วนช่วยในการสนับสนุนการทำงานของบริษัท องค์กร และผู้ประกอบธุรกิจต่าง ๆ มีการนำความรู้ทางด้านวิศวกรรมศาสตร์มาช่วยในกระบวนการพัฒนาซอฟต์แวร์ เพื่อให้การพัฒนาซอฟต์แวร์มีคุณภาพสูงและลดต้นทุนต่าง ๆ และเพื่อให้ตอบสนองต่อการพัฒนาซอฟต์แวร์ให้มีคุณภาพสูงนั้น ทำให้มีนักพัฒนาภาษาสำหรับการเขียนโปรแกรมใหม่ ๆ ขึ้น ซึ่งในแต่ละภาษาที่ถูกพัฒนาขึ้นนั้นก็จะมีความเหมาะสมในซอฟต์แวร์แต่ละประเภทด้วย การศึกษาภาษาเขียนโปรแกรมที่ได้รับการพัฒนาขึ้นมาใหม่นั้น จึงถือว่ามีส่วนสำคัญในพิจารณาเลือกซอฟต์แวร์ให้เหมาะสมกับการพัฒนา การเปรียบเทียบและการประเมินผลภาษาเขียนโปรแกรมที่พัฒนาขึ้นมาใหม่กับภาษาที่กำลังได้รับความนิยมอยู่ เป็นอีกแนวทางหนึ่งในการศึกษาคุณลักษณะและคุณสมบัติของภาษา

ในงานวิจัยนี้ทำการศึกษาภาษาดาร์ท ซึ่งเขียนโปรแกรมที่ถูกพัฒนาขึ้นมาใหม่ โดยการเปรียบเทียบกับภาษาจาวา ซึ่งเป็นภาษาที่ได้รับความนิยมภาษาหนึ่งในปัจจุบัน โดยงานวิจัยนี้มุ่งเน้นที่จะศึกษาค้นคว้าโดยมีวัตถุประสงค์ของการศึกษาคือ เพื่อศึกษาแนวคิดเบื้องต้นในการเขียนโปรแกรม และไวยากรณ์ต่าง ๆ การศึกษาคุณสมบัติและคุณลักษณะในการเขียนโปรแกรม และการศึกษาประสิทธิภาพของเวลาที่ใช้ในการประมวลผลด้วยชุดทดสอบประสิทธิภาพมาตรฐาน

ขั้นตอนในการดำเนินงานวิจัยนี้แบ่งออกเป็น 3 ส่วน โดยส่วนแรกเป็นการศึกษาดีไซน์แพทเทิร์นของ GoF และศึกษาการเขียนโปรแกรม คุณสมบัติ และคุณลักษณะของภาษาดาร์ท ส่วนที่สองคือ การออกแบบและพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นในภาษาดาร์ทและภาษาจาวา และส่วนสุดท้ายคือ การเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรมเริ่มต้น การประเมินคุณสมบัติและคุณลักษณะของภาษาดาร์ทเปรียบเทียบกับภาษาจาวา รวมถึงการนับจำนวนของบรรทัดคำสั่ง และการทดสอบประสิทธิภาพในส่วนของเวลาที่ใช้ในการประมวลผลตัวอย่างดีไซน์แพทเทิร์นที่ได้ทำการออกแบบและพัฒนา

5.1 สรุปผลการวิจัย

การเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรม ผู้วิจัยได้นำเสนอการเปรียบเทียบในส่วนของการเขียนโปรแกรมอย่างง่าย คำสงวน แบบชนิดของข้อมูล และตัวดำเนินการ จากการศึกษาพบว่าการเขียนโปรแกรมเพื่อแสดงผลอย่างง่ายคำสงวน แบบชนิดของข้อมูล และ

ตัวดำเนินการ ส่วนใหญ่แล้วภาษาดาร์ทจะใกล้เคียงกับภาษาจาวามาก เพราะดาร์ทมีวัตถุประสงค์ในการออกแบบอย่างหนึ่งคือ เพื่อให้ง่ายต่อการเรียนรู้ซึ่งอาจหมายถึงการออกแบบตัวภาษาให้เหมือนกับภาษาที่มีอยู่ก่อนหน้าแล้วอย่างเช่นจาวา บางส่วนมีการใช้คำสงวนที่แตกต่างกัน แต่มีหน้าที่เหมือนกัน เช่น คำสงวน package ในจาวา แต่ดาร์ทใช้คำสงวนว่า library เป็นต้น

ในการศึกษาและประเมินคุณสมบัติและคุณลักษณะของภาษาดาร์ท โดยศึกษาจากตัวอย่างการใช้งานดีไซน์แพทเทิร์นที่พัฒนาด้วยภาษาดาร์ทและภาษาจาวา พบว่าดาร์ทมีความสามารถในการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นได้เทียบเท่ากับจาวา แต่ด้วยคุณลักษณะเฉพาะของดาร์ททำให้ลักษณะของการพัฒนาโค้ดส่วนใหญ่แตกต่างจากจาวา แม้ว่าการพัฒนารหัสต้นฉบับบางส่วนจะพัฒนาได้ยากกว่าภาษาจาวา แต่ในขณะเดียวกันดาร์ทยังสามารถพัฒนาโค้ดบางส่วนได้ง่ายกว่าภาษาจาวามาก ทำให้จำนวนของบรรทัดที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นส่วนใหญ่ลดลงได้เป็นจำนวนมาก นอกจากนี้บางแพทเทิร์นยังสามารถทำการปรับปรุงโครงสร้างของดีไซน์แพทเทิร์น และสามารถพัฒนาให้สามารถนำรหัสต้นฉบับกลับไปใช้ซ้ำได้ง่าย ซึ่งมีการปรับปรุงโครงสร้างของแพทเทิร์นเชิงประกอบ และแพทเทิร์นที่ระลึก สำหรับการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นโดยส่วนใหญ่แล้วดาร์ทใช้จำนวนบรรทัดคำสั่งที่น้อยกว่าจาวา ซึ่งลดลงมากสูงสุดถึง 38 % ในแพทเทิร์นโรงงานเชิงนามธรรม ขณะที่แพทเทิร์นต้นแบบใช้จำนวนบรรทัดเพิ่มขึ้น 20% เมื่อนับรวมคลาสที่พัฒนาขึ้นสำหรับการคัดลอกอ๊อบเจกต์

ในการทดสอบประสิทธิภาพของภาษาดาร์ทโดยเปรียบเทียบกับภาษาจาวาจากรหัสต้นฉบับที่พัฒนาขึ้นมาทั้ง 2 ชุด ซึ่งประกอบด้วยรหัสต้นฉบับชุดที่ 1 พัฒนาด้วยภาษาจาวา รหัสต้นฉบับชุดที่ 2 พัฒนาด้วยภาษาดาร์ท ในการทดสอบพบว่า (1) แพทเทิร์นเชิงการสร้าง สำหรับแพทเทิร์นโรงงานเชิงนามธรรม แพทเทิร์นตัวสร้าง และแพทเทิร์นเดี่ยว ใช้เวลาในการประมวลผลที่ใกล้เคียงกันมาก ในส่วนของแพทเทิร์นเมธอดโรงงานดาร์ทใช้เวลาในการประมวลผลที่น้อยกว่าจาวา โดยเวลาลดลงถึง 19.71% ส่วนแพทเทิร์นต้นแบบดาร์ทใช้เวลาในการประมวลผลมากกว่าจาวา โดยใช้เวลามากกว่าถึง 24.24% เนื่องจากเมธอดสำหรับคัดลอกในดาร์ทได้ถูกพัฒนาขึ้นมาใหม่ ในขณะที่จาวาถูกฝังไว้ในเครื่องจักรเสมือนแล้ว (2) แพทเทิร์นเชิงโครงสร้าง สำหรับแพทเทิร์นบริดจ์ แพทเทิร์นตัวตกแต่ง และแพทเทิร์นตัวแทนใช้เวลาในการประมวลผลที่ใกล้เคียงกันมาก สำหรับแพทเทิร์นตัวแปลง และแพทเทิร์นเชิงประกอบดาร์ทใช้เวลาในการประมวลผลที่มากกว่า 5.19% และ 10.46% ตามลำดับ ส่วนแพทเทิร์นส่วนหน้า และแพทเทิร์นน้ำหนักเบา ดาร์ทใช้เวลาในการประมวลผลที่น้อยกว่า โดยใช้เวลาลดลง 16.08% และ 98.82% ตามลำดับ (3) และแพทเทิร์นเชิงพฤติกรรม สำหรับแพทเทิร์นคำสั่งและแพทเทิร์นเมธอดแม่แบบใช้เวลาในการประมวลผลที่ใกล้เคียงกัน แพทเทิร์นห่วงโซ่ความรับผิดชอบ แพทเทิร์นตัวแปล แพทเทิร์นที่ระลึก และแพทเทิร์น

กลยุทธ์ ดาร์ทใช้เวลาในการประมวลผลมากกว่าจาวา โดยแพทเทิร์นตัวแปลดาร์ทใช้เวลามากกว่าถึง 10.03% แพทเทิร์นตัววนซ้ำ แพทเทิร์นตัวกลาง แพทเทิร์นตัวสังเกตการณ์ แพทเทิร์นสถานะ และแพทเทิร์นตัวเยี่ยมเยือน ดาร์ทใช้เวลาในการประมวลผลที่น้อยกว่าภาษาจาวา โดยแพทเทิร์นตัววนซ้ำใช้เวลาในการประมวลผลน้อยกว่าถึง 27.09 %

การเปรียบเทียบเวลาที่ใช้ในการประมวลผลเพื่อทดสอบประสิทธิภาพของภาษาดาร์ท สามารถสรุปผลการทดสอบออกได้เป็น 3 ส่วนด้วยกัน โดยส่วนแรกเป็นผลการทดสอบเวลาของตัวอย่างการใช้งานดีไซน์แพทเทิร์นที่ภาษาดาร์ทใช้เวลาในการประมวลผลใกล้เคียงกับภาษาจาวามี 5 แพทเทิร์น ประกอบด้วย แพทเทิร์นตัวสร้าง แพทเทิร์นตัวตกแต่ง แพทเทิร์นตัวแทน แพทเทิร์นคำสั่ง และแพทเทิร์นเมธอดแม่แบบ

ผลการทดสอบในส่วนที่สองคือ การทดสอบเวลาของตัวอย่างการใช้งานดีไซน์แพทเทิร์นที่ภาษาดาร์ทใช้เวลาในการประมวลผลน้อยกว่าภาษาจาวามี 10 แพทเทิร์น ประกอบด้วย แพทเทิร์นโรงงานเชิงนามธรรม แพทเทิร์นเมธอดโรงงาน แพทเทิร์นเดี่ยว แพทเทิร์นส่วนหน้า แพทเทิร์นน้ำหนักเบา แพทเทิร์นตัววนซ้ำ แพทเทิร์นตัวกลาง แพทเทิร์นตัวสังเกตการณ์ แพทเทิร์นสถานะ และแพทเทิร์นตัวเยี่ยมเยือน ในส่วนนี้ผลการศึกษาถือว่าเป็นไปตามสมมติฐานที่ได้ตั้งไว้

ผลการทดสอบในส่วนที่สามคือ การทดสอบเวลาของตัวอย่างการใช้งานดีไซน์แพทเทิร์นที่ภาษาดาร์ทใช้เวลาในการประมวลผลมากกว่าภาษาจาวามี 8 แพทเทิร์น ประกอบด้วย แพทเทิร์นต้นแบบ แพทเทิร์นตัวแปลง แพทเทิร์นบริดจ์ แพทเทิร์นเชิงประกอบ แพทเทิร์นห่วงโซ่ความรับผิดชอบ แพทเทิร์นตัวแปล แพทเทิร์นที่ระลึก และแพทเทิร์นกลยุทธ์ ในส่วนสุดท้ายนี้ไม่เป็นไปตามสมมติฐานที่ได้ตั้งไว้ อาจเนื่องมาจากการปรับปรุงโครงสร้างของแพทเทิร์น รวมถึงการพยายามพัฒนาให้สามารถนำรหัสต้นฉบับกลับไปใช้ซ้ำโดยไม่จำเป็นต้องทำการปรับปรุงรหัสต้นฉบับดังกล่าวเพิ่มเติม หรือให้มีการปรับปรุงเมื่อนำไปใช้ให้น้อยที่สุด การปรับปรุงโครงสร้างของแพทเทิร์นให้เหมาะกับภาษาดาร์ทเกิดขึ้นในแพทเทิร์นเชิงประกอบ และแพทเทิร์นที่ระลึก ซึ่งสามารถนำรหัสต้นฉบับจากตัวอย่างการใช้งานกลับไปใช้ซ้ำได้ง่าย แพทเทิร์นที่ไม่ได้มีการปรับปรุงในส่วนของโครงสร้าง แต่ได้พัฒนารหัสต้นฉบับให้สามารถนำไปใช้ซ้ำได้โดยไม่ต้องทำการอิมพลีเมนต์เพิ่มเติม คือ แพทเทิร์นตัววนซ้ำ และแพทเทิร์นน้ำหนักเบา ที่ได้พัฒนาให้สามารถนำไปใช้ซ้ำได้แต่ต้องมีการเปลี่ยนแปลงรหัสต้นฉบับเล็กน้อย ส่วนแพทเทิร์นที่เหลือที่ดาร์ทใช้เวลาในการประมวลผลมากกว่าจาวาอาจเกิดจากการเรียกใช้คลาสที่เกี่ยวข้องกับการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นทำให้ใช้เวลาที่มากขึ้น

จากผลการศึกษาพบว่าดาร์ทมีความสามารถในการพัฒนารหัสต้นฉบับได้เทียบเท่ากับจาวา และด้วยคุณสมบัติและคุณลักษณะเฉพาะของดาร์ททำให้สามารถพัฒนาตัวอย่างการใช้งาน

ดีไซน์แพทเทิร์นได้มีประสิทธิภาพมากกว่าจาวา รวมถึงสามารถทำการปรับปรุงโครงสร้างของดีไซน์แพทเทิร์นให้เหมาะสมกับคุณสมบัติของดาร์ทเพื่อให้สามารถนำรหัสต้นฉบับกลับไปใช้ซ้ำได้อีกด้วย

5.2 ข้อเสนอแนะ

ในการศึกษาการเปรียบเทียบและการประเมินผลภาษาดาร์ทด้วยดีไซน์แพทเทิร์นนี้ เป็นการศึกษาโดยใช้ประสบการณ์ของผู้วิจัย สำหรับการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นทั้ง 23 แพทเทิร์น ซึ่งอาจทำให้ไม่สามารถใช้คุณลักษณะของภาษาได้ครบถ้วนตามคุณลักษณะของภาษาที่นักพัฒนาภาษาที่ได้มีการออกแบบไว้

การศึกษาในวิทยานิพนธ์ฉบับนี้เป็นเพียงวิธีการหนึ่งในการเปรียบเทียบและการประเมินผลของภาษาเขียนโปรแกรมเท่านั้น ผู้วิจัยหวังว่าการศึกษาครั้งนี้จะเป็นแนวทางในการประเมินผลของภาษาเขียนโปรแกรมด้วยวิธีการอื่น ๆ ต่อไป

รายการอ้างอิง

B. Nystrom, (2013, March). **Idiomatic Dart**. Retrieved June 1, 2013, from: http:// www.dartlang.org/articles/idiomatic-dart/

D. Flanagan, (2011). **JavaScript: The Definitive Guide.** 6th ed. Sebastopol, CA: O'Reilly Media, Inc.

E. Gamma, R. Helm, R. Johnson & J. Vlissides, (1995). **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley.

F. Schmager, N. Cameron, and J. Noble, (2010). **GoHotDraw: Evaluating the Go Programming Language with Design Patterns**. In Evaluation and Usability of Programming Languages and Tools (PLATEAU) 2010. Nevada, USA.

G. Bracha , (2013, November). **Mixins in Dart**. Retrieved March 1, 2014, from https:// www.dartlang.org/articles/mixins/

G. Bracha & L. Bak, (2011). **Dart, a new programming language for structured web programming**. GOTO Aarhus conferences, Oct. 2011.

Johnmccutchan, (2014). **Dart Benchmark Harness**. Retrieved June 2, 2014, from https:// github.com/dart-lang/benchmark_harness

K. Walratb & S. Ladd, (2012). **What is Dart?** California: O'Reilly Media, Inc.,.

K. Walrath, S. Ladd, C. Hearse, D. Futato, R. Demarest & R. Comer, (2013a, March 29). **Chapter 1. Quick Start**. Retrieved May 25, 2013, from : http://www.dartlang.org/ docs/dart-up-and-running/contents/ch01.html#ch01-quick-look

K. Walrath, S. Ladd, C. Hearse, D. Futato, R. Demarest, and R. Comer. (2013b). **Technical Overview**. Retrieved June 12, 2012, from: http://www.dartlang.org/docs/technical-overview/

K. Walrath, S. Ladd, C. Hearse, D. Futato, R. Demarest, and R. Comer, (2013c, March 29). **Chapter 2. A Tour of the Dart Language**. Retrieved July 13, 2013, from https://www.dartlang.org/docs/dart-up-and-running/contents/ch02.html

M. R. Nami, (2008). **A Comparison of Object-Oriented Languages in Software Engineering**. ACM SIGSOFT Software Engineering Notes, (pp. 1-5, Vol. 33, No. 4).

N. Botev, (2013). **Dart Benchmark Harness**. Retrieved June 2, 2014, from https://github.com/dart-lang/benchmark_harness

Oracle, (2014a). **Java Language Keywords**. Retrieved March 1, 2014, from: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

Oracle, (2014b). **Primitive Data Types**. Retrieved March 1, 2014, from http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Oracle, (2014c). **Summary of Operators**. Retrieved March 1, 2014, from http://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html

S. Clarke, (2001). **Evaluating a new programming language**. 13th Workshop of the Psychology of Programming Interest Group, (pp. 275-289). Bournemouth UK.

S. Ladd, (2013, March 7). **Classes: Setters & Getters - Dart Tips, Ep 10**. Retrieved July 13, 2013, from http://www.dartlang.org/dart-tips/dart-tips-ep-10.html

S. S. Chandra & K. Chandra, (2005). **A COMPARISON OF JAVA AND C#**. CCSC: Rocky Mountain Conference, (pp. 238-254).

The Dart Team, (2013, June 4). **Language Specification**. Retrieved May 30, 2013, from: http://www.dartlang.org/docs/spec/

The Dart Team, (2014, September). **Dart VM and dart2js Performance**. Retrieved September 22, 2014, from: https://www.dartlang.org/performance/

Z. Hemel, (2013). **Dart2js Outperforms Hand-Written JavaScript in DeltaBlue Benchmark**. Retrieved September 25, 2014, from: http://www.infoq.com/news/2013/04/dart2js-outperforms-js

ภาคผนวก ก

บทความทางวิชาการที่ได้รับการตีพิมพ์เผยแพร่ในระหว่างการศึกษา

รายชื่อบทความที่ได้รับการตีพิมพ์เผยแพร่ในระหว่างการศึกษา

นริศรา ธาระพุทธ และ พิชโยทัย มหัทธนาภิวัฒน์ (2557). **การใช้ดีไซน์แพทเทิร์นเพื่อเปรียบเทียบคุณลักษณะของภาษาดาร์ทและภาษาจาวา**. การประชุมวิชาการระดับชาติ มหาวิทยาลัยราชภัฏนครปฐม ครั้งที่ 6 (The 6th NPRU National Academic Conference 2014)., หน้า 29 - 37

นริศรา ธาระพุทธ, พิชโยทัย มหัทธนาภิวัฒน์ และ ศุภกฤษฎิ์ ตั้งเสริมสิทธิ์ (2557). **การอิมพลีเมนต์ดีไซน์แพทเทิร์นในภาษาดาร์ท**. การประชุมวิชาการ มหาวิทยาลัยนอร์ท-เชียงใหม่ ครั้งที่ 1., หน้า 223 - 230

The 6th NPRU National Academic Conference 2014
Nakhon Pathom Rajabhat University | Nakhon Pathom | Thailand | 30 – 31 May 2014

การใช้ดีไซน์แพทเทิร์นเพื่อเปรียบเทียบคุณลักษณะของภาษาดาร์ทและภาษาจาวา
Using Design Patterns for Comparison of Dart Language and Java Language

นริศรา ธาระพุทธ[1*] และ พิชโยทัย มหัทธนาภิวัฒน์[1]

[1]*สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี*
[*]*n.tharaputh@gmail.com*

บทคัดย่อ

ดีไซน์แพทเทิร์นเป็นการบันทึกประสบการณ์ที่เกี่ยวกับการออกแบบซอฟต์แวร์เชิงวัตถุ ดีไซน์แพทเทิร์นจะถูกใช้อธิบายวิธีการแก้ปัญหาในบริบทหนึ่งๆ และจะถูกอิมพลีเมนต์ด้วยภาษาโปรแกรม ภาษาดาร์ทเป็นภาษาที่ได้รับการพัฒนาขึ้นมาใหม่ สำหรับการพัฒนาโปรแกรมประยุกต์บนเว็บ ซึ่งมีวัตถุประสงค์เพื่อให้สามารถสร้างโปรแกรมประยุกต์บนเว็บที่สามารถทำงานได้เร็วขึ้นและมีเป้าหมายเพื่อแทนที่ภาษาจาวาสคริปต์ ด้วยวัตถุประสงค์และเป้าหมายดังกล่าวอาจทำให้ดาร์ทได้รับความนิยมเป็นอย่างมากในอนาคต บทความนี้ได้ทำการอิมพลีเมนต์ดีไซน์แพทเทิร์นโดยใช้ภาษาดาร์ทและภาษาจาวา แพทเทิร์นเดี่ยว แพทเทิร์นตัวแปลง และแพทเทิร์นตัวสังเกตการณ์ เป็นแพทเทิร์นที่ถูกเลือกสำหรับการเปรียบเทียบคุณลักษณะในเชิงความสามารถและการนับบรรทัดของคำสั่ง (LOC) ผลการศึกษาพบว่าภาษาดาร์ทมีความสามารถในการพัฒนาซอร์สโค้ดได้เทียบเท่ากับภาษาจาวา และด้วยคุณลักษณะเฉพาะของภาษาดาร์ททำให้สามารถพัฒนาซอร์สโค้ดได้ง่ายและจำนวนบรรทัดในการพัฒนาของภาษาดาร์ทมีจำนวนน้อยกว่าภาษาจาวา

คำสำคัญ: การเปรียบเทียบภาษาเขียนโปรแกรม, ภาษาดาร์ท, ดีไซน์แพทเทิร์น

Abstract

Design patterns are the records of experience about object-oriented software design. Design patterns will be used to explain how to resolve a problem in a contexts and they are implemented using programming language. Dart is a new language for developing web applications. The purpose of Dart is to allow web applications to run faster and to replace JavaScript. It is likely that Dart will be popular in the future. In this paper, design patterns are implemented using Dart language and Java language. Singleton, Adapter and Observer patterns are chosen for the comparison of implementation, feature and line of code (LOC). The result is that Dart language has the ability to develop source code equivalent to Java language and idiomatic of Dart language to develop source code is easier. Lines of code in a development with Dart language are less than that of Java language.

Keywords: comparison programing language, dart language, design patterns

1. บทนำ

ภาษาดาร์ท (Dart) เป็นภาษาใหม่ที่ได้รับการพัฒนาโดยทีมพัฒนาของกูเกิล (Google) และได้รับการเปิดตัวเมื่อเดือนตุลาคม ค.ศ. 2011 (Bracha & Bak, 2011) ภาษาดาร์ทถูกพัฒนาขึ้นสำหรับการพัฒนาโปรแกรมประยุกต์บนเว็บ (Web Application) ซึ่งมีวัตถุประสงค์เพื่อให้การพัฒนาเว็บสามารถทำได้รวดเร็วและพัฒนาได้ง่ายขึ้น ทำให้ภาษาดาร์ทเป็นภาษาที่ได้รับความสนใจเป็นอย่างมากในวงการพัฒนาโปรแกรมประยุกต์บนเว็บ (Walrath & Ladd, 2012) ในขณะที่นักพัฒนาเชื่อว่าภาษาดาร์ทจะสามารถแทนที่ข้อจำกัดบางอย่างของภาษาจาวาสคริปต์ (JavaScript) ได้ (Miller, 2010) ภาษาดาร์ทถูก

ออกแบบให้มีวากยสัมพันธ์ (Syntax) คล้ายกับภาษาอื่นโดยเฉพาะภาษาจาวา (Java) และภาษาจาวาสคริปต์ ซึ่งเป็นอีกหนึ่งเหตุผลที่ทำให้ภาษาดาร์ทเป็นภาษาที่ง่ายต่อการเรียนรู้ (Nystrom, 2013)

งานวิจัยที่เกี่ยวข้องกับการเปรียบเทียบและการประเมินผลของภาษาเขียนโปรแกรมมีเป็นจำนวนมาก เช่น การประเมินผลภาษาเขียนโปรแกรมด้วยการเปรียบเทียบข้อมูลที่ได้จากห้องปฏิบัติการในการพัฒนาซอร์สโค้ดตามที่กำหนดไว้และแบบสอบถามที่เกี่ยวข้องกับการพัฒนาซอร์สโค้ดดังกล่าว โดยภาษาที่ใช้ในการศึกษาคือ ภาษาซีชาร์ป (C#) (Clarke, 2001) วิธีการดังกล่าวอาจมีความคลาดเคลื่อนได้เนื่องมาจากประสบการณ์และความชำนาญของผู้ทดสอบ การเปรียบเทียบแนวคิดเบื้องต้นในการเขียนโปรแกรมระหว่างภาษาจาวาและภาษาซีชาร์ป โดยการเปรียบเทียบนี้เน้นการเขียนโปรแกรมอย่างง่าย การเปรียบเทียบคำสงวน (Keyword) ตัวดำเนินการ เป็นต้น (Chandra & Chandra, 2005) วิธีการนี้เป็นเพียงการศึกษาถึงวากยสัมพันธ์เบื้องต้นเท่านั้น การประเมินผลภาษาโก (Go) ด้วยดีไซน์แพทเทิร์น (Design patterns) และพัฒนาเฟรมเวิร์ค HotDraw เพื่อศึกษาถึงความสามารถของภาษาโดยการเลือกพัฒนาเฉพาะส่วน (Schmager et al., 2010) สำหรับวิธีการนี้เป็นการศึกษาถึงดีไซน์แพทเทิร์นที่ปรากฏอยู่ในภาษาและความสามารถของภาษาในการพัฒนาซอร์สโค้ดตามดีไซน์แพทเทิร์น จากงานวิจัยที่ได้กล่าวมาในข้างต้น ผู้วิจัยจึงได้เลือกใช้ดีไซน์แพทเทิร์นในการเปรียบเทียบคุณลักษณะของภาษา เพราะนอกจากจะสามารถศึกษาถึงวากยสัมพันธ์ในการพัฒนาซอร์สโค้ดของภาษาดาร์ทเมื่อเทียบกับภาษาจาวาแล้ว ยังสามารถศึกษาถึงความสามารถในการอิมพลีเมนต์ดีไซน์แพทเทิร์นของภาษาดาร์ทได้อีกด้วย

บทความนี้ผู้วิจัยจึงเลือกใช้ดีไซน์แพทเทิร์นของ Gang of Four (GoF) (Gamma et al., 1995) ในการศึกษาเพื่อเปรียบเทียบคุณลักษณะของภาษาดาร์ทและภาษาจาวา เพราะนอกจากจะสามารถเปรียบเทียบคุณลักษณะเบื้องต้นได้แล้ว ยังสามารถศึกษาถึงดีไซน์แพทเทิร์นที่ปรากฏอยู่ในภาษาเขียนโปรแกรมได้อีกด้วย ในการศึกษาผู้วิจัยจะอธิบายตัวอย่างการใช้งานของดีไซน์แพทเทิร์นจากประสบการณ์และอภิปรายถึงความแตกต่างของวากยสัมพันธ์ในการพัฒนาตัวอย่างการใช้งานในภาษาดาร์ท เมื่อเปรียบเทียบกับภาษาจาวารวมถึงการนับจำนวนของบรรทัดคำสั่ง ซึ่งในภาษาจาวาการพัฒนาตัวอย่างการใช้งานในครั้งนี้จะไม่มีการใช้ @Override เพื่อลดจำนวนบรรทัด แพทเทิร์นที่เลือกสำหรับการศึกษาคือ แพทเทิร์นเดี่ยว (Singleton pattern) แสดงถึงการสร้างอ็อบเจกต์ แพทเทิร์นตัวแปลง (Adapter pattern) แสดงถึงความสัมพันธ์ของอ็อบเจกต์ และแพทเทิร์นตัวสังเกตการณ์ (Observer pattern) ที่แสดงถึงการติดต่อสื่อสารกันระหว่างอ็อบเจกต์

2. ทฤษฎีที่เกี่ยวข้อง

2.1 ภาษาดาร์ท

ภาษาดาร์ทเป็นภาษาเขียนโปรแกรมเชิงวัตถุ ที่มีลักษณะของการสืบทอดเป็นการสืบทอดแบบเดี่ยว (Single-inheritance) เป็นภาษาเขียนโปรแกรมที่มีแบบชนิดเชิงทางเลือก (Optional types) และสนับสนุนแบบชนิดทั่วไป (General types) โดยโปรแกรมดาร์ทจะมีการกระทำการ (Execute) อยู่ 2 แบบวิธี ซึ่งจะเลือกกระทำการเพียงหนึ่งแบบวิธี (The Dart Team, 2013a) คือ 1) แบบวิธีการผลิต (Production mode) เป็นแบบวิธีที่ถูกกำหนดให้เป็นแบบวิธีเริ่มต้นของการนำไปใช้งาน ซึ่งในแบบวิธีนี้จะไม่สนใจแบบชนิดคงที่ (Static types) และคำสั่ง assert และ 2) แบบวิธีการตรวจสอบ (Checked mode) เป็นแบบวิธีสำหรับการพัฒนาและการดักจับข้อผิดพลาด ซึ่งจะช่วยให้สามารถจัดการกับข้อผิดพลาดบางอย่างในช่วงระยะเวลาดำเนินการ (Run-time) (Walrath et al., 2013)

2.2 ดีไซน์แพทเทิร์น

ดีไซน์แพทเทิร์นจะถูกนำมาใช้สำหรับการแก้ไขปัญหาที่เกิดขึ้นในการออกแบบโปรแกรมเชิงวัตถุเพื่อให้สามารถนำซอร์สโค้ดกลับมาใช้ซ้ำได้ โดยดีไซน์แพทเทิร์นเป็นการบันทึกประสบการณ์ที่เกี่ยวข้องกับการออกแบบเพื่อให้สามารถพัฒนาโปรแกรมได้อย่างมีประสิทธิภาพ ดีไซน์แพทเทิร์นของ GoF (Gamma and et al., 1995) เป็นดีไซน์แพทเทิร์นที่ได้รับความนิยมในวงการพัฒนาซอฟต์แวร์ มีหนังสือเป็นจำนวนมากใช้ภาษาเขียนโปรแกรมในการอธิบายดีไซน์แพทเทิร์นของ GoF คือ ภาษาซีชาร์ป (Bishop, 2008) ภาษาจาวา (Metsker & Wake, 2006) ภาษาจาวาสคริปต์ (Harmes & Diaz, 2007) ภาษารูบี (Ruby) (Olsen, 2007) และภาษาสมอลล์ทอร์ก (Smalltalk) (Alpert and et al., 1998) สำหรับการอธิบายดีไซน์แพทเทิร์นของ GoF ด้วยความจำกัดของพื้นที่ผู้วิจัยจะนำเสนอเพียงวัตถุประสงค์ของแพทเทิร์นที่ได้นำมาศึกษาดังนี้

1) แพทเทิร์นเดี่ยว มีวัตถุประสงค์เพื่อตรวจสอบให้แน่ใจว่าคลาสจะมีเพียงหนึ่งอินสแตนซ์เท่านั้น และเข้าถึงอินสแตนซ์นั้นโดยใช้เมธอดที่สามารถเข้าถึงจากทุกคลาส (public method)

2) แพทเทิร์นตัวแปลง มีวัตถุประสงค์เพื่อช่วยให้อินเตอร์เฟซที่ไม่เข้ากันสามารถทำงานร่วมกันได้ โดยการแปลง (Convert) อินเตอร์เฟซของคลาสไปยังอินเตอร์เฟซอื่นๆ ที่คาดหวัง

3) แพทเทิร์นตัวสังเกตการณ์ มีวัตถุประสงค์เพื่อให้อ็อบเจกต์หนึ่งเมื่อมีการเปลี่ยนแปลงแล้วอ็อบเจกต์ที่มีการพึ่งพากันจะทำการอัพเดตหรือเปลี่ยนแปลงข้อมูลอัตโนมัติ โดยจำเป็นจะต้องกำหนดการพึ่งพากันของอ็อบเจกต์แบบหนึ่งต่อหลาย (One-to-many)

3. วิธีการศึกษา

การดำเนินงานวิจัยในครั้งนี้ผู้วิจัยได้แบ่งวิธีการศึกษาออกเป็น 3 ส่วน โดยมีรายละเอียดของการดำเนินการดังนี้

3.1 การศึกษาดีไซน์แพทเทิร์น

ดีไซน์แพทเทิร์นที่ได้ที่ผู้วิจัยเลือกสำหรับนำมาศึกษาในครั้งนี้เป็นดีไซน์แพทเทิร์นของ GoF (Gamma et al., 1995) ที่ได้ทำการเก็บรวบรวมลักษณะของการเขียนโปรแกรมออกมาเป็นแพทเทิร์นทั้งหมด 23 แพทเทิร์น โดยเลือกแพทเทิร์นสำหรับศึกษามาจากการจัดกลุ่มตามวัตถุประสงค์ คือแพทเทิร์นเดี่ยวที่แสดงถึงการสร้างอ็อบเจกต์ แพทเทิร์นตัวแปลงที่แสดงถึงลักษณะของโครงสร้าง และแพทเทิร์นตัวสังเกตการณ์ที่แสดงถึงลักษณะของพฤติกรรม

3.2 การศึกษาภาษาดาร์ท

คุณสมบัติและคุณลักษณะของภาษาดาร์ทที่น่าสนใจมีจำนวนมาก (Nystrom, 2013 ; The Dart Team, 2013b) ด้วยความจำกัดของพื้นที่ในการนำเสนอจึงเลือกมาเฉพาะคุณสมบัติและคุณลักษณะที่เกี่ยวข้องกับแพทเทิร์นที่ทำการศึกษาคือ

3.2.1 คลาส อ็อบเจกต์ และคอนสตรัคเตอร์

ในการสร้างคลาสของภาษาดาร์ทนั้นมีลักษณะเหมือนกับการสร้างคลาสในภาษาจาวา คือ มีการใช้คำสงวน class ตามด้วย ชื่อคลาส และสัญลักษณ์ { } ตัวอย่างเช่น class Car {...} ภาษาดาร์ทถูกออกแบบให้เป็นภาษาเชิงวัตถุ ทำให้ทุกอย่างถูกมองเป็นอ็อบเจกต์ และทุกอ็อบเจกต์ถูกกำหนดให้สืบทอดมาจากคลาส Object ซึ่งเป็นคลังข้อมูลที่มีอยู่ในตัว (Build-in library) ส่วนคอนสตรัคเตอร์ (Constructor) ในภาษาดาร์ทจะแตกต่างจากในภาษาจาวา โดยภาษาดาร์ทมีแนวคิดของคอนสตรัคเตอร์ที่น่าสนใจอยู่ 3 แนวคิดคือ 1) การเริ่มต้นเขตข้อมูลอัตโนมัติ 2) การตั้งชื่อให้กับคอนสตรัคเตอร์ เนื่องจากภาษาดาร์ทไม่สนับสนุนการโอเวอร์โหลดดิง (Overloading) และ 3) **แฟกทอรีคอนสตรัคเตอร์** (Factory constructor) เป็นคอนสตรัคเตอร์ที่สามารถส่งค่ากลับได้ ตัวอย่างคอนสตรัคเตอร์ทั้ง 3 แนวคิด แสดงในตารางที่ 1

ตารางที่ 1 ตัวอย่างแนวคิดของคอนสตรัคเตอร์ทั้ง 3 แนวคิด

การเริ่มต้นเขตข้อมูลอัตโนมัติ	การตั้งชื่อให้กับคอนสตรัคเตอร์	แฟกทอรีคอนสตรัคเตอร์

class Car{
  var x;
  Car(this.x);
}
	class Car{
	  var x;
	  Car(this.x);
	  Car.sport(){...}
	}
		class Car{
		  var x;
		  factory Car(this. x)
		  => new Car.sport();
		  Car.sport(){...}
		}

3.2.2 คลาสนามธรรม

คลาสนามธรรมเป็นตัวช่วยในการกำหนดอินเตอร์เฟซ โดยส่วนใหญ่การสร้างคลาสนามธรรมจะพบการสร้างเมธอดนามธรรมด้วย การนำคลาสนามธรรมไปใช้สามารถทำได้โดยการ implements หรือ extends ปกติคลาสนามธรรมในภาษาดาร์ทจะไม่สามารถสร้างอ็อบเจกต์ได้ แต่ถ้าต้องการสร้างอ็อบเจกต์สามารถทำได้โดยการใช้ลักษณะของแฟกทอรีคอนสตรัคเตอร์

3.2.3 ฟังก์ชัน

ภาษาดาร์ทมีคุณสมบัติเป็นแบบเฟิร์สคลาสฟังก์ชัน (First-class function) ในการสร้างฟังก์ชันสามารถสร้างได้จาก 3 สัญลักษณ์ คือ ฟังก์ชันระบุชื่อ ฟังก์ชันที่ไม่ระบุชื่อ และฟังก์ชันลูกศรหรือเมธอดบรรทัดเดียว (One-line method) และภาษาดาร์ทยังมีลักษณะของฟังก์ชันภายใน (Inner function)

3.3 การพัฒนาตัวอย่างดีไซน์แพทเทิร์นด้วยภาษาจาวาและภาษาดาร์ท

เมื่อได้ทำการศึกษาดีไซน์แพทเทิร์นของ GoF ผู้วิจัยได้ทำการเลือกแพทเทิร์นสำหรับการศึกษาในครั้งนี้ 3 แพทเทิร์น คือ 1) แพทเทิร์นเดี่ยว 2) แพทเทิร์นตัวแปลง และ 3) แพทเทิร์นตัวสังเกตการณ์ เมื่อได้แพทเทิร์นสำหรับการศึกษาผู้วิจัยได้ทำการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นด้วยภาษาจาวา แล้วจึงได้พัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นด้วยภาษาดาร์ทโดยใช้คุณลักษณะเฉพาะของภาษาซึ่งต้องคำนึงถึงผลลัพธ์ที่ได้ให้เหมือนกับดีไซน์แพทเทิร์นที่พัฒนาด้วยภาษาจาวา

4. ผลการศึกษา

ผลการดำเนินงานวิจัยในครั้งนี้ผู้วิจัยได้ผลการศึกษาออกเป็นแพทเทิร์น 3 แพทเทิร์น โดยมีรายละเอียดดังนี้

4.1. แพทเทิร์นเดี่ยว

จากตารางที่ 2 พิจารณาคลาส Singleton ในจาวามีการกำหนดคอนสตรัคเตอร์ 1 คอนสตรัคเตอร์ โดยกำหนดให้เป็น private เพื่อไม่ให้สามารถเข้าถึงได้จากคลาสอื่น และใช้คลาสซ้อนคลาสโดยมีการระบุคำสงวน static (static nested class) สำหรับการสร้างอินสแตนซ์ของคลาส Singleton และกำหนดให้เก็บไว้ในตัวแปร INSTANCE ที่กำหนดไว้เป็นตัวแปรแบบ private และ final และมีเมธอด getInstance() สำหรับการส่งค่าตัวแปร INSTANCE กลับเมื่อมีการเรียกใช้

ในดาร์ทคลาส Singleton มีการกำหนดคอนสตรัคเตอร์ 2 คอนสตรัคเตอร์ คือ Singleton._inner() และ Singleton() สำหรับ Singleton._inner() เป็นคอนสตรัคเตอร์ที่มีไว้สำหรับสร้างอ็อบเจกต์เพื่อเก็บไว้ในตัวแปร *_INSTANCE* ซึ่งเป็นตัวแปรแบบ private และกำหนดให้เป็น final เช่นเดียวกับจาวา (ดาร์ทไม่มีการกำหนดคำสงวน “private” การกำหนดให้เป็น private ทำได้โดยการใส่เครื่องหมาย “_” ไว้ด้านหน้า) และ Singleton() ที่มีการส่งค่าจากตัวแปร *_INSTANCE* กลับ เมื่อมีการสร้างอินสแตนซ์ของคลาส Singleton ซึ่งจากโค้ดตัวอย่างเป็นการเขียนแบบลดรูปคอนสตรัคเตอร์ที่มีการส่งค่ากลับเพียงหนึ่งค่า ด้วยคุณสมบัติของฟังก์ชันลูกศร

พิจารณาเมธอด main() ที่มีการเรียกใช้คลาส Singleton ในจาวาการเรียกใช้อินสแตนซ์ของคลาส Singleton ทำได้โดยการเรียกผ่าน getInstance()เท่านั้น ไม่สามารถทำการ new อ็อบเจกต์ของคลาส Singleton ได้ ในทางตรงกันข้ามดาร์ทสามารถเรียกใช้ อินสแตนซ์ของคลาส Singleton ได้ด้วยการ new อ็อบเจกต์

ตารางที่ 2 ตัวอย่างการใช้งานของแพทเทิร์นเดี่ยวที่พัฒนาด้วยภาษาจาวาและภาษาดาร์ท

จาวา		ดาร์ท	
1	package Singleton;	1	library Singleton;
2	class Singleton {	2	class Singleton {
3	private Singleton(){}	3	Singleton._inner();
4	private static class SingleHolder {		
5	private static final Singleton INSTANCE = new Singleton();	4	static final Singleton *_INSTANCE* = new Singleton._inner();
6	}		
7	public static Singleton getInstance() {	5	factory Singleton() => *_INSTANCE*;
8	return SingleHolder.INSTANCE;	6	}
9	}		
10	}		
11	class Client {		
12	public static void main(String[] args){	7	void main() {
13	Singleton s = Singleton.getInstance();	8	Singleton s = new Singleton();
14	}	9	}
15	}		

4.2 แพทเทิร์นตัวแปลง

จากตารางที่ 3 พิจารณา Shapes ในจาวาถูกกำหนดให้เป็น interface ภายในอินเตอร์เฟซมีเพียงเมธอด (draw) แต่ในดาร์ท Shapes ถูกกำหนดให้เป็น abstract class เนื่องจากในดาร์ทไม่มี interface คลาสนามธรรม Shapes ของดาร์ทในบรรทัดที่ 2 จะพบว่าคลาสนามธรรมมีการ implements คลาส 2 คลาส คือ คลาส LegacyRectangles และ LegacyCircles ซึ่งภายในของทั้ง 2 คลาส มีเมธอด (draw) แต่ภายในตัว (Body) ของคลาสนามธรรม Shapes ไม่ปรากฏเมธอดใดๆ เมื่อคลาส RectangleAdept และคลาส CircleAdept ได้ทำการ implements คลาสนามธรรม Shapes จะสามารถทำการโอเวอร์ไรด์ (Override) เมธอด (draw) ได้ แต่ในกรณีที่คลาส LegacyRectangles และคลาส LegacyCircles มีชื่อของเมธอดภายในที่แตกต่างกันจะไม่สามารถใช้ลักษณะดังกล่าวได้ คลาสนามธรรมของดาร์ทสามารถนำไป extends ได้เช่นเดียวกับคลาสนามธรรมในจาวา และจากตัวอย่างคลาสนามธรรม Shapes ในดาร์ทแสดงให้เห็นว่า คลาสนามธรรมสามารถทำการ implements คลาสทั่วไปได้ ในขณะที่จาวาไม่สามารถทำเช่นนั้นได้

พิจารณา (main) สำหรับตัวแปร shapes ในจาวามีการประกาศตัวแปรเป็นแบบอะเรย์ (Array) ซึ่งมีแบบชนิดเป็น Shapes ส่วนในดาร์ทไม่มีอะเรย์แต่มีลิสต์ (List) ซึ่งเป็นแบบชนิดที่มีอยู่ในตัว (Built-in type) และมีคุณสมบัติเหมือนกับอะเรย์ในจาวา

ตารางที่ 3 ตัวอย่างการใช้งานของแพทเทิร์นตัวแปลงที่พัฒนาด้วยภาษาจาวาและภาษาดาร์ท

จาวา		ดาร์ท	
1	package Adapter;	1	library adapter;
2	interface Shapes {	2	abstract class Shapes implements LegacyRectangles, LegacyCircles {}
3	void draw();		
4	}		
5	class LegacyRectangle {	3	class LegacyRectangles {
6	public void draw(){	4	void draw() => print("Drawing Rectangle");
7	System.out.println("Drawing Rectangle");	5	}
8	}		
9	}		
10	class LegacyCircle {	6	class LegacyCircles {
11	public void draw(){	7	void draw() => print("Drawing Circle");
12	System.out.println("Drawing Circle");	8	}
13	}		
14	}		
15	class RectangleAdept implements Shapes {	9	class RectangleAdept implements Shapes {
16	private LegacyRectangle adaptee = new LegacyRectangle();	10	LegacyRectangles _adaptee = new LegacyRectangles();
17	public void draw() {	11	void draw() => _adaptee.draw ();
18	adaptee.draw();	12	}
19	}		
20	}		
21	class CircleAdept implements Shapes {	13	class CircleAdept implements Shapes {
22	private LegacyCircle adaptee = new LegacyCircle();	14	LegacyCircles _adaptee = new LegacyCircles();
		15	void draw() => _adaptee.draw ();
23	public void draw() {	16	}
24	adaptee.draw();		
25	}		
26	}		
27	public class Client {	17	void main() {
28	public static void main(String[] args){	18	List<Shapes> shapes = [new RectangleAdept()
29	Shapes[] shapes = {new RectangleAdept()		, new CircleAdept()];

จาวา		ดาร์ท	
	, new CircleAdept());	19	shapes.forEach((Shapes shape)
30	for(Shapes shape : shapes)		{shape.draw ();});
31	shape.draw();	20	}
32	}		
33	}		

4.3 แพทเทิร์นตัวสังเกตการณ์

จากตารางที่ 4 สำหรับจาวาเมื่อพิจารณาคลาส BinObs มีการประกาศตัวแปร convertObsrv สำหรับเก็บค่าอินสแตนซ์ของคลาสซ้อนคลาสที่ไม่ระบุชื่อ (Anonymous inner class) ของอินเตอร์เฟซ Observer เพื่อทำการโอเวอร์ไรด์เมธอด update (Observable obs, Object ob) สำหรับเปลี่ยนแปลงข้อมูลเมื่อได้รับการแจ้งเตือน ส่วนในดาร์ทเมื่อพิจารณาฟังก์ชัน binObs(int _dex) ซึ่งมีค่าเทียบเท่ากับคลาส BinObs ในจาวา ภายในฟังก์ชัน binObs(int _dex)มีการส่งค่ากลับเป็นฟังก์ชัน ซึ่งก็คือฟังก์ชันไม่ระบุชื่อในบรรทัดที่ 3 ของดาร์ทและมีค่าเทียบเท่ากับเมธอด update(...) ในจาวา

พิจารณาคลาส Observable สำหรับตัวแปร observer ที่ใช้ในการเก็บค่าสำหรับการเปลี่ยนแปลงข้อมูลเมื่อได้รับการแจ้งเตือน ในจาวาใช้คลาส Vector เป็นแบบชนิดของตัวแปร ซึ่งจะต้องทำการเก็บค่าของ observer ไว้ในตัวแปรที่เป็นอะเรย์ก่อนจะสามารถนำไปใช้ได้ (บรรทัดที่ 29) ส่วนในดาร์ทใช้แบบชนิดที่เป็น List<Function> เมื่อจะนำไปใช้จะต้องทำเก็บค่าไว้ในตัวแปรที่มีแบบชนิดเป็น Function (บรรทัดที่ 13)

ตารางที่ 4 ตัวอย่างการใช้งานของแพทเทิร์นตัวสังเกตการณ์ที่พัฒนาด้วยภาษาจาวาและภาษาดาร์ท

จาวา		ดาร์ท	
1	package observer;	1	library Observer;
2	import java.util.Vector;		
3	interface Observer {		
4	void update(Observable o, Object arg);		
5	}		
6	class BinObs {	2	binObs(int _dex) => (obs,ob)
7	private int dex;	3	=>print({"Bin:${_dex.toRadixString(2)}"});
8	private Observer cvObsrv = new Observer() {		
9	public void update(Observable obs,Object ob){		
10	System.out.println("Bin : " + Integer.toBinaryString(dex));		
11	}		
12	};		
13	BinObs(int dex) {		
14	this.dex = dex;		
15	}		
16	public Observer convrtObs() {		
17	return this.cvObsrv;		
18	}		
19	}		
20	class Observable {		
21	Vector observer = new Vector();	4	class Observable{
22	public void addObs(Observer o) {	5	List<Function> observer = [];
23	observer.addElement(o);	6	void addObs(Function obs) => this.observer.add(obs);
24	}		
25	public void deleteObs() {	7	void deleteObs(){
26	observer.removeAllElements();	8	for(int i = observer.length -1 ;i>=0;i--)

จาวา		ดาร์ท	
27	}	9	this.observer.remove(observer[i]);
28	public void notifyObs() {	10	}
29	Object[] obs = observer.toArray();	11	void notifyObs() {
30	for (int i = obs.length - 1; i >= 0; i--)	12	for(int i = observer.length-1;i>=0;i--){
31	((Observer) obs[i]).update(this, null);	13	Function fObserver = this.observer[i];
32	}	14	fObserver(this, null);
33	}	15	}
34	class Subject extends Observable {	16	}
35	private boolean isOpen;	17	}
36	public Subject() {	18	class Subject extends Observable{
37	this.isOpen = false;	19	bool _isOpen;
38	}	20	Subject() {
39	public void notifyObs() {	21	this._isOpen = false;
40	if (isOpen)	22	}
41	super.notifyObs();	23	void notifyObs() {
42	}	24	if(_isOpen)
43	public void convert() {	25	super.notifyObs();
44	this.isOpen = true;	26	}
45	this.notifyObs();	27	void convert() {
46	}	28	this._isOpen = true;
47	}	29	this.notifyObs();
48	class Client {	30	}
49	private int dex = 14;	31	}
50	Subject sj = new Subject();	32	class Client {
51	public void test() {	33	int _dex = 15;
52	BinObs bin = new BinObs(dex);	34	Subject sj = new Subject();
53	System.out.println("Input Dex: " + dex);	33	void test(){
54	sj.addObs(bin.convrtObs());	34	Function bObs = binObs(_dex);
55	sj.convert();	35	print("Input Dex : $_dex");
56	sj.deleteObs();	36	sj.addObs(bObs);
57	}	37	sj.convert();
58	public static void main(String[] args) {	38	sj.deleteObs();
59	new Client().test();	39	}
60	}	40	}
61	}	41	void main() => new Client().test();

จากการศึกษาดีไซน์แพทเทิร์นทั้ง 3 แพทเทิร์น ทำให้สามารถสรุปความแตกต่างระหว่างภาษาจาวาและภาษาดาร์ท ที่พบในระหว่างการศึกษาได้ดังแสดงในตารางที่ 5 โดยสามารถจำแนกออกได้ 3 ลักษณะ คือ

1) คอนสตรัคเตอร์ของดาร์ทหากภายในตัวคอนสตรัคเตอร์ไม่มีคำสั่งใดๆ สามารถใช้สัญลักษณ์ ; แทนการใช้สัญลักษณ์ { } ในการเริ่มต้นค่าของตัวแปรสามารถเขียนไว้ในพารามิเตอร์ของคอนสตรัคเตอร์ได้ เช่น BinObs (this.dex); แทนการเขียน BinObs (int dex) { this.dex = dex; } แต่ดาร์ทไม่สนับสนุนการโอเวอร์โหลดดิงทำให้ไม่สามารถสร้างคอนสตรัคเตอร์มากกว่า 1 คอนสรัคเตอร์ได้ การจะสร้างมากกว่า 1 คอนสรัคเตอร์นั้นจึงจำเป็นต้องมีการตั้งชื่อให้กับคอนสรัคเตอร์ เช่น BinObs.first ();

2) การสร้างเมธอดในจาวาเมธอด main จำเป็นต้องสร้างไว้ภายในคลาส มีการกำหนดชนิดของเมธอดและพารามิเตอร์ที่ใช้รับค่า แต่ดาร์ทจะต้องสร้างเมธอด main ไว้ภายนอกคลาสซึ่งไม่ต้องกำหนดชนิดของเมธอด และไม่จำเป็นต้องกำหนดพารามิเตอร์ให้กับเมธอด main การเขียนเมธอดทั่วไปที่มีการส่งค่ากลับหรือการพิมพ์ค่าเพื่อแสดงผล ดาร์ทสามารถใช้คุณสมบัติของฟังก์ชันลูกศรในการเขียนเมธอดได้ โดยการส่งค่ากลับสามารถเขียนได้โดยไม่ต้องใช้คำสงวน

return เช่น String draw () => "Drawing Rectangle"; แทนการเขียนด้วย String draw () { return "Drawing Rectangle"; }

3) วากยสัมพันธ์ทั่วไปจากการศึกษาสามารถสรุปได้ดังนี้

3.1) ดาร์ทไม่มีการใช้คำสงวน interface เหมือนในจาวาแต่จะใช้ abstract class แทน ซึ่งสามารถกำหนดให้เป็นอินเตอร์เฟซได้โดยการใช้คำสงวน implements และให้เป็นคลาสนามธรรมสำหรับการสืบทอดโดยการใช้คำสงวน extends ทั้งนี้ขึ้นอยู่กับลักษณะของการใช้งาน

3.2) ลูป for จาวาสามารถเขียนให้อยู่ในลักษณะ for (Shapes shape : shapes) { shape.draw(); } ซึ่งดาร์ทสามารถเขียนในลักษณะ for (Shapes shape in shapes) { shape.draw(); } ได้เช่นกัน และยังสามารถเขียนให้อยู่ในลักษณะของ forEach คือ shapes.forEach ((Shapes shape) { shape.draw (); }); ได้อีกด้วย

3.3) ในจาวาจะรวมคลาสไว้ใน package ส่วนในดาร์ทจะรวมไว้ใน library

3.4) ดาร์ทมีการกำหนดการเข้าถึงเพียง 2 ลักษณะคือ private และ public (ไม่มี protected เหมือนกับจาวา) การกำหนดการเข้าถึงโดยใช้ private จะใช้สัญลักษณ์ _ (Underscore) เช่น int _dex; การเข้าถึงที่ไม่ได้เป็น private จะเป็น public ทั้งหมดโดยไม่ต้องระบุ public กำกับไว้

3.5) ดาร์ทใช้ลิสต์ในการเก็บชุดของตัวแปรที่เป็นชนิดเดียวกัน ซึ่งมีคุณสมบัติคล้ายกับอะเรย์ในจาวา

3.6) การพิมพ์ค่าออกทางหน้าจอดาร์ทสามารถเขียนได้สั้นกว่า คือ print("Input Dex : $_dex"); ในขณะที่จาวาเขียนยาวกว่า คือ System.out.println("Input Dex: " + dex);

ตารางที่ 5 สรุปความแตกต่างระหว่างภาษาจาวาและภาษาดาร์ทที่พบใน 3 แพทเทิร์น

จาวา	ดาร์ท
คอนสตรัคเตอร์	
BinObs () {} BinObs (int dex) { this.dex = dex; }	BinObs.first (); BinObs (this.dex);
เมธอด	

class Main {
public static void main(String[] args){}
}
	void main(){}

public String draw(){
return "Drawing Rectangle";
}
	String draw () => "Drawing Rectangle";

public void draw(){
System.out.println("Drawing Rectangle");
}
	void draw () => print("Drawing Rectangle");

วากยสัมพันธ์ทั่วไป	
interface	abstract class
for(Shapes shape : shapes) shape.draw();	shapes.forEach((Shapes shape){shape.draw ();});
package	library
private	_ (underscore)
public	ถ้าไม่มีการระบุว่าเป็น private จะเป็น public ทั้งหมด
Shapes[] shapes = {};	List<Shapes> shapes = [];
System.out.println("Input Dex: " + dex);	print("Input Dex : $_dex");

5. สรุปผลการศึกษา

บทความนี้มีการแนะนำและการเปรียบเทียบคุณลักษณะของภาษาดาร์ท โดยเราได้ทำการพัฒนาตัวอย่างการใช้งานดีไซน์แพทเทิร์นเฉพาะแพทเทิร์นเดี่ยว แพทเทิร์นตัวแปลง และแพทเทิร์นตัวสังเกตการณ์ เน้นการเปรียบเทียบคุณลักษณะในด้านความสามารถของภาษาในการพัฒนาและการนับจำนวนบรรทัดคำสั่ง จากการพัฒนาตัวอย่างการใช้งานพบว่าภาษาดาร์ทมีความสามารถในการพัฒนาซอร์สโค้ดให้ผลลัพธ์ที่ได้เหมือนกับภาษาจาวา แต่ด้วยคุณลักษณะเฉพาะของดาร์ททำให้ลักษณะของการพัฒนาโค้ดส่วนใหญ่แตกต่างจากจาวา และดาร์ทยังสามารถพัฒนาโค้ดบางส่วนได้ง่ายกว่าภาษาจาวา ทำให้จำนวนของบรรทัดที่ใช้ในการพัฒนาตัวอย่างการใช้งานแพทเทิร์นลดลงได้เป็นจำนวนมาก ดังเห็นได้จากแพทเทิร์นเดี่ยวในภาษาจาวาใช้ 15 บรรทัด ภาษาดาร์ทใช้เพียง 9 บรรทัด แพทเทิร์นตัวแปลงภาษาจาวาใช้ 33 บรรทัด ภาษาดาร์ทใช้เพียง 20 บรรทัด และแพทเทิร์นตัวสังเกตการณ์ภาษาจาวาใช้ 61 บรรทัด ภาษาดาร์ทใช้เพียง 41 บรรทัด

งานวิจัยในอนาคตเราจะทำการศึกษาและพัฒนาตัวอย่างการใช้งานทั้ง 23 แพทเทิร์น เพื่อทำการเปรียบเทียบคุณลักษณะของภาษาดาร์ทเพิ่มขึ้น การปรับปรุงโครงสร้างของดีไซร์แพทเทิร์นเพื่อให้เหมาะสมกับคุณลักษณะของภาษา และการศึกษาประสิทธิภาพในเรื่องของเวลาที่ใช้ในการประมวลผล

6. เอกสารอ้างอิง

Alpert, S. R., Brown, K. & Woolf, B. (1998). **The Design Patterns Smalltalk Companion**. Boston, MA USA: Addison-Wesley Longman Publishing Co., Inc.

Bishop, J. (2008). **C# 3.0 Design Patterns**. 1st ed. Sebastopol, CA: O'Reilly Media.

Bracha, G., & Bak, L. (2011). **Dart, a new programming language for structured web programming.** GOTO Aarhus conferences, Oct. 2011.

Chandra, S. S., & Chandra, K. (2005). **A comparison of Java and C#.** CCSC: Rocky Mountain Conference, (pp. 238-254).

Clarke, S. (2001). **Evaluating a new programming language.** 13th Workshop of the Psychology of Programming Interest Group, (pp. 275-289). Bournemouth UK.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley.

Harmes , R. & Diaz, D. (2007). **Pro JavaScript Design Patterns.** Apress. 1st ed.

Metsker, S. J. & Wake, W. C. (2006). **Design Patterns in Java.** 2nd ed. Upper Saddle River, NJ: Addison-Wesley.

Miller, M. S. (2010). **"Future of Javascript" doc from our internal "JavaScript Summit".** Retrieved from https://gist.github.com/paulmillr/1208618

Nystrom, B. (2013). **Idiomatic Dart.** Retrieved from : http://www.dartlang.org/articles/idiomatic-dart/

Olsen, R. (2007). **Design Patterns in Ruby.** Boston, MA: Addison-Wesley Professional.

Schmager, F., Cameron, N., & Noble, J. (2010). **GoHotDraw: Evaluating the Go Programming Language with Design Patterns.** In Evaluation and Usability of Programming Languages and Tools (PLATEAU) 2010. Nevada, USA.

The Dart Team. (2013a). **Language Specification.** Retrieved from : http://www.dartlang.org/docs/spec/

The Dart Team. (2013b). **Technical Overview.** Retrieved June 12, 2012, from : http://www.dartlang.org/docs/technical-overview/

Walrath, K., & Ladd, S. (2012). **What is Dart?.** California: O'Reilly Media, Inc.,.

Walrath, K., Ladd, S., Hearse, C., Futato, D., Demarest, R., & Comer, R. (2013). **Chapter 2. A Tour of the Dart Language.** Retrieved July 13, 2013, from https://www.dartlang.org/docs/dart-up-and-running/contents/ch02.html

การประชุมวิชาการระดับชาติด้านวิทยาการและเทคโนโลยี ครั้งที่ 1
วันที่ 23 พฤษภาคม 2557 ณ โรงแรมดิเอ็มเพรส เชียงใหม่

การอิมพลีเมนต์ดีไซน์แพทเทิร์นในภาษาดาร์ท
Implemented Design Patterns in Dart Language

นริศรา ธาระพุทธ[1*] พิชโยทัย มหัทธนาภิวัฒน์[2] และศุภกฤษฎิ์ ตั้งเสริมสิทธิ์[3]

[1, 2, 3] สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี นครราชสีมา

อีเมล์: [1] n.tharaputh@gmail.com, [2] pmh@sut.ac.th, [3] supagrid.tangsermsit@gmail.com

บทคัดย่อ

ดีไซน์แพทเทิร์นเป็นการบันทึกประสบการณ์ในการออกแบบซอฟต์แวร์เชิงวัตถุ ที่มีส่วนช่วยในการออกแบบซอฟต์แวร์ให้สามารถนำกลับมาใช้ซ้ำได้อย่างมีประสิทธิภาพ งานวิจัยนี้ได้ทำการอิมพลีเมนต์ดีไซน์แพทเทิร์นในภาษาดาร์ท ซึ่งเป็นภาษาเขียนโปรแกรมใหม่ที่คาดว่าน่าจะได้รับความนิยมในอนาคต โดยมีวัตถุประสงค์เพื่อทำการปรับปรุงโครงสร้างของดีไซน์แพทเทิร์นให้เหมาะสมกับคุณลักษณะของภาษาดาร์ทเพื่อให้สามารถพัฒนาซอร์สโค้ดที่สามารถนำกลับมาใช้ซ้ำได้ง่าย ในการศึกษาผู้วิจัยได้ทำการศึกษาเฉพาะแพทเทิร์นต้นแบบ แพทเทิร์นเชิงประกอบ และแพทเทิร์นที่ระลึก ผลการศึกษาพบว่าแพทเทิร์นต้นแบบไม่สามารถทำการปรับปรุงโครงสร้างของแพทเทิร์นได้ แต่สามารถอิมพลีเมนต์เมธอดสำหรับการคัดลอกเพื่อนำไปใช้ซ้ำได้ ส่วนแพทเทิร์นเชิงประกอบและแพทเทิร์นที่ระลึกผู้วิจัยได้ทำการปรับปรุงโครงสร้างของแพทเทิร์นเพื่อให้สามารถพัฒนาซอร์สโค้ดให้สามารถนำกลับไปใช้ซ้ำได้

คำสำคัญ: ภาษาดาร์ท, ดีไซน์แพทเทิร์น, การใช้ซ้ำ

ABSTRACT

The record of experience in object-oriented software design is known as design patterns. Design patterns that help to software design can be reused effectively. In this paper, we implement design patterns in Dart language. Dart language is a new programming language that will be popular in the future. The objective is to improve the structure of design patterns to suit the features of the Dart language, and to develop source code that can be reuse it easily. Prototype, Composite and Memento patterns are chosen for the restructuring and implementation. We do not restructure Prototype pattern, but we implement the clone method for reusing. The Composite and Memento patterns are restructured so that they can be applied to the source code to reuse.

Keywords: dart language, design patterns, reuse

*Corresponding

1. บทนำ

การบันทึกประสบการณ์ที่เกี่ยวข้องกับการออกแบบซอฟต์แวร์เชิงวัตถุ และการนำการออกแบบนั้นไปใช้ในการเขียนโปรแกรมได้อย่างมีประสิทธิภาพถือเป็นสิ่งที่สำคัญ เพราะการออกแบบซอฟต์แวร์เชิงวัตถุเพื่อให้สามารถนำซอร์สโค้ดกลับมาใช้ซ้ำได้นั้นเป็นเรื่องที่ยากกว่าการออกแบบซอฟต์แวร์เชิงวัตถุทั่วไป ดีไซน์แพทเทิร์น (Design patterns) จึงเป็นทางเลือกสำหรับการออกแบบที่ทำให้ซอฟต์แวร์สามารถนำกลับมาใช้ใหม่ได้อย่างมีประสิทธิภาพ (Gamma and et al., 1995) ดีไซน์แพทเทิร์นของ Gang of Four (GoF) เป็นดีไซน์แพทเทิร์นที่ได้รับความนิยมเป็นอย่างมาก มีหนังสือเป็นจำนวนไม่น้อยที่อธิบายดีไซน์แพทเทิร์นของ GoF ในภาษาเขียนโปรแกรมที่แตกต่างกัน คือ ภาษาซีชาร์ป (C#) (Bishop, 2008) ภาษาจาวา (Java) (Metsker & Wake, 2006) ภาษาจาวาสคริปต์ (JavaScript) (Harmes & Diaz, 2007) ภาษารูบี (Ruby) (Olsen, 2007) และภาษาสมอลล์ทอร์ก (Smalltalk) (Alpert and et al., 1998) และมีงานวิจัยที่ทำการศึกษาดีไซน์แพทเทิร์นของ GoF สำหรับการประเมินภาษาโก (Go) (Schmager, 2010)

งานวิจัยจำนวนมากมีการศึกษาเรื่องการนำซอฟต์แวร์กับมาใช้ใหม่ (Software reuse) (Frakes & Kang, 2005; Sharma and et al., 2009; Singh and et al., 2011; Spoelstra and et al., 2011) การศึกษาเรื่องการใช้ซ้ำของซอร์สโค้ดระหว่างดีไซน์แพทเทิร์นและซอฟต์แวร์แพคเก็จ (Software packages) พบว่ามีการนำซอร์สโค้ดที่อิมพลีเมนต์ตามดีไซน์แพทเทิร์นกลับมาใช้มากกว่า 40 เปอร์เซ็นต์ (Ampatzoglou and et al., 2011) การอิมพลีเมนต์ดีไซน์แพทเทิร์นด้วยแอสเปกเจ (AspectJ) เพื่อศึกษาความสัมพันธ์กันระหว่างคลาส และการนำซอร์สโค้ดกลับมาใช้ซ้ำ (Hannemann, 2002) การศึกษาดีไซน์แพทเทิร์นที่มีอยู่ในซอฟต์แวร์ (Christensen, 2004) รวมถึงการศึกษาความสามารถในการนำซอร์สโค้ดกับมาใช้ซ้ำเฉพาะแพทเทิร์น (Oliveira and et al., 2008)

งานวิจัยนี้เป็นการอิมพลีเมนต์ดีไซน์แพทเทิร์นโดยใช้ภาษาดาร์ท ซึ่งเป็นภาษาที่ได้รับการพัฒนาและเผยแพร่เมื่อ ค.ศ. 2011 (Bracha & Bak, 2011) และยังเป็นภาษาที่ได้รับความสนใจเป็นอย่างมากในวงการพัฒนาโปรแกรมประยุกต์บนเว็บ (Web application) (Walrath & Ladd, 2012) ในการศึกษาได้มีการปรับปรุงลักษณะและโครงสร้างของแพทเทิร์นเพื่อให้เหมาะสำหรับคุณลักษณะเฉพาะของภาษาดาร์ท โดยมีวัตถุประสงค์หลักเพื่อให้สามารถนำซอร์สโค้ดกลับมาใช้ซ้ำได้ง่ายและมีประสิทธิภาพ ในงานวิจัยนี้ผู้วิจัยได้เลือกดีไซน์แพทเทิร์นในการศึกษาเฉพาะแพทเทิร์นต้นแบบ (Prototype pattern) เพื่อพัฒนาในส่วนของเมธอดสำหรับการคัดลอกอ็อบเจกต์ แพทเทิร์นเชิงประกอบ (Composite pattern) เพื่อพัฒนาในส่วนของการประกาศอินเตอร์เฟสสำหรับอ็อบเจกต์องค์ประกอบ และแพทเทิร์นที่ระลึก (Memento pattern) เพื่อพัฒนาในส่วนของการเก็บสถานะของคลาสเป้าหมายโดยมีคลาสตัวกลางสำหรับการติดต่อระหว่างคลาสเป้าหมายและคลาสที่เก็บสถานะ

2. วิธีการศึกษา

การดำเนินงานวิจัยในครั้งนี้ผู้วิจัยได้แบ่งวิธีการศึกษาออกเป็น 3 ส่วน คือ 1) การศึกษาดีไซน์แพทเทิร์น 2) การศึกษาภาษาดาร์ท และ 3) การอิมพลีเมนต์ดีไซน์แพทเทิร์นด้วยภาษาดาร์ท ซึ่งมีรายละเอียดของการดำเนินการดังนี้

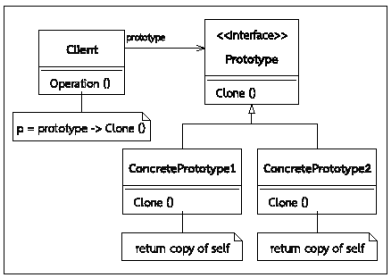
2.1 ดีไซน์แพทเทิร์น

ดีไซน์แพทเทิร์นที่ได้นำมาศึกษาในครั้งนี้เป็นดีไซน์แพทเทิร์นของ GoF (Gamma and et al., 1995) ประกอบด้วยแพทเทิร์นทั้งหมด 23 แพทเทิร์น ที่ได้มาจากการบันทึกลักษณะในการออกแบบและพัฒนาโปรแกรม โดยมีการแบ่งกลุ่มของดีไซน์แพทเทิร์นออกเป็น 2 กลุ่ม คือ แบ่งตามวัตถุประสงค์และแบ่งตามขอบเขต ในการศึกษาครั้งนี้เราได้เลือกแพทเทิร์นมาจากการจัดกลุ่มตามวัตถุประสงค์ คือแพทเทิร์นต้นแบบที่แสดงถึงการสร้างอ็อบเจกต์ แพทเทิร์นเชิงประกอบที่แสดงถึงลักษณะของโครงสร้าง และแพทเทิร์นที่ระลึกที่แสดงถึงลักษณะของพฤติกรรม

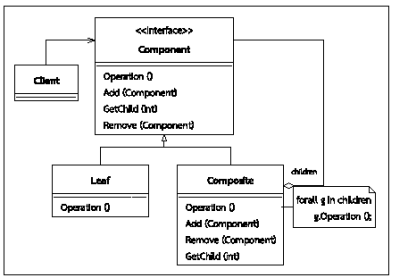
1) แพทเทิร์นต้นแบบ เป็นแพทเทิร์นที่สร้างอ็อบเจกต์ใหม่ด้วยวิธีการคัดลอกจากอ็อบเจกต์เริ่มต้น ลักษณะโครงสร้างของแพทเทิร์นต้นแบบแสดงในรูปที่ 1

2) แพทเทิร์นเชิงประกอบ เป็นแพทเทิร์นที่ช่วยให้ไคลเอนต์สามารถดำเนินการกับองค์ประกอบของอ็อบเจกต์ที่มีโครงสร้างประกอบกันเป็นแผนผังรูปต้นไม้แบบลำดับชั้นได้อย่างเท่าเทียมกัน โครงสร้างแพทเทิร์นเชิงประกอบแสดงในรูปที่ 2

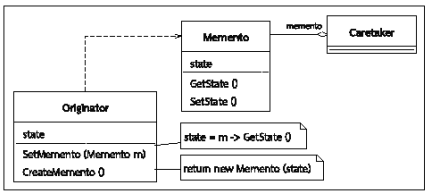
3) แพทเทิร์นที่ระลึก เป็นแพทเทิร์นที่ช่วยให้อ็อบเจกต์สามารถกลับไปยังสถานะก่อนหน้าได้ โดยการเก็บสถานะภายในของอ็อบเจกต์ ให้อยู่ภายนอกเพื่อไม่ให้เป็นการละเมิดกฎการห่อหุ้ม (Encapsulation) โครงสร้างแพทเทิร์นที่ระลึกแสดงในรูปที่ 3



รูปที่ 1 โครงสร้างแพทเทิร์นต้นแบบ



รูปที่ 2 โครงสร้างแพทเทิร์นเชิงประกอบ



รูปที่ 3 โครงสร้างแพทเทิร์นที่ระลึก

2.2 ภาษาดาร์ท

ภาษาดาร์ทเป็นภาษาที่ได้รับการพัฒนาโดยทีมพัฒนาของกูเกิล (Google) (Bracha & Bak, 2011) โดยมีแนวคิดที่สำคัญที่ได้รับการนำเสนอจากทีมพัฒนาดังนี้ (The Dart Team, 2013)

- ดาร์ทมี 2 แบบวิธี ในการกระทำการ (Execute) คือ แบบวิธีการผลิต (Production mode) เป็นแบบวิธีที่กำหนดไว้เป็นค่าเริ่มต้นในแบบวิธีนี้จะสามารถทำงานได้อย่างรวดเร็ว และแบบวิธีการตรวจสอบ (Checked mode) แบบวิธีนี้เหมาะสำหรับกระบวนการพัฒนาและการดักจับข้อผิดพลาด
- เครื่องมือดาร์ท (Dart tools) สามารถรายงานคำเตือน (Warnings) และข้อผิดพลาด (Errors) ซึ่งข้อผิดพลาดอาจเกิดขึ้นได้ทั้งในขณะเวลาคอมไพล์ (Compile-time) และรันไทม์ (Run-time)
- ดาร์ทมีการกำหนดการเข้าถึง 2 ลักษณะ คือ `public` และ `private` การเข้าถึงชนิด `public` ไม่ต้องระบุ `public` กำกับไว้ (เช่น `int num`) สำหรับการเข้าถึงชนิด `private` จะใช้สัญลักษณ์ `_` (Underscore) กำกับไว้ (เช่น `int _num`)
- ดาร์ทสนับสนุนการทำงานในระดับบนสุด (Top-level) ไม่ว่าจะเป็นการกำหนดฟังก์ชันหรือตัวแปร
- ดาร์ทสนับสนุนทั้งการระบุแบบชนิดคงที่ (Static types) และการกำหนดแบบชนิดเชิงทางเลือก (Optional types)

- ตัวแปรในดาร์ทสามารถกำหนดให้เป็นอ๊อบเจกต์ได้ และอ๊อบเจกต์ทุกอย่างจะเป็นอินสแตนซ์ของคลาส ไม่ว่าจะเป็นจำนวน (Number) ฟังก์ชัน หรือค่าว่าง (Null) โดยที่ทุก ๆ อ๊อบเจกต์จะต้องทำการสืบทอด (Inherit) มาจากคลาส `Object`

ภาษาดาร์ทถูกออกแบบโดยมีวัตถุประสงค์เพื่อให้เป็นภาษาที่ง่ายต่อการเรียนรู้ จึงทำให้วากยสัมพันธ์ (Syntax) ของภาษาดาร์ทโดยทั่วไปมีความใกล้เคียงกับภาษาอื่นเป็นอย่างมากโดยเฉพาะภาษาจาวา และภาษาจาวาสคริปต์ (Nystrom, 2013) ในการพัฒนาจึงจำเป็นต้องศึกษาคุณลักษณะของภาษาดาร์ทเพื่อให้สามารถใช้คุณลักษณะของภาษาได้ตรงตามวัตถุประสงค์ของการออกแบบภาษา

2.3 การอิมพลีเมนต์ดีไซน์แพทเทิร์นในภาษาดาร์ท

เมื่อได้ทำการศึกษาดีไซน์แพทเทิร์นของ GoF และทำการเลือกแพทเทิร์นที่จะใช้ในการศึกษาโดยทำการเลือกแพทเทิร์นที่จะศึกษาเฉพาะแพทเทิร์นต้นแบบสำหรับการศึกษาและพัฒนาในส่วนของการคัดลอกอ๊อบเจกต์ แพทเทิร์นเชิงประกอบสำหรับการศึกษาและพัฒนาในส่วนของการประกาศอินเตอร์เฟสสำหรับอ๊อบเจกต์องค์ประกอบ และแพทเทิร์นที่ระลึกสำหรับการศึกษาและพัฒนาในส่วนของการเก็บสถานะของคลาสเป้าหมายโดยมีคลาสตัวกลางสำหรับการติดต่อระหว่างคลาสเป้าหมายและคลาสที่เก็บสถานะ

หลังจากการศึกษาดีไซน์แพทเทิร์นแล้ว ผู้วิจัยได้ทำการพัฒนาตัวอย่างการใช้งานตามโครงสร้างดีไซน์แพทเทิร์นของ GoF เพื่อศึกษาถึงวิธีการพัฒนาตัวอย่างการใช้งานและคุณลักษณะของภาษาดาร์ทที่ใช้ในการพัฒนา เมื่อทราบถึงคุณลักษณะของภาษาดาร์ทที่เหมาะสมกับการพัฒนาตัวอย่างการใช้งาน ผู้วิจัยได้ทำการปรับปรุงโครงสร้างของดีไซน์แพทเทิร์นให้เหมาะสมกับคุณลักษณะของภาษาดาร์ทและการนำซอร์สโค้ดไปใช้ซ้ำ ขั้นตอนสุดท้ายผู้วิจัยได้ทำการพัฒนาซอร์สโค้ดตามโครงสร้างที่ได้ทำการปรับปรุงเพื่อสามารถนำซอร์สโค้ดที่ใช้ในการพัฒนากลับมาใช้ซ้ำได้ง่าย

3. ผลการศึกษา

ผลการศึกษาในงานวิจัยนี้แบ่งออกตามแพทเทิร์นที่ได้เลือกสำหรับการศึกษาดังนี้

3.1 แพทเทิร์นต้นแบบ

แพทเทิร์นต้นแบบผู้วิจัยไม่ได้ทำการปรับปรุงโครงสร้างของแพทเทิร์น แต่ผู้วิจัยได้ทำการพัฒนาคลาส `Cloneable` สำหรับสร้างเมธอดการคัดลอกอ๊อบเจกต์ เนื่องจากในดาร์ทยังไม่มีเมธอดสำหรับการคัดลอกที่รันบนบรรทัดคำสั่ง (Command-line) โดยการพัฒนาคลาส `Cloneable` ที่ภายในประกอบด้วยเมธอด `clone()` ดังแสดงในรูปที่ 4 ในการพัฒนาดังกล่าวอ้างอิงจากเอกสารของภาษาจาวาโดยเป็นการคัดลอกที่มีลักษณะเป็นการคัดลอกแบบตื้น (Shallow copy) ตัวอย่างการเรียกใช้งานเมธอด `clone()` และการพัฒนาตัวอย่างการใช้งานแพทเทิร์นต้นแบบดังแสดงในรูปที่ 5

```
1  //file: cloneable.dart
2  library Sut_Cloneable;
3  import 'dart:mirrors';
4  class Cloneable{
5   Object clone(var myClass){
6    InstanceMirror im = reflect(myClass);
7    ClassMirror MyClassMirror = im.type;
8    InstanceMirror im2 =
        MyClassMirror.newInstance
                    (const Symbol(''), []);
9    Iterable<DeclarationMirror> fileds =
        MyClassMirror.declarations.values.where(
              (dm) => dm is VariableMirror);
10   for(VariableMirror  vm in fileds){
11     im2.setField(vm.simpleName,
12        im.getField(vm.simpleName).reflectee);
13   }
14   return im2.reflectee;
15  }
16 }
```

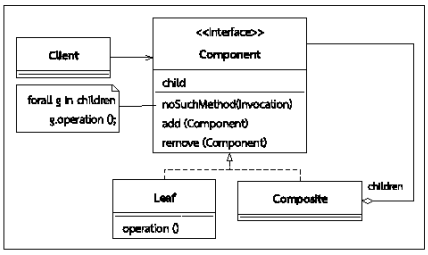
รูปที่ 4 คลาส `Cloneable`

```
1  library Sut_Prototype;
2  import 'cloneable.dart';
3  abstract class Area extends Object
                      with Cloneable {
4   void circle(int r);
5   Circle clone();
6  }
7  class Circle extends Area{
8   void circle(int r) =>
9     print("Area of Circle : ${(22/7)*r*r}");
10  Circle clone()=> super.clone(this);
11 }
12 void main(){
13  Circle obj1 = new Circle();
14  Area obj2 = obj1.clone();
15  obj2.circle(7);
16  print("${obj2 != obj1}");
17 }
```

รูปที่ 5 การเรียกใช้เมธอด `clone()` ในแพทเทิร์นต้นแบบ

3.2 แพทเทิร์นเชิงประกอบ

แพทเทิร์นเชิงประกอบผู้วิจัยได้ทำการปรับปรุงโครงสร้างของแพทเทิร์น โดยการเปลี่ยนแปลงที่อินเตอร์เฟซ Component และคลาส Composite ดังแสดงในรูปที่ 6 จากโครงสร้างเดิมของแพทเทิร์นที่กำหนดให้อินเตอร์เฟซ Component มีเพียงเมธอดนามธรรมและให้คลาส Composite ทำการสืบทอดและอิมพลีเมนต์เมธอดที่ได้ทำการโอเวอร์ไรด์ (Override) ได้ปรับปรุงโครงสร้างโดยกำหนดให้อินเตอร์เฟซ Component เป็นคลาสนามธรรมและได้ทำการอิมพลีเมนต์เมธอดในคลาส Component ทั้งหมดดังแสดงในรูปที่ 7 ส่วนคลาส Composite ให้ทำการสืบทอดตามโครงสร้างเดิมแต่ไม่ต้องทำการอิมพลีเมนต์ ตัวอย่างการใช้งานคลาส Component ในแพทเทิร์นเชิงประกอบดังแสดงในรูปที่ 8



รูปที่ 6 โครงสร้างแพทเทิร์นเชิงประกอบเมื่อทำการปรับปรุง

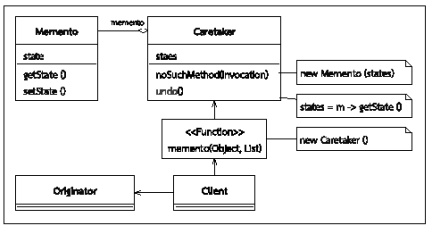
```
1  //file: component.dart
2  library Sut_Component;
3  import 'dart:mirrors';
4  abstract class Component{
5   List<Component> _child = [];
6   noSuchMethod(Invocation iv) {
7     for(Component c in _child){
8      reflect(c).invoke(iv.memberName,
                  iv.positionalArguments,
                  iv.namedArguments
           );
9      }
10   }
11  add(Component c)=> _child.add(c);
12  remove (Component c) => _child.remove(c);
13 }
```

รูปที่ 7 คลาสนามธรรม Component

```
1  library Sut_Composite;
2  import 'component.dart';
3  class Composite extends Component{}
4  class Ellipse extends Component{
5    void operation()=> print("Ellipse");
6  }
7  void main() {
8    Ellipse e = new Ellipse();
9    Ellipse e2 = new Ellipse();
10   Composite c  = new Composite();
11   Composite c2 = new Composite();
12   c2.add(e);
13   c2.add(e2);
14   c.add(c2);
15   c.operation();
16 }
```

รูปที่ 8 การใช้งานคลาส Component ในแพทเทิร์นเชิงประกอบ

3.3 แพทเทิร์นที่ระลึก



รูปที่ 9 โครงสร้างแพทเทิร์นที่ระลึกเมื่อทำการปรับปรุง

แพทเทิร์นที่ระลึกผู้วิจัยได้ทำการปรับปรุงโครงสร้างของแพทเทิร์นดังแสดงในรูปที่ 9 จากเดิมโครงสร้างของแพทเทิร์นจะประกอบด้วยคลาส 3 คลาส และการสร้างอินสแตนซ์ของคลาส Memento สำหรับการเก็บสถานะของอ็อบเจกต์จะถูกสร้างภายในคลาส Originator ผู้วิจัยได้ทำการปรับปรุงโครงสร้างของแพทเทิร์นการปรับปรุงเกิดขึ้นที่คลาส Caretaker คลาส Originator และมีการ

เพิ่มฟังก์ชัน memento() โดยไม่ได้ทำการปรับปรุงคลาส Memento สำหรับฟังก์ชัน memento() ถูกกำหนดให้เป็นฟังก์ชันสำหรับให้คลาส Client เรียกใช้เพื่อทำการสร้างอินสแตนซ์ของคลาส Caretaker และการสร้างอินสแตนซ์ของคลาส Memento จะถูกสร้างโดยคลาส Caretaker ดังแสดงในรูปที่ 10 ทำให้คลาส Originator ซึ่งเป็นคลาสเป้าหมายที่จะทำการเก็บสถานะนั้นสามารถเปลี่ยนแปลงเป็นคลาสอื่นได้ทันทีโดยไม่ต้องคำนึงถึงการสร้างอ็อบเจกต์เพื่อเก็บสถานะเมื่อมีการเปลี่ยนแปลง ตัวอย่างการใช้งานแพทเทิร์นที่ระลึกดังแสดงในรูปที่ 11

```
1  //file: memento.dart
2  library Sut_Memento;
3  import 'dart:mirrors';
4  class Memento {
5   Map<Symbol, dynamic> _state = {};
6   Memento(this._state);
7   Map get getState => this._state;
8  }
9  class Caretaker {
10  Object _o;
11  List<Symbol> _ls = [];
12  Map<int, Memento> _states = {};
13  int index = 0;
14  Caretaker(this._o,this._ls);
15  noSuchMethod(Invocation iv){
16   Map<Symbol, dynamic> _m = {};
17   InstanceMirror im = reflect(_o);
18   _ls.forEach((Symbol s){_m[s] =
              im.getField(s).reflectee;});
19    _states[index++] = new Memento(_m);
20    im.invoke(iv.memberName,
              iv.positionalArguments,
              iv.namedArguments
    );
21  }
22   undo(){
23    if(_states.isNotEmpty){
24     ` Map<Symbol, dynamic> _gm =
                  _states.values.last.getState;
25     _states.remove(_states.length-1);
26     _gm.forEach((Symbol K,dynamic V)
                  {reflect(_o).setField(K,V);});
27    }else{
28     print("-------State's empty!!!-------");
29    }
30   }
31  }
32  memento(o,l) => new Caretaker(o,l);
```

รูปที่ 10 คลาส Memento และคลาส Caretaker

```
1  library Sut_Originator;
2  import 'memento.dart';
3  class Originator {
4    int i = 0;
5    String str = null;
6    void m(s) {
7      i++;
8      this.str = s;
9    }
10 }
11 void main() {
12   Originator o = new Originator();
13   Caretaker c = memento(o,[#i,#str]);
14   c.m('State 0');
15   c.m('State 1');
16   print("${o.str} : ${o.i}");
17   c.undo();
18   print("${o.str} : ${o.i}");
19 }
```

รูปที่ 11 คลาส Originator และการใช้งานแพทเทิร์นที่ระลึก

4. อภิปรายผล

4.1 แพทเทิร์นต้นแบบ

แพทเทิร์นต้นแบบผู้วิจัยไม่ได้ทำการปรับปรุงโครงสร้างเนื่องจากแพทเทิร์นนี้มีวัตถุประสงค์เพื่อทำการคัดลอกอ็อบเจกต์ต้นแบบ ปัจจุบันในดาร์ทยังไม่มีเมธอดสำหรับคัดลอกอ็อบเจกต์ที่รันบนบรรทัดคำสั่ง ผู้วิจัยจึงได้ทำการพัฒนาในส่วนของเมธอดดังกล่าวไว้ในคลาส Cloneable ซึ่งเป็นคลาสที่สามารถนำไปใช้ซ้ำได้ทันทีเมื่อต้องการคัดลอกอ็อบเจกต์

ในการใช้งานคลาส Cloneable จะใช้คุณสมบัติไมนิก (Mixins) (Bracha, 2013) ของภาษาดาร์ท การใช้งานคุณสมบัติไมนิกทำให้คลาสที่ต้องการโอเวอร์ไรน์เมธอด clone() จะต้องมีการสืบทอดคลาสอย่างน้อยหนึ่งคลาส หากคลาสดังกล่าวไม่ได้มีการสืบทอดคลาสใด ๆ ให้คลาสนั้นทำการสืบทอดคลาส Object ดังแสดงในรูปที่ 5 บรรทัดที่ 3 และในบรรทัดที่ 10 เป็นการเรียกใช้เมธอด clone() โดยเรียกใช้ผ่าน super และจำเป็นต้องส่งอินสแตนซ์ของคลาสที่ทำการเรียกด้วย

4.2 แพทเทิร์นเชิงประกอบ

แพทเทิร์นเชิงประกอบผู้วิจัยได้ทำการปรับปรุงโครงสร้างในส่วนของคลาส Component และคลาส Composite ดังแสดงในรูปที่ 6 จากการปรับปรุงทำให้สามารถนำแพทเทิร์นเชิงประกอบไปใช้ซ้ำได้ทันที โดยการเปลี่ยนแปลงที่คลาส Leaf เท่านั้น จากโครงสร้างของแพทเทิร์นก่อนทำการปรับปรุงในรูปที่ 2 จะเห็นว่าเมธอดในคลาส Leaf จำเป็นที่จะต้องมีชื่อเดียวกันกับเมธอดที่ใช้ดำเนินการในคลาส Component ซึ่งก็คือเมธอด Operation() แต่ด้วยคุณสมบัติของภาษาดาร์ทคลาส Leaf

สามารถเปลี่ยนชื่อของเมธอดสำหรับดำเนินการได้ตามลักษณะของงานที่ต้องการใช้ได้ทันทีโดยที่ไม่ต้องทำการอิมพลีเมนต์เพิ่มเติมหรือเปลี่ยนแปลงคลาส Component

การเปลี่ยนแปลงคลาส Component ทำเพื่อให้คลาส Composite สามารถสืบทอดโดยที่ไม่จำเป็นต้องทำการโอเวอร์ไรน์เมธอดในคลาส Component ดังแสดงในรูปที่ 8 บรรทัดที่ 3 และเปลี่ยนแปลงเมธอด Operation() ในคลาส Component ตามโครงสร้างของแพทเทิร์นเชิงประกอบดังแสดงในรูปที่ 2 ให้เป็นเมธอด noSuchMethod() ตามโครงสร้างของแพทเทิร์นที่ได้รับการปรับปรุงดังแสดงในรูปที่ 6 เพื่อให้คลาส Leaf สามารถเปลี่ยนแปลงชื่อของเมธอดจาก Operation() เป็นชื่อเมธอดอื่นได้

4.3 แพทเทิร์นที่ระลึก

แพทเทิร์นที่ระลึกผู้วิจัยได้ทำการปรับปรุงโครงสร้างในส่วนของคลาส Caretaker คลาส Originator และมีการเพิ่มฟังก์ชัน memento() ดังแสดงในรูปที่ 9 คลาส Caretaker จากเดิมที่ภายในจะมีเฉพาะเมธอดสำหรับกลับไปยังสถานะก่อนหน้า ได้มีการเพิ่มเมธอด noSuchMethod() ซึ่งเป็นเมธอดที่จะทำการสร้างอินสแตนซ์ของ Memento เพื่อเก็บสถานะอ็อบเจกต์ของคลาส Originator จากโครงสร้างของแพทเทิร์นที่ได้รับการปรับปรุงแล้ว สำหรับคลาส Originator จะสามารถเปลี่ยนแปลงเป็นคลาสอื่น ๆ ได้ โดยที่ไม่ต้องทำการอิมพลีเมนต์ซอร์สโค้ดเพิ่มและสำหรับฟังก์ชัน memento() เป็นฟังก์ชันสำหรับการสร้างอินสแตนซ์ของคลาส Caretaker

การเรียกใช้งานของ Client จะเรียกใช้งานคลาส Caretaker ผ่านทางฟังก์ชัน memento() โดยการส่งค่าของอินสแตนซ์ของ Originator และค่าของขอบเขต (Fields) ที่ต้องการเก็บค่าให้กับ memento() การนำแพทเทิร์นที่ระลึกไปใช้สามารถนำไปใช้ซ้ำได้โดยการเปลี่ยนแปลงคลาส Originator

5. บทสรุปและข้อเสนอแนะ

ดีไซน์แพทเทิร์นช่วยให้การออกแบบซอฟต์แวร์เชิงวัตถุสามารถนำซอร์สโค้ดกลับมาใช้ซ้ำได้อย่างมีประสิทธิภาพ มีนักวิจัยจำนวนมากได้ทำการอธิบายดีไซน์แพทเทิร์นในภาษาเขียนโปรแกรมที่แตกต่างกัน งานวิจัยนี้ผู้วิจัยได้ทำศึกษาโครงสร้างของดีไซน์แพทเทิร์นของ GoF และทำการพัฒนาตัวอย่างการใช้งานตามโครงสร้างดังกล่าว จากนั้นได้ทำการปรับปรุงโครงสร้างของดีไซน์แพทเทิร์นเพื่อให้เหมาะสมกับคุณลักษณะเฉพาะของภาษาดาร์ทในการพัฒนาซอร์สโค้ดให้สามารถนำกลับมาใช้ซ้ำได้อย่างมีประสิทธิภาพ แพทเทิร์นที่ผู้วิจัยได้เลือกมาศึกษาคือ 1) แพทเทิร์นต้นแบบ สำหรับแพทเทิร์นนี้ผู้วิจัยไม่ได้ทำการปรับปรุงโครงสร้างของแพทเทิร์น แต่ทำการพัฒนาคลาส Cloneable ที่ประกอบด้วยเมธอด clone() สำหรับการคัดลอกอ็อบเจกต์แบบตื้น 2) แพทเทิร์นเชิงประกอบ ผู้วิจัยได้ทำการปรับปรุงโครงสร้างให้สามารถนำคลาส Component ไปใช้ซ้ำได้ และคลาส Leaf สามารถเปลี่ยนชื่อเมธอดสไหรับดำเนินการเป็นชื่อเมธอดใด ๆ ได้ และ 3) แพทเทิร์นที่ระลึกผู้วิจัยได้ทำการปรับปรุงโครงสร้างของแพทเทิร์นเพื่อให้สามารถนำซอร์สโค้ดจากแพทเทิร์นนี้ไปใช้ซ้ำได้ และสามารถเปลี่ยนแปลงคลาสเป้าหมายซึ่งก็คือคลาส Originator ไปเป็นคลาสเป้าหมายอื่นได้

งานวิจัยในอนาคตผู้วิจัยจะทำการศึกษาโครงสร้างของดีไซน์แพทเทิร์นทั้ง 23 แพทเทิร์น เพื่อพิจารณาความเหมาะสมในการปรับปรุงโครงสร้างของดีไซน์แพทเทิร์น และอิมพลีเมนต์ดีไซน์แพทเทิร์นตามโครงสร้างที่ได้ทำการปรับปรุงโดยมีวัตถุประสงค์หลักเพื่อให้สามารถนำซอร์สโค้ดของแต่ละแพทเทิร์นไปใช้ใหม่ได้ง่าย และศึกษาถึงเวลาที่ใช้ในการประมวลผลของดีไซน์แพทเทิร์นทั้งก่อนและหลังการปรับปรุงโครงสร้างของแพทเทิร์น

6. เอกสารอ้างอิง

Alpert, S. R., Brown, K. & Woolf, B. (1998). **The Design Patterns Smalltalk Companion**. Boston, MA USA: Addison-Wesley Longman Publishing Co., Inc.

Ampatzoglou, A., Kritikos, A., Kakarontzas, G. & Stamelos, I. (2011). An empirical investigation on the reusability of design patterns and software. **The Journal of Systems and Software**, 2265–2283.

Sharma, A., Grover, P. S. & Rajesh Kumar. (2009). Reusability assessment for software components. **ACM SIGSOFT Software Engineering Notes**, 1-6.

Bishop, J. (2008). **C# 3.0 Design Patterns**. 1st ed. Sebastopol, CA: O'Reilly Media.

Bracha , G. (2013, November). **Mixins in Dart**. Retrieved March 1, 2014, from https://www.dartlang.org/articles/mixins/

Bracha, G., & Bak, L. (2011). Dart, a new programming language for structured web programming. **GOTO Aarhus conferences, Oct. 2011.**

Frakes, W. B., & Kang, K. (2005). Software Reuse Research: Status and Future. **IEEE TRANSACTIONS ON SOFTWARE ENGINEERING**. 31(7), 529-536.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley.

Hannemann, J. K. (2002). Design Pattern Implementation in Java and AspectJ. **OOPSLA '02 Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**. 161-173.

Harmes , R. & Diaz, D. (2007). **Pro JavaScript Design Patterns**. Apress. 1st ed.

Henrik B. Christensen. (2004). Frameworks: Putting Design Patterns into Perspective. **ITICSE '04 Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education**, 142-145.

Metsker, S. J. & Wake, W. C. (2006). **Design Patterns in Java**. 2nd ed. Upper Saddle River, NJ: Addison-Wesley.

Nystrom, B. (2013, March). **Idiomatic Dart**. Retrieved March 1, 2014, from http://www.dartlang.org/articles/idiomatic-dart/

Oliveira, B. C., Wang, M. & Gibbons, J. (2008). The VISITOR Pattern as a Reusable, Generic, Type-Safe Component. **OOPSLA '08 Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications**, 439-456.

Olsen, R. (2007). **Design Patterns in Ruby**. Boston, MA: Addison-Wesley Professional.

Schmager, F. (2010). **Evaluating the GO Programming Language with Design Patterns**. Retrieved March 1, 2014, from https://ecs.victoria.ac.nz/twiki/pub/Main/TechnicalReportSeries/ECSTR11-01.pdf

Singh, Y., Bhatia, P. K. & Sangwan, O. (2011). Software reusability assessment using soft computing techniques. **ACM SIGSOFT Software Engineering Notes**, 1-7.

Spoelstra , W., Iacob, M. & Sinderen, M. v. (2011). Software reuse in agile development organizations: a conceptual management tool. **SAC '11 Proceedings of the 2011 ACM Symposium on Applied Computin**, 315-322 .

The Dart Team. (2013). **Chapter 2. A Tour of the Dart Language**. Retrieved March 2, 2014 from https://www.dartlang.org/docs/dart-up-and-running/contents/ch02.html

Walrath, K. & Ladd, S. (2012). **What is Dart?**. California: O'Reilly Media, Inc.,.

ประวัติผู้เขียน

นางสาวนริศรา ธาระพุทธ เกิดเมื่อวันที่ 1 กรกฎาคม พ.ศ. 2532 ที่จังหวัดนครราชสีมา สำเร็จการศึกษาระดับประถมศึกษาที่ 6 โรงเรียนบ้านปากช่อง (คุรุสามัคคี ๑) จังหวัดนครราชสีมา จากนั้นสำเร็จการศึกษาระดับชั้นมัธยมศึกษาปีที่ 6 โรงเรียนปากช่อง จังหวัดนครราชสีมา และสำเร็จการศึกษาระดับปริญญาตรี สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิยาลัยเทคโนโลยีสุรนารี ในปีการศึกษา 2554 หลังจากสำเร็จการศึกษาระดับปริญญาตรีแล้วได้เข้าศึกษาต่อระดับปริญญาโท สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปีการศึกษา 2555 ในระหว่างศึกษาได้มีโอกาสเป็นผู้ช่วยสอนปฏิบัติการรายวิชา เทคโนโลยีเชิงวัตถุ (Object - Oriented Technology) วิชาหัวข้อขั้นสูงในวิศวกรรมคอมพิวเตอร์ 1 (Advanced Topics In Computer Engineering I) วิชาการวิเคราะห์และออกแบบระบบงาน (System Analysis and Design) วิชาวิศวกรรมซอฟต์แวร์ (Software Engineering) รวมถึงการรับผิดชอบจัดทำปฏิบัติการสำหรับรายวิชาการเขียนโปรแกรมคอมพิวเตอร์ 1 (Computer Programming I) วิชาการเขียนโปรแกรมคอมพิวเตอร์ 2 (Computer Programming II) และได้รับทุนผู้ช่วยสอนและผู้ช่วยวิจัยในระหว่างการศึกษาด้วย โดยมีผลงานการประชุมวิชาการระดับชาติ 2 ผลงาน ดังแสดงในภาคผนวก ก