บทที่ 2

แนวคิดและทฤษฎีที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงรายละเอียดของงานวิจัยและทฤษฎีที่เกี่ยวข้องกับงานวิจัยในการวิเคราะห์และออกแบบโครงร่างสำหรับสร้างโปรแกรมเชิงวัตถุด้วยรูปแบบการออกแบบ โดยจะอธิบายโครงร่างในลักษณะต่าง ๆ อธิบายรูปแบบการออกแบบในความหมายและส่วนประกอบ อธิบายลักษณะเฉพาะของภาษาจาวา ซึ่งเป็นภาษาคอมพิวเตอร์ที่นำมาใช้ในการพัฒนางานวิจัยนี้ และการอธิบายแบบจำลองเอ็มวีซี ที่นำมาใช้กันแพร่หลายในการเขียนโปรแกรมเชิงวัตถุ

2.1 โครงร่าง (Framework)

Lewis, T. และคณะ [2] ได้นิยาม โครงร่าง คือลำดับชั้นของคลาส (Class hierachy) รวมกับรูปแบบสำเร็จรูป (Built-in model) ซึ่งได้แสดงให้เห็นหน้าที่ของแต่ละออปเจค และความสัมพันธ์กันในแต่ละลำดับชั้น

Gamma, E. และคณะ [3] ได้นิยามโครงร่าง คือ ชุดของคลาสที่ทำงานร่วมกันเพื่อนำมาสร้างคลาสใหม่สำหรับทำงานอย่างใดอย่างหนึ่งโดยเฉพาะ (Specific class) โดยมีสถาปัตยกรรมการออกแบบที่แบ่งคลาสเป็นส่วนๆ พร้อมทั้งกำหนดหน้าที่ และการทำงานร่วมกันของแต่ละส่วนไว้ชัดเจน ผู้ใช้สามารถดัดแปลงโครงร่างให้ตรงกับความต้องการโดยการสับคลาส จากคลาสโครงร่าง

Viljamaa, A. [1] ได้นิยามโครงร่าง คือ ชุดของซอฟต์แวร์ที่แบ่งคลาสออกเป็นส่วนๆ ที่มีหน้าที่ชัดเจน โดยที่ผู้ใช้สามารถดัดแปลงโครงร่างให้ตรงกับความต้องการ เพื่อใช้ในการแก้ปัญหาเฉพาะอย่างใดอย่างหนึ่ง ผู้พัฒนาโครงร่างได้นำเอาประสบการณ์ และความชำนาญในการแก้ไขปัญหาในการออกแบบมาใช้สร้างโครงร่างในส่วนพื้นฐานของการออกแบบที่โปรแกรมโดยทั่วไปต้องใช้ เป็นการลดปริมาณของชุดคำสั่งที่จะต้องเขียน และทดสอบ โดยโครงร่างส่วนใหญ่มักจะเตรียมในส่วนของการติดต่อกับผู้ใช้ (User interface) ไว้ให้

Camp, D. V. [4] ได้กล่าวถึงโครงร่างสามารถแบ่งได้เป็น 2 ลักษณะ คือ

1. *ไวท์บอคเฟรมเวอร์ค (White-box frameworks)*

   หมายถึง โครงร่างที่อยู่บนพื้นฐานของการสืบทอดมรดก (Inheritance) โดยจะมีโปรแกรมที่เตรียมไว้เป็นโครงร่างให้ผู้พัฒนาโปรแกรมทำการดัดแปลงไวท์บอคเฟรมเวอร์คให้ตรง

ความต้องการ โดยการสร้างสับคลาส (Deriving subclasses) ขึ้นมาใหม่แล้วทำเขียนทับฟังก์ชัน (Overriding member function) ซึ่งผู้ใช้จะมองเห็นความสัมพันธ์ระหว่าง พาเรนท์คลาส (Parent class) กับสับคลาส (Subclass) ที่เรียกว่าความสัมพันธ์แบบ “is A”

Gamma, E. [3] ได้กล่าวถึงข้อได้เปรียบ และข้อเสียเปรียบของไวท์บอคเฟรมเวอร์ค ไว้ว่าการสืบทอดมรดกคลาส (Class inheritance) เป็นการกำหนดตายตัวขณะเวลาแปลโปรแกรม (Compile-time) และใช้งานอย่างตรงไปตรงมา ทำให้ง่ายในการดัดแปลงเพื่อนำกลับมาใช้อีก อย่างไรก็ตามวิธีการนี้ ไม่สามารถเปลี่ยนแปลงได้ เพราะขณะเวลาดำเนินงาน (Run-time) ได้มีการสืบทอดมรดกจากพาเรนท์คลาส การที่จะไปกระทำการใดๆ กับสับคลาส ก็จะเกี่ยวพันกับพาเรนท์คลาสของมันด้วย การที่สับคลาสขึ้นอยู่กับพาเรนท์คลาส นี้ ทำให้ความยืดหยุ่นมีจำกัด

2. *แบล็คบอคเฟรมเวอร์ค (Black-box frameworks)*

หมายถึงโครงร่างที่อยู่บนพื้นฐานของการประกอบวัตถุนั่นคือฟังก์ชันใหม่ได้จากการประกอบของวัตถุ ซึ่งวิธีนี้จะมีความยืดหยุ่นมากกว่าการสืบทอดมรดก เพราะสามารถเปลี่ยนแปลงได้แบบพลวัต (Dynamic) ในขณะที่โครงสร้างของการสืบทอดมรดก เป็นแนวคิดแบบสถิต (Static) ความสัมพันธ์ระหว่าง คลาส กับ สับคลาส จะเป็นแบบ “has A” ซึ่งผู้ใช้ต้องการเข้าใจเฉพาะตัวประสานภายนอก (External interface) ของส่วนประกอบ (Component) เท่านั้น

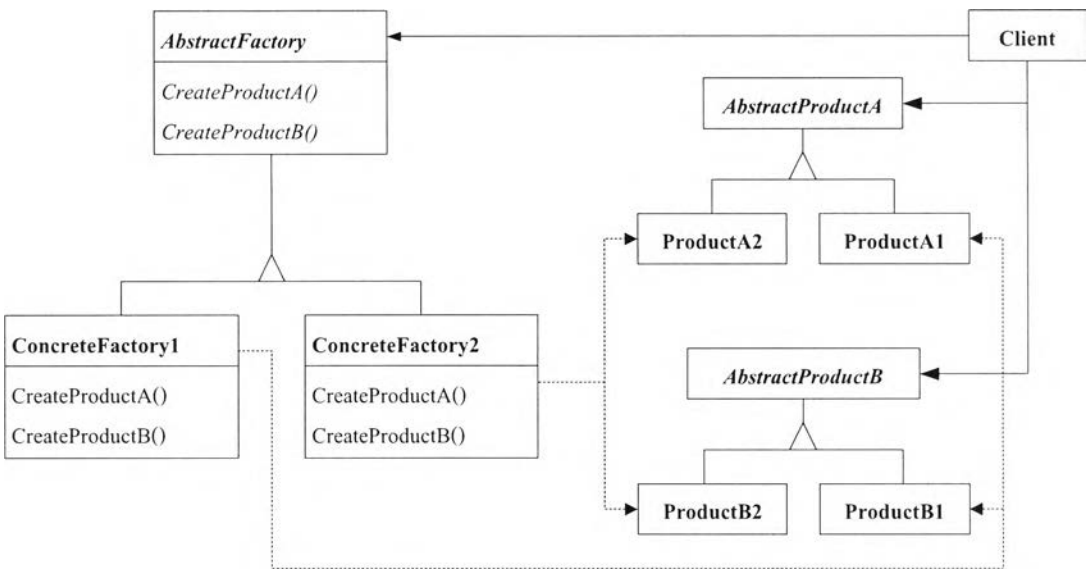
2.2 รูปแบบการออกแบบ

สถาปนิกชื่อ Christopher Alexander ได้พัฒนาแนวความคิดของรูปแบบการออกแบบ ในการออกแบบบ้าน และการติดต่อสื่อสาร ที่อธิบายถึงวิธีการแก้ปัญหาต่างๆ ในการออกแบบ โดยพยายามแยกปัญหาขนาดใหญ่ให้เล็กลงเป็นขั้นๆ ในลักษณะเดียวกัน โครงร่างก็สามารถที่จะออกแบบให้เป็นรูปแบบได้ แต่ละรูปแบบก็จะอธิบายถึงวิธีการแก้ปัญหาในส่วนเล็กๆ จากปัญหาใหญ่ในการออกแบบ ซึ่งปัญหาที่เกิดมักจะเกิดขึ้นซ้ำๆ เดิม และลักษณะการแก้ไขปัญหาก็จะเป็นแบบเดิม เราสามารถที่จะนำข้อมูลในการแก้ไขปัญหานี้ไปใช้กับปัญหาที่มีลักษณะคล้ายกัน ต่อมา Erich Gamma และคณะ [3] ซึ่งเรียกว่า Gang of Four (GoF) โดยได้รวบรวมรูปแบบการออกแบบไว้เป็น 3 ประเภท มีรายละเอียดดังต่อไปนี้

1 Creational เป็นรูปแบบการออกแบบเพื่อใช้ในการสร้างคลาสหรือออปเจค

1.1 Abstract Factory

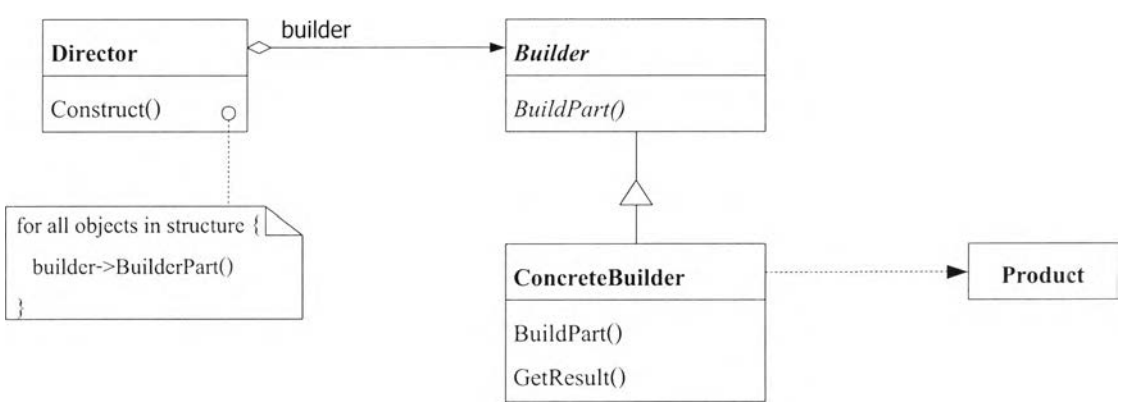
วัตถุประสงค์เพื่อกำหนดรูปแบบของคลาสเพื่อใช้สร้างออปเจคโดยจัดทำเป็นอินเตอร์เฟสคลาสสำหรับใช้ในการสร้างออปเจคต่างๆโดยไม่ต้องระบุลักษณะเฉพาะของคอนครีทคลาส ดังแสดงในรูปที่ 2.1 แผนภาพคลาสรูปแบบการออกแบบ Abstract Factory



รูปที่ 2.1 รูปแบบการออกแบบ Abstract Factory [3]

1.2 Builder

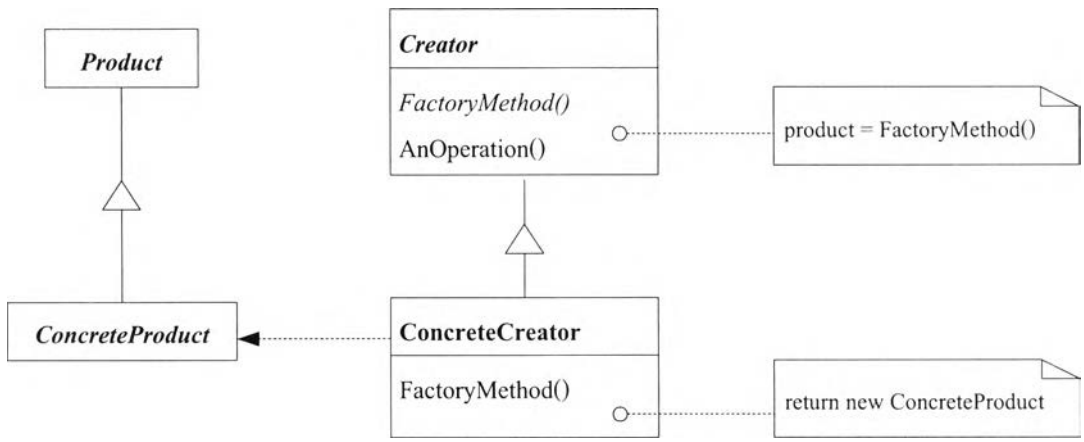
วัตถุประสงค์เพื่อกำหนดรูปแบบของคลาสเพื่อใช้สร้างออปเจคในแบบต่าง ๆ โดยใช้ขั้นตอนการสร้างเดียวกัน ดังแสดงในรูปที่ 2.2 แผนภาพคลาสรูปแบบการออกแบบ Builder



รูปที่ 2.2 รูปแบบการออกแบบ Builder [3]

1.3 Factory Method

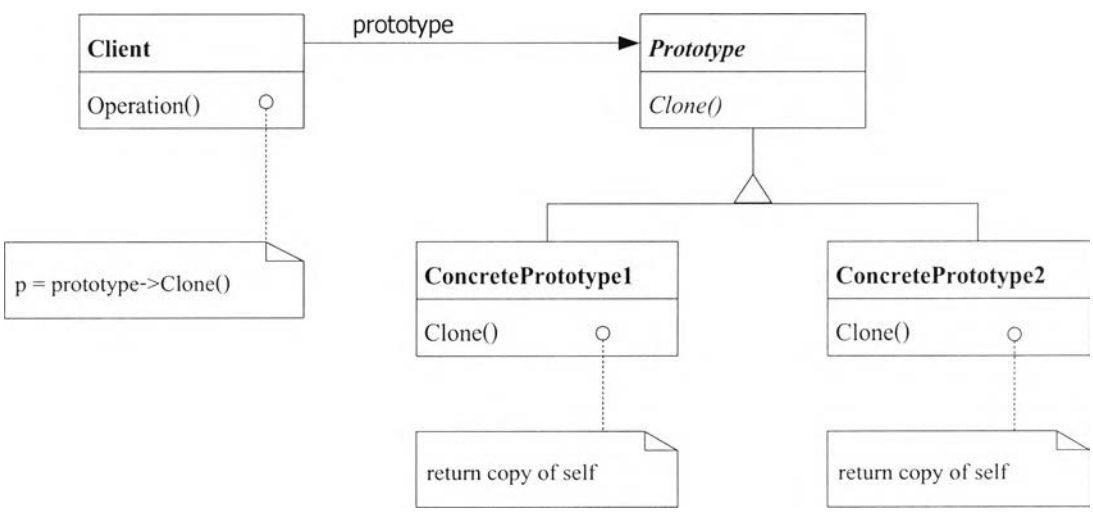
วัตถุประสงค์เพื่อกำหนดอินเตอร์เฟสเพื่อใช้เป็นแบบในการสร้างออปเจคโดยวิธีการสับคลาสไปใช้งาน ดังแสดงในรูปที่ 2.3 แผนภาพคลาสรูปแบบการออกแบบ Factory Method



รูปที่ 2.3 รูปแบบการออกแบบ Factory Method [3]

1.4 Prototype

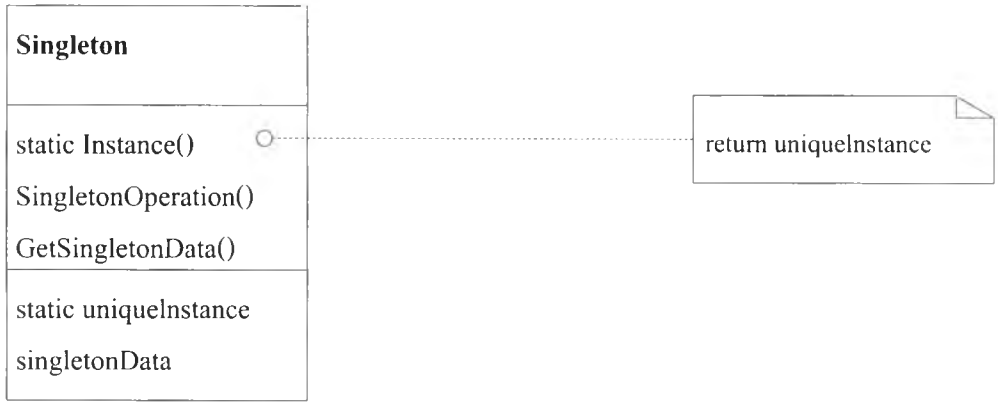
วัตถุประสงค์เพื่อกำหนดคุณลักษณะของออปเจคที่ต้องการไว้ก่อนที่จะสร้างเป็นอินสแตนท์ ดังแสดงในรูปที่ 2.4 แผนภาพคลาสรูปแบบการออกแบบ Prototype



รูปที่ 2.4 รูปแบบการออกแบบ Prototype [3]

1.5 Singleton

วัตถุประสงค์เพื่อสร้างคลาสที่เป็นอินสแตนท์เดียวเท่านั้นจะไม่มีซ้ำกัน ดังแสดงในรูปที่ 2.5 แผนภาพคลาสรูปแบบการออกแบบ Singleton

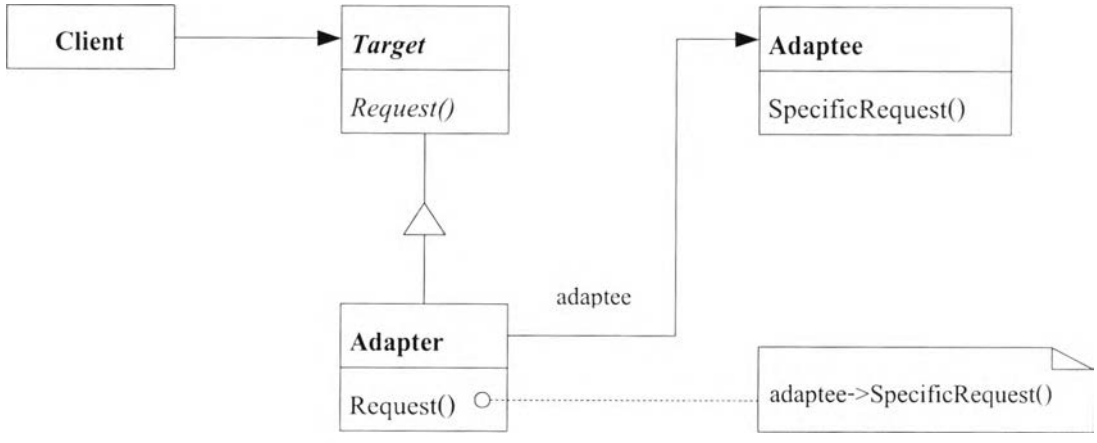


รูปที่ 2.5 รูปแบบการออกแบบ Singleton [3]

2 *Structural เป็นรูปแบบการออกแบบเพื่อใช้กำหนดโครงสร้างความสัมพันธ์ของคลาส*

2.1 Adapter

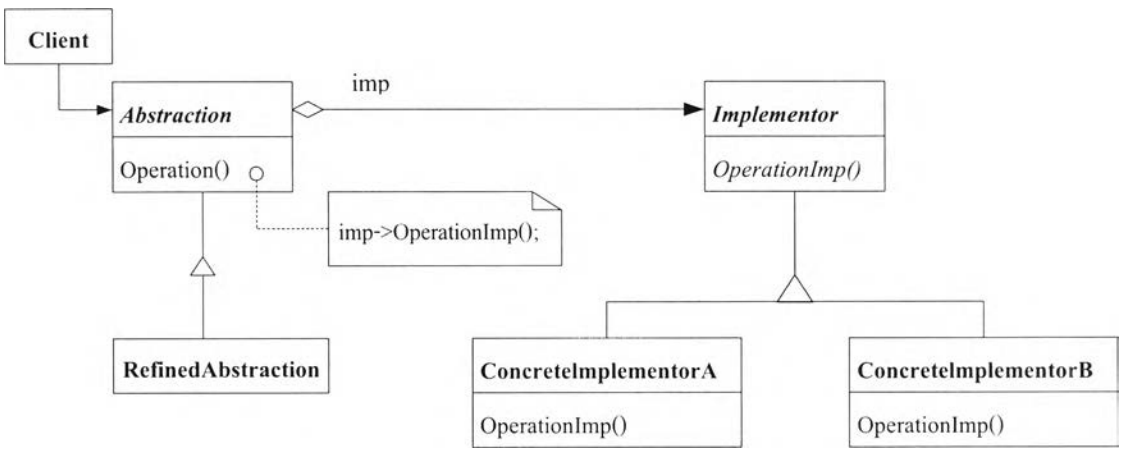
วัตถุประสงค์เพื่อกำหนดโครงสร้างอินเตอร์เฟสคลาสให้สามารถนำคลาสต่างชนิดกันนำมาใช้งานร่วมกันได้ ดังแสดงในรูปที่ 2.6 แผนภาพคลาสรูปแบบการออกแบบ Adapter



รูปที่ 2.6 รูปแบบการออกแบบ Adapter [3]

2.2 Bridge

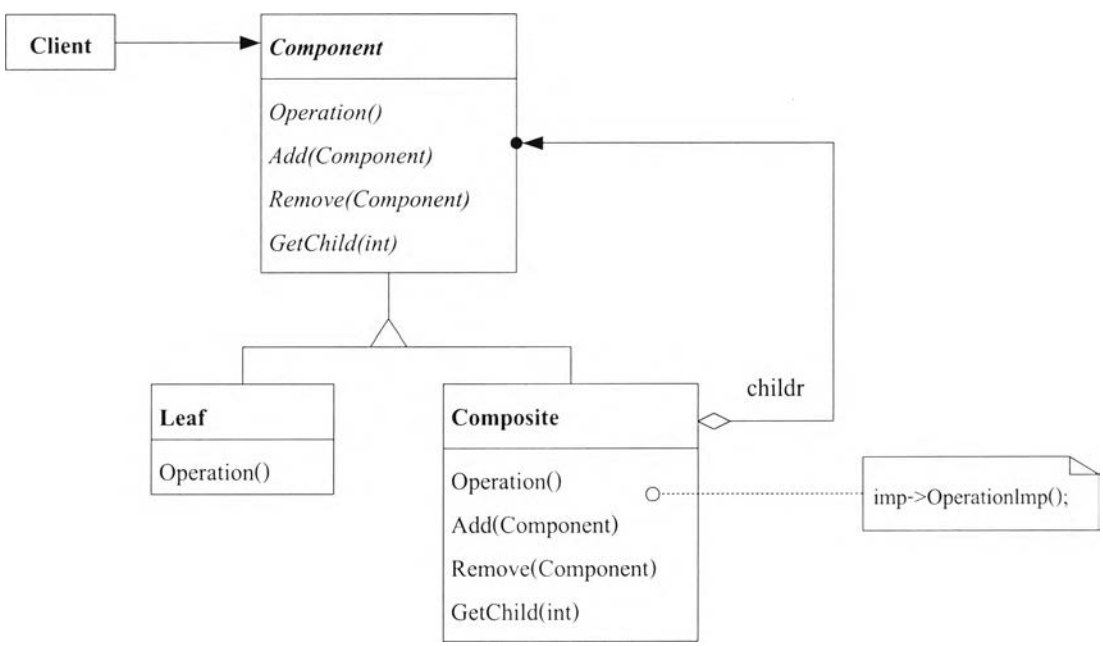
วัตถุประสงค์เพื่อกำหนดโครงสร้างของคลาสที่ใช้สร้างออปเจคในลักษณะต่างๆโดยไม่ขึ้นอยู่กับแอปสแตกคลาสต้นแบบ ดังแสดงในรูปที่ 2.7 แผนภาพคลาสรูปแบบการออกแบบ Bridge



รูปที่ 2.7 รูปแบบการออกแบบ Bridge [3]

2.3 Composite

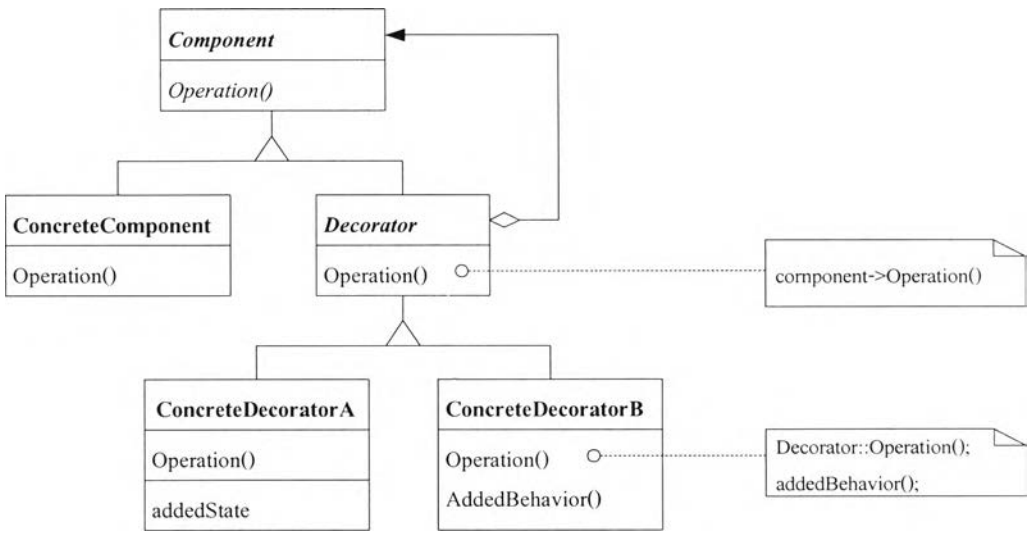
วัตถุประสงค์เพื่อกำหนดความสัมพันธ์ของออปเจคที่เป็นส่วนประกอบกันมีลักษณะเป็นโครงสร้างต้นไม้ ดังแสดงในรูปที่ 2.8 แผนภาพคลาสรูปแบบการออกแบบ Composite



รูปที่ 2.8 รูปแบบการออกแบบ Composite [3]

2.4 Decorator

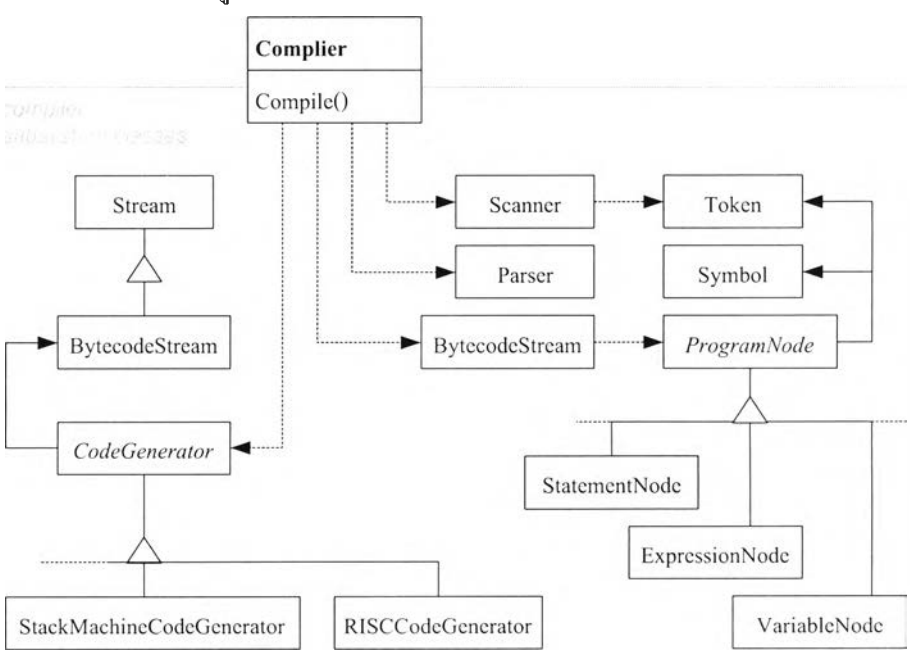
วัตถุประสงค์เพื่อกำหนดโครงสร้างของออปเจคที่สามารถปรับเปลี่ยนเพิ่มเติมภาระหน้าที่ให้แก่ออปเจคได้ทันที ดังแสดงในรูปที่ 2.9 แผนภาพคลาสรูปแบบการออกแบบ Decorator



รูปที่ 2.9 รูปแบบการออกแบบ Decorator [3]

2.5 Facade

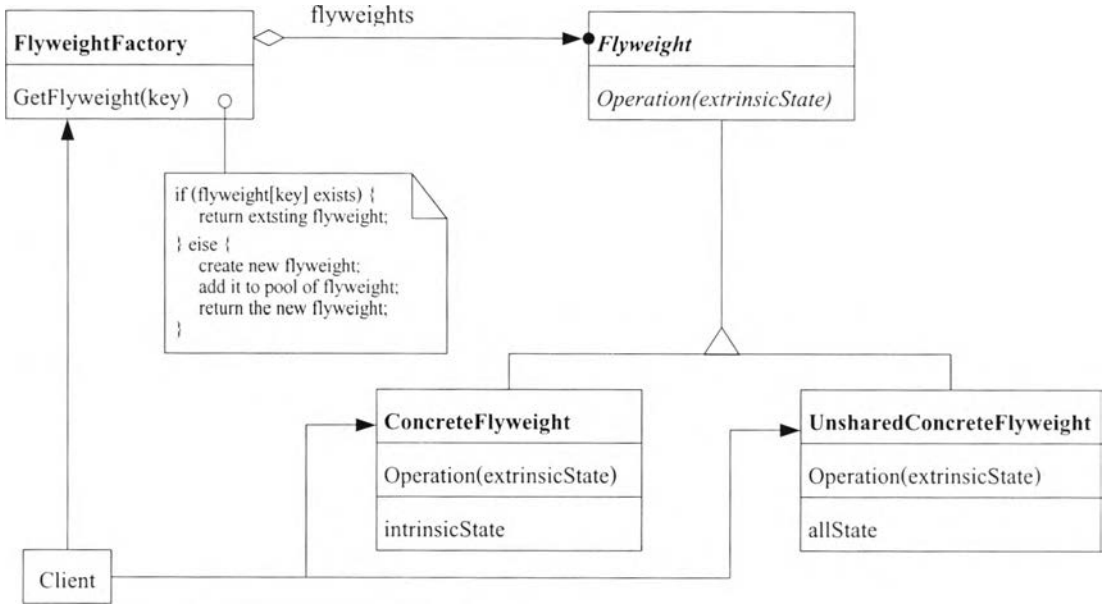
วัตถุประสงค์เพื่อกำหนดโครงสร้างของคลาสเป็นอินเตอร์เฟสเชื่อมต่อไปยังชุดของอินเตอร์เฟสในระบบย่อย ทำให้การใช้งานระบบย่อยได้ง่าย ดังแสดงในรูปที่ 2.10 แผนภาพคลาสรูปแบบการออกแบบ Facade



รูปที่ 2.10 รูปแบบการออกแบบ Facade [3]

2.6 Flyweight

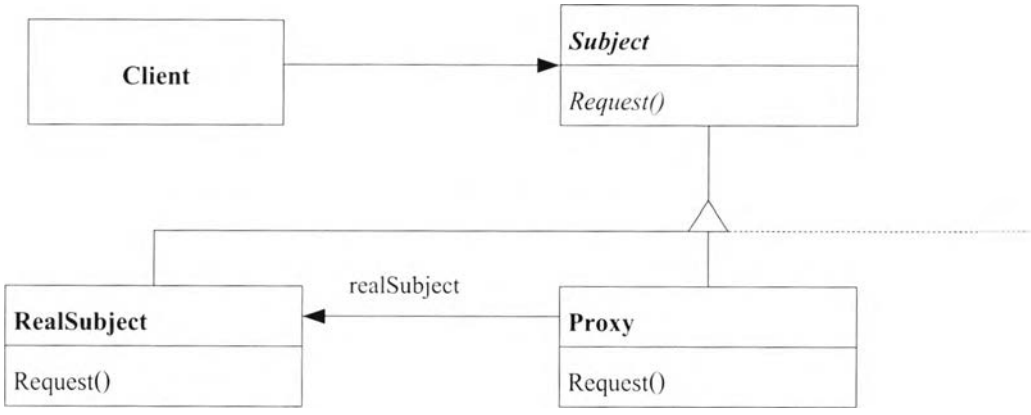
วัตถุประสงค์เพื่อกำหนดโครงสร้างให้ออปเจคจำนวนมากสามารถทำงานโดยใช้ทรัพยากรร่วมกันได้ ดังแสดงในรูปที่ 2.11 แผนภาพคลาสรูปแบบการออกแบบ Flyweight



รูปที่ 2.11 รูปแบบการออกแบบ Flyweight [3]

2.7 Proxy

วัตถุประสงค์เพื่อซ่อนตำแหน่งจริงในการเข้าถึงออปเจคเพื่อประโยชน์ในการรักษาความปลอดภัย ดังแสดงในรูปที่ 2.12 แผนภาพคลาสรูปแบบการออกแบบ Proxy

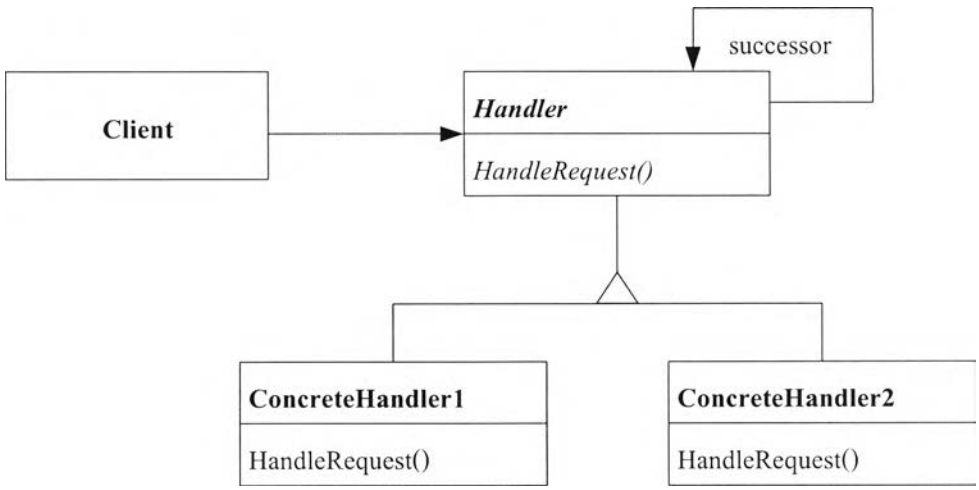


รูปที่ 2.12 รูปแบบการออกแบบ Proxy [3]

3 Behavioral เป็นรูปแบบการออกแบบที่เกี่ยวกับหน้าที่หรือลักษณะการทำงาน

3.1 Chain of Responsibility

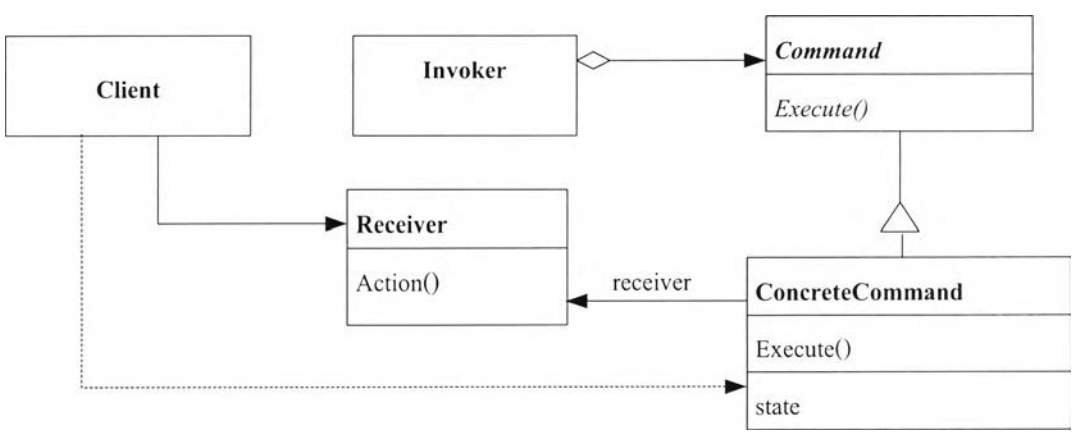
วัตถุประสงค์เพื่อให้ออปเจคปฏิบัติตามความต้องการเป็นไปตามลำดับ ดังแสดงในรูปที่ 2.13 แผนภาพคลาสรูปแบบการออกแบบ Chain of Responsibility



รูปที่ 2.13 รูปแบบการออกแบบ Chain of Responsibility [3]

3.2 Command

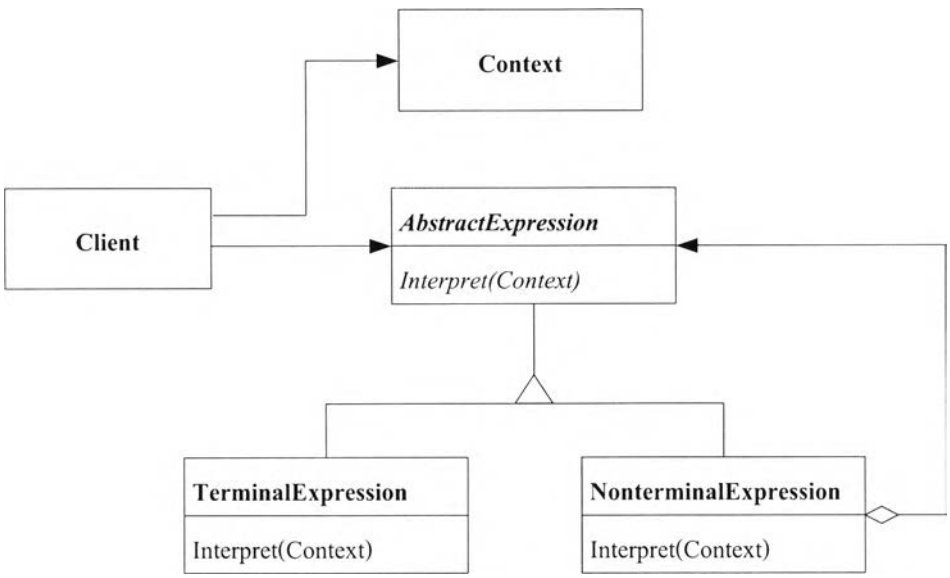
วัตถุประสงค์เพื่อกำหนดความต้องการให้รวมอยู่กับออปเจค โดยใช้พารามิเตอร์เป็นตัวแยกความแตกต่างของความต้องการ ดังแสดงในรูปที่ 2.14 แผนภาพคลาสรูปแบบการออกแบบ Command



รูปที่ 2.14 รูปแบบการออกแบบ Command [3]

3.3 Interpreter

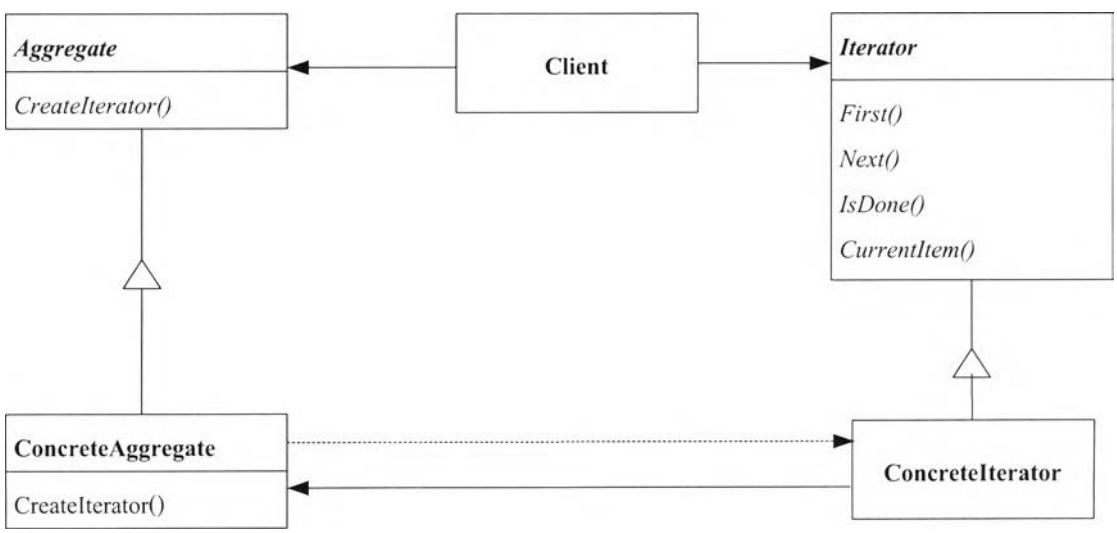
วัตถุประสงค์เพื่อกำหนดไวยกรณ์ของภาษา โดยมีการสร้างเมทธอดอินเตอร์พรีทเพื่อใช้แปลความหมายของไวยากรณ์ ดังแสดงในรูปที่ 2.15 แผนภาพคลาสรูปแบบการออกแบบ Interpreter



รูปที่ 2.15 รูปแบบการออกแบบ Interpreter [3]

3.4 Iterator

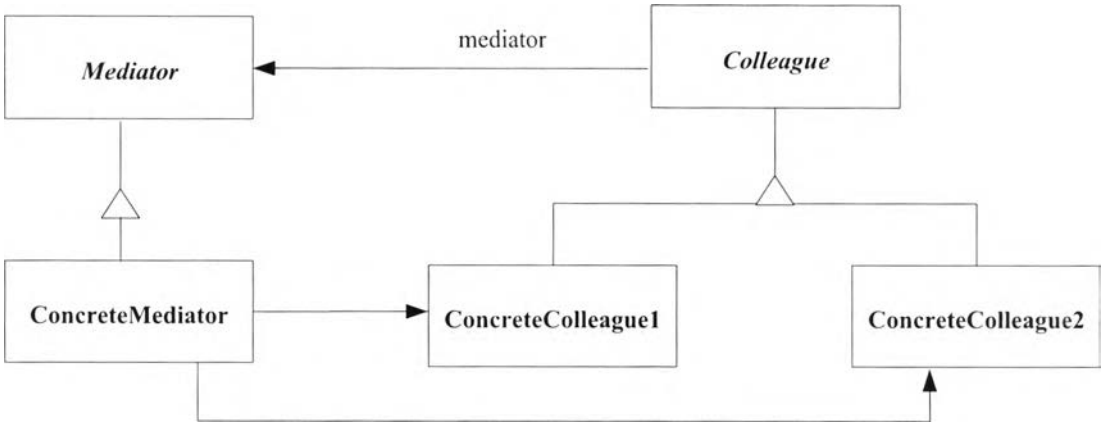
วัตถุประสงค์เพื่อกำหนดวิธีการเข้าถึงออปเจคแต่ละตัวที่อยู่รวมกันในแบบตามลำดับ ดังแสดงในรูปที่ 2.16 แผนภาพคลาสรูปแบบการออกแบบ Iterator



รูปที่ 2.16 รูปแบบการออกแบบ Iterator [3]

3.5 Mediator

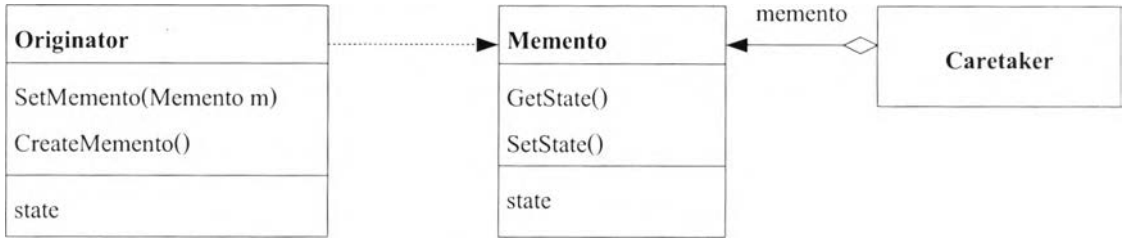
วัตถุประสงค์เพื่อกำหนดออปเจคที่เป็นตัวกลางในการควบคุมการโต้ตอบกันระหว่างออปเจคภายในกลุ่ม ดังแสดงในรูปที่ 2.17 แผนภาพคลาสรูปแบบการออกแบบ Mediator



รูปที่ 2.17 รูปแบบการออกแบบ Mediator [3]

3.6 Memento

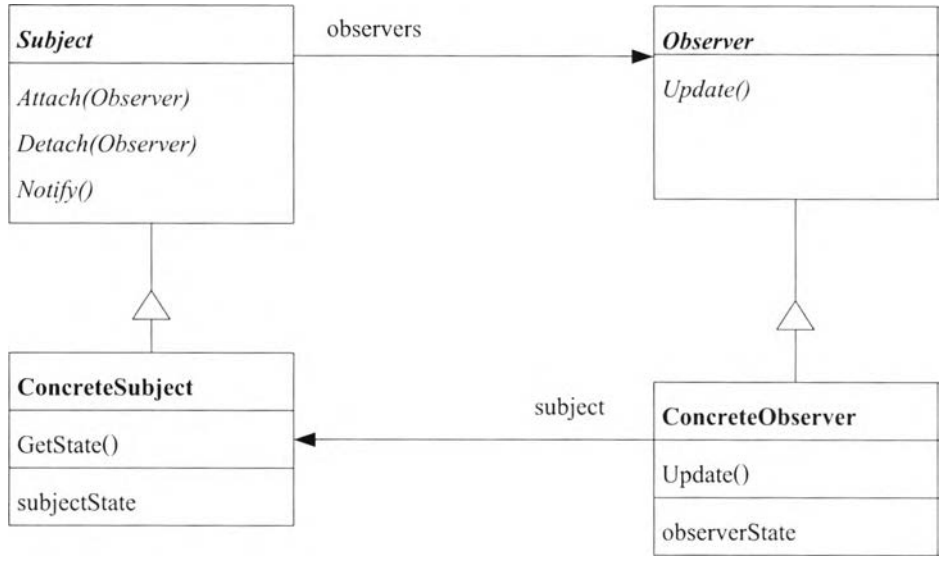
วัตถุประสงค์เพื่อเก็บสถานะของออปเจคไว้ เพื่อจะนำมาใช้เรียกคืนสถานะภายหลัง ดังแสดงในรูปที่ 2.18 แผนภาพคลาสรูปแบบการออกแบบ Memento



รูปที่ 2.18 รูปแบบการออกแบบ Memento [3]

3.7 Observer

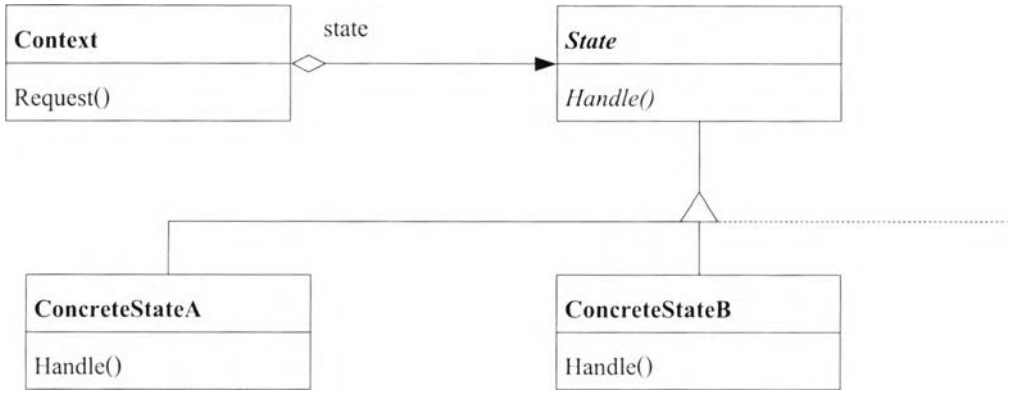
วัตถุประสงค์เพื่อกำหนดความสัมพันธ์แบบ "one-to-many" ระหว่างออปเจคโดยเมื่อออปเจคหนึ่งเปลี่ยนสถานะจะทำการแจ้งไปยังออปเจคอื่นๆให้ปรับปรุงสถานะตามไปด้วย ดังแสดงในรูปที่ 2.19 แผนภาพคลาสรูปแบบการออกแบบ Obsever



รูปที่ 2.19 รูปแบบการออกแบบ Observer [3]

3.8 State

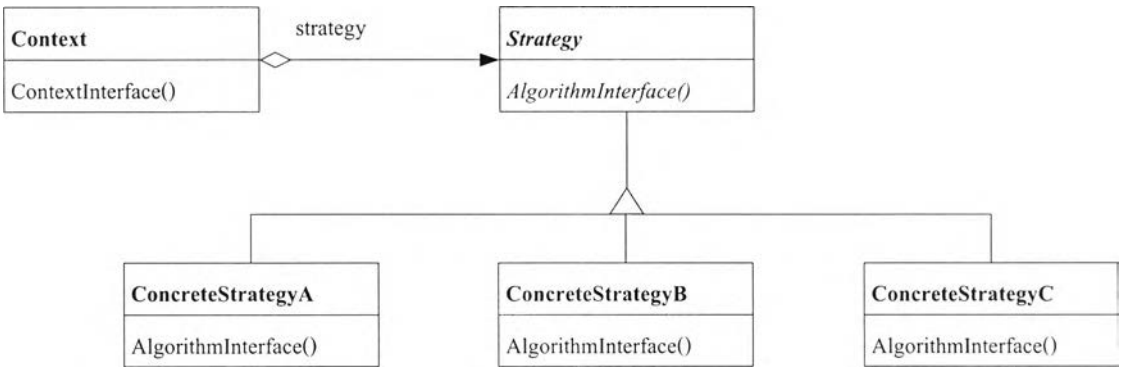
วัตถุประสงค์เพื่อให้ออปเจคสามารถเปลี่ยนแปลงพฤติกรรมหรือหน้าที่ตามการเปลี่ยนสถานะภายในของคลาส ดังแสดงในรูปที่ 2.20 แผนภาพคลาสรูปแบบการออกแบบ State



รูปที่ 2.20 รูปแบบการออกแบบ State [3]

3.9 Strategy

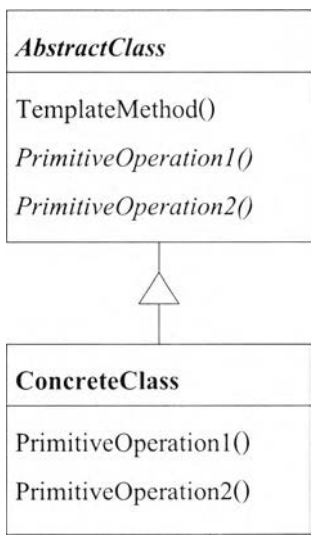
วัตถุประสงค์เพื่อให้ชุดของอัลกอริทึมสามารถนำมาใช้แลกเปลี่ยนกันได้ ดังแสดงในรูปที่ 2.21 แผนภาพคลาสรูปแบบการออกแบบ Strategy



รูปที่ 2.21 รูปแบบการออกแบบ Strategy [3]

3.10 Template Method

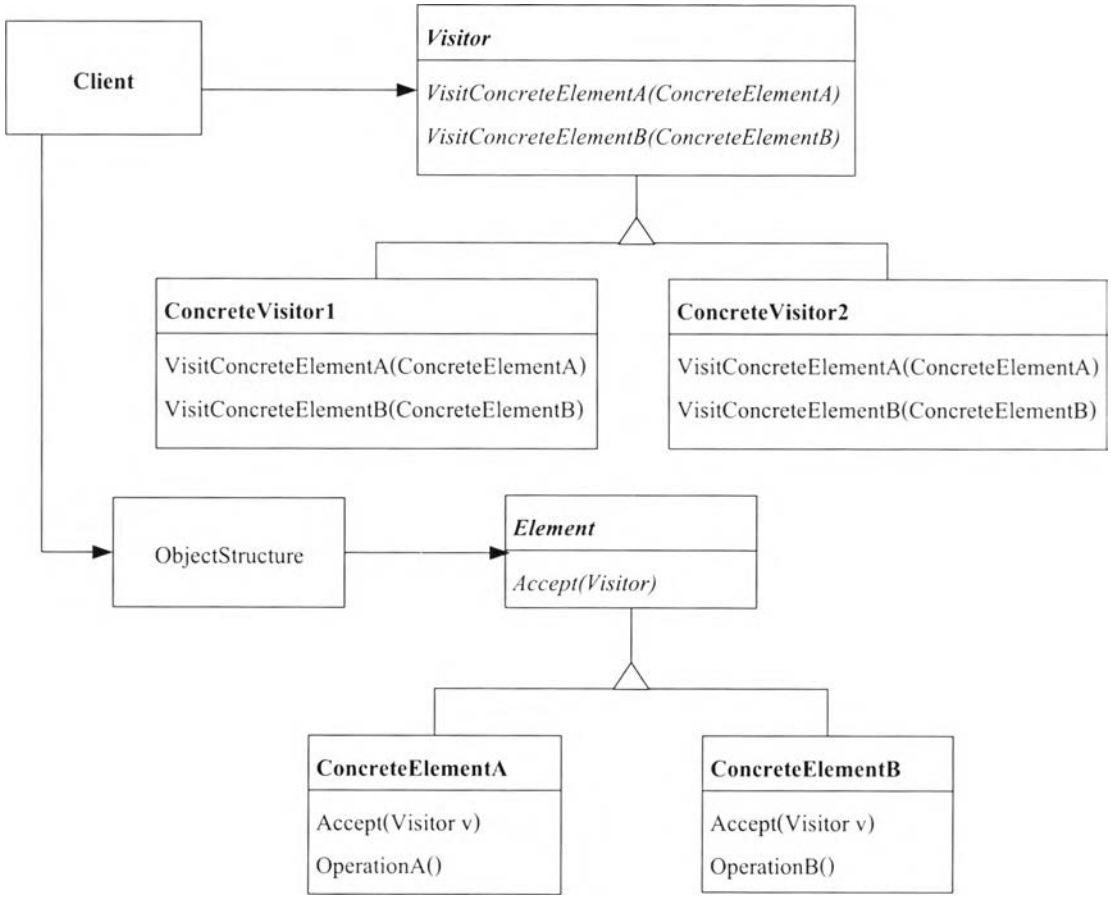
วัตถุประสงค์เพื่อกำหนดอัลกอริทึมในการทำงาน โดยจะกำหนดรายละเอียดในขั้นตอนสับคลาส ดังแสดงในรูปที่ 2.22 แผนภาพคลาสรูปแบบการออกแบบ Template Method



รูปที่ 2.22 รูปแบบการออกแบบ Template Method [3]

3.11 Visitor

วัตถุประสงค์เพื่อแสดงการทำงานที่ถูกกำหนดตามองค์ประกอบของโครงสร้างออปเจค โดยสามารถกำหนดการทำงานให้ใหม่โดยไม่ต้องเปลี่ยนแปลงคลาส ดังแสดงในรูปที่ 2.23 แผนภาพคลาสรูปแบบการออกแบบ Visitor



รูปที่ 2.23 รูปแบบการออกแบบ Visitor [3]

Gamma และคณะ [3] ได้ระบุถึงรูปแบบการออกแบบที่มีใช้กันนั้น โดยทั่วไปจะมี 4 ส่วนที่สำคัญคือ

1. *ชื่อรูปแบบการออกแบบ (Pattern name)*

   ชื่อจะแสดงลักษณะเฉพาะของรูปแบบการออกแบบ การตั้งชื่อที่เหมาะสมนั้นจำเป็นที่สุด เพราะจะสามารถสื่อความหมายของรูปแบบการออกแบบ และนำมาใช้เป็นศัพท์เรียกในการออกแบบ

2. *ปัญหา (Problem)*

   เป็นการอธิบายถึงลักษณะของปัญหาที่สามารถจะนำรูปแบบการออกแบบมาใช้

3. *วิธีการแก้ปัญหา (Solution)*

   เป็นการอธิบายถึงวิธีการแก้ปัญหา หลักการที่นำมาใช้ในการออกแบบ เช่น ความสัมพันธ์ (Relationships) หน้าที่ (Responsibilities) ความร่วมมือ (Collaborations) ในระดับนามธรรม (Abstract) มีหลายวิธีที่ใช้ในการอธิบาย เช่น ใช้รูปภาพแสดงให้เห็นการออกแบบคลาส (Design class) ในรูปแบบการออกแบบโดยอาจใช้เทคนิคการออกแบบเชิงวัตถุ (Object Modeling Technique : OMT) ของ Rumbaugh

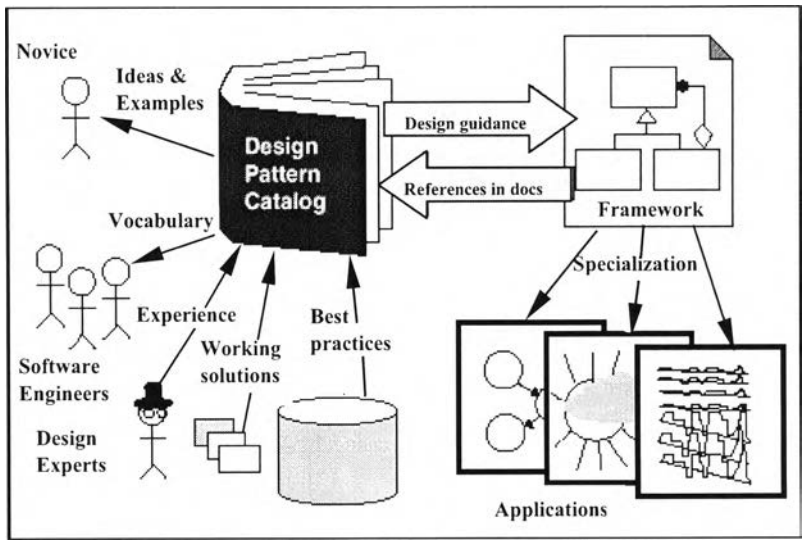
4. *ผลต่อเนื่อง (Consequences)*

   อธิบายถึงผลที่จะตามมาจากการใช้รูปแบบการออกแบบและช่วยให้บรรลุวัตถุประสงค์ได้อย่างไร

Gamma และคณะ [3] ได้กล่าวถึงความแตกต่างระหว่าง รูปแบบการออกแบบ และโครงร่าง ไว้ดังนี้

1. รูปแบบการออกแบบ เป็นแนวความคิดที่ช่วยในการออกแบบ ซึ่งจะมีความเป็นนามธรรมมากกว่าโครงร่าง โดยตัวของโครงร่างประกอบด้วยชุดคำสั่ง ที่สามารถนำไปเขียนชุดคำสั่งเพิ่มเติมแล้วนำไปใช้ได้เลย
2. รูปแบบการออกแบบ จะเป็นสถาปัตยกรรมขนาดเล็กกว่าโครงร่าง และทั่วไปจะนำรูปแบบการออกแบบหลาย ๆ แบบประกอบกันเพื่อสร้างโครงร่าง
3. รูปแบบการออกแบบ สามารถนำไปใช้ได้ทั่วไปใกล้เคียงกันแม้โปรแกรมประยุกต์ จะมีรูปแบบต่างกัน ต่างจากโครงร่างที่มักจะทำตามขอบเขตของโปรแกรมประยุกต์

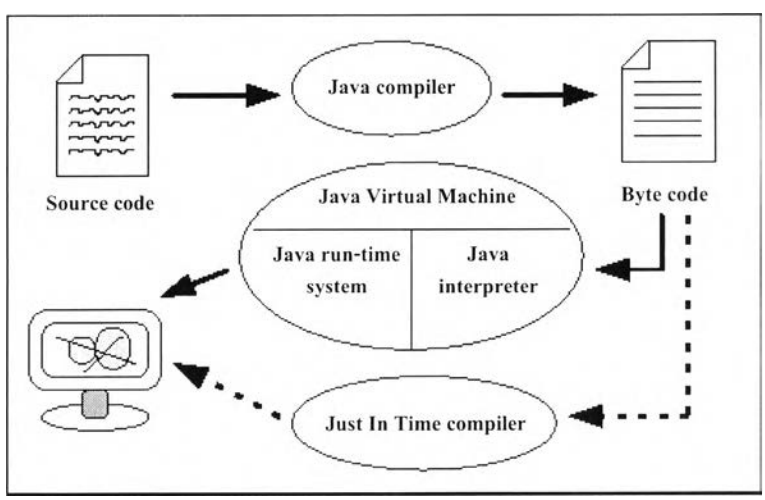
Viljamaa, J.[5] ได้แสดงความสัมพันธ์ของโครงร่างและรูปแบบการออกแบบ ดังแสดงในรูปที่ 2.24 โดยชุดรายการรูปแบบการออกแบบ (Design pattern catalog) จะได้จากการรวบรวมประสบการณ์ ผลงานที่ได้ออกแบบแล้วใช้งานได้จริง ผู้เชี่ยวชาญได้นำไปใช้แก้ปัญหาในการออกแบบซอฟต์แวร์ได้เป็นอย่างดี จากชุดรายการรูปแบบการออกแบบผู้ใช้สามารถนำแนวทางการออกแบบไปใช้สร้างโครงร่าง สำหรับพัฒนาโปรแกรมประยุกต์ในแบบต่าง ๆ และโครงร่างที่ได้ก็นำไปใช้เป็นเอกสารอ้างอิงในชุดรายการรูปแบบการออกแบบ ผู้ที่ศึกษาชุดรายการรูปแบบการออกแบบก็จะได้แนวคิดและตัวอย่างไปใช้เป็นแนวทางในการออกแบบ วิศวกรซอฟต์แวร์สามารถนำคำศัพท์ต่าง ๆ ไปใช้สื่อสารกันระหว่างคณะทำงาน



รูปที่ 2.24 ความสัมพันธ์ของโครงร่างและรูปแบบการออกแบบ [5]

2.3 การเขียนโปรแกรมภาษาจาวา

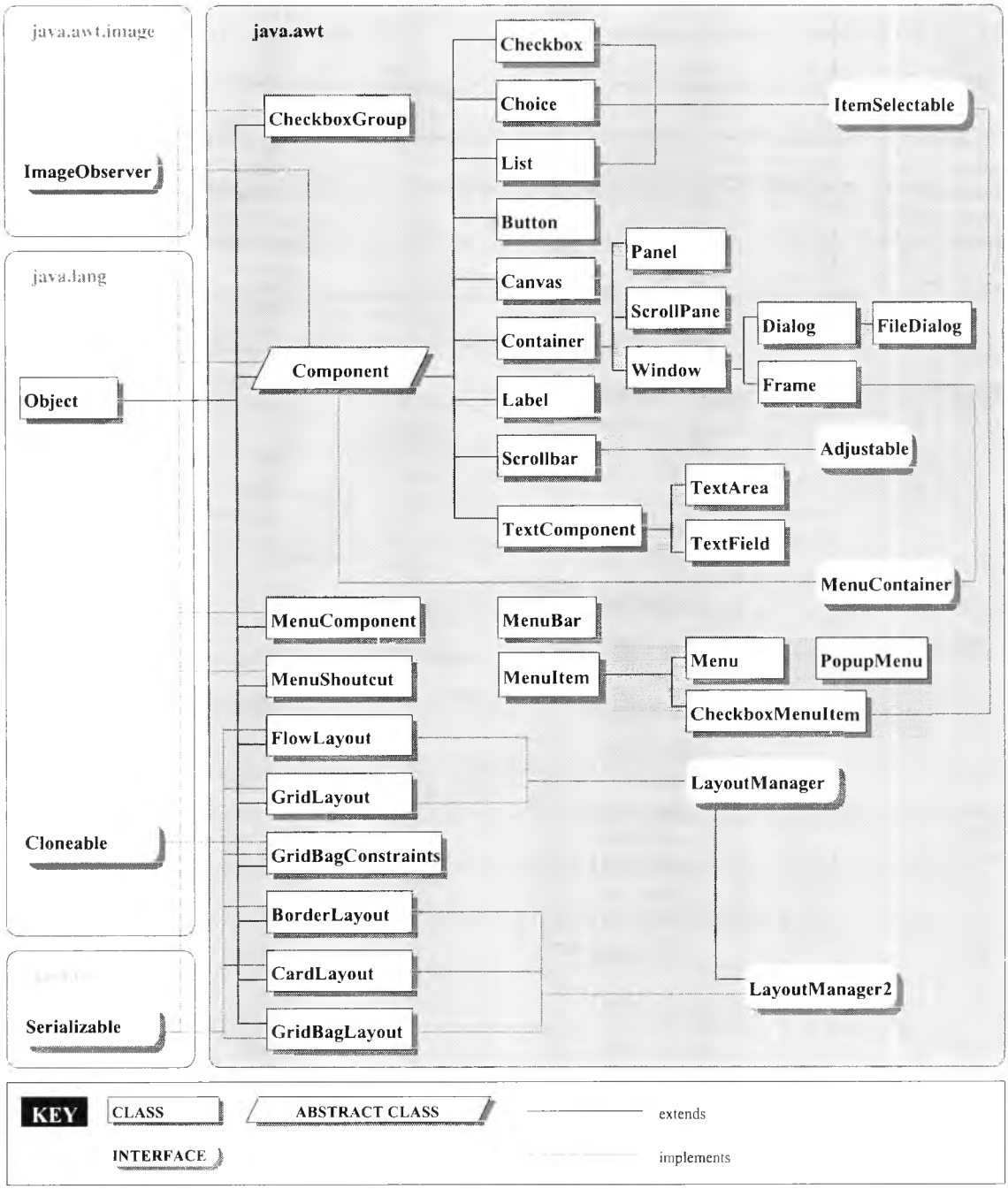
Viljamaa, A.[1] ได้อธิบายสภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวา ไว้ว่าจาวาเป็นภาษาสำหรับพัฒนาโปรแกรมเชิงวัตถุ พัฒนาโดยบริษัท Sun Microsystems มีโครงสร้างภาษาคล้ายภาษา C++ เมื่อเขียนโปรแกรมภาษาจาวาแล้วนำมาทำการคอมไพล์ ตัวคอมไพล์เลอร์ภาษาจาวาจะไม่แปลซอร์สโค้ดเป็นภาษาเครื่อง แต่จะแปลเป็นไบท์โค้ด (Byte code) ซึ่งไม่ใช่ตัวโปรแกรมที่จะใช้งานได้ทันที แต่ต้องนำไปใช้กับระบบปฏิบัติการที่มีตัวแปลภาษาจาวา ที่เรียกว่า Java Virtual Machine (JVM) ซึ่งประกอบด้วย ตัวแปลภาษาจาวา (Java interpreter) และรันไทม์ซิสเต็ม (Run-time system) หรืออีกวิธีหนึ่งคือการแปลไบท์โค้ด โดยใช้ Just In Time Compiler (JIT) ซึ่งจะแปลเป็นภาษาเครื่องใช้เฉพาะระบบปฏิบัติการ แตกต่างกันไปตามบริษัทผู้ผลิต ดังแสดงในรูปที่ 2.25



รูปที่ 2.25 สภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวา [1]

จาวาเป็นภาษาที่ใช้ได้กับคอมพิวเตอร์ทุกระบบ เมื่อพัฒนาโปรแกรมด้วยภาษาจาวา สามารถนำโปรแกรมไปใช้งานบนระบบระบบปฏิบัติการของคอมพิวเตอร์แบบใดๆก็ได้โดยไม่ต้องทำการคอมไพล์ใหม่ และไม่ต้องแก้ไขตัวโปรแกรมแต่อย่างใด จาวาเป็นภาษาที่ออกแบบมาเพื่อให้เขียนได้ง่าย กะทัดรัด จะมีโครงสร้างคล้ายภาษา C และ C++ จะแตกต่างในรายละเอียดปลีกย่อย เช่น ไม่มีการใช้พอยเตอร์ ผู้ที่เคยเขียนภาษา C หรือ C++ สามารถเรียนรู้ได้ง่าย ภาษาจาวามีการจัดการหน่วยความจำโดยอัตโนมัติ (Automatic memory management) ทำให้ผู้พัฒนาโปรแกรมไม่ต้องเป็นห่วงปัญหาที่จะเกิดกับหน่วยความจำ เช่น การเขียนทับหน่วยความจำ ภาษาจาวาไม่มีคำสั่ง GOTO ซึ่งจะแทนด้วยโครงสร้างคำสั่ง break และ continue ไม่มีการใช้เฮดเดอร์ไฟล์ (header files) การใช้ตัวปฏิบัติการเดียวกันกับหลาย ๆ ชนิดข้อมูล (Operator overloading) การสืบทอดคุณสมบัติจากหลายคลาส (Multiple inheritance)

ชุลีรัตน์ [6] ได้อธิบายภาษาจาวามีชุดของคลาสเตรียมไว้ให้ใช้เรียกว่าแพ็กเกจ (Package) เพื่ออำนวยความสะดวกในการพัฒนาโปรแกรม แพ็กเกจเอดับเบิลยูที(AWT)เป็นแพ็กเกจที่รวบรวมคลาสต่างๆที่เกี่ยวข้องกับการพัฒนาโปรแกรมโดยใช้หลักการของ Graphics User Interface (GUI) คือการใช้กราฟิกในการเป็นส่วนเชื่อมประสานระหว่างคอมพิวเตอร์กับผู้ใช้ในรูปแบบทางสัญลักษณ์ ดังแสดงในรูปที่ 2.26 แสดงลำดับชั้นของคลาสต่างๆในแพ็กเกจเอดับเบิลยูที ซึ่งสามารถจำแนกได้เป็นสามกลุ่มใหญ่ คือ กราฟิก(Graphics) คอมโพเนนท์(Components) และเลย์เอาท์เมเนเจอร์(Layout manager) คลาสกราฟิกเป็นคลาสที่รวบรวมเมทธอดต่างๆที่จัดการด้านการวาดภาพ ส่วนที่สองคือคอมโพเนนท์ เป็นส่วนที่เชื่อมประสานระหว่างผู้ใช้กับคอมพิวเตอร์ ได้แก่ ปุ่มคำสั่ง (Button) รายการเลือก(List) กรอบข้อความ(Text field) เมนู(Menu) และองค์ประกอบอื่นๆ ส่วนที่สามคือ เลย์เอาท์เมเนเจอร์ ช่วยในการจัดการคอมโพเนนท์ต่างๆบนจอภาพ เมทธอดที่สำคัญคือ “GridBagLayout” “BorderLayout” “GridLayout” โดยทำงานแบ่งพื้นที่บนจอเพื่อวางตำแหน่งคอมโพเนนท์ที่ต้องการ



รูปที่ 2.26 แผนภาพแสดงการสืบทอดคุณสมบัติต่างๆของแพ็กเกจเอดับเบิลยูที [6]

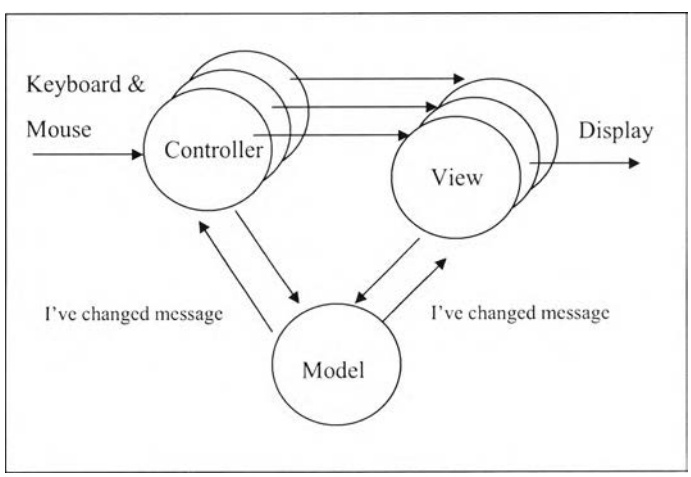
ภาษาจาวาสามารถเขียนในลักษณะแอพเพล็ต (Applets) ซึ่งเป็นโปรแกรมขนาดเล็กฝังไว้ในเวบเพจ (Web page) ที่เขียนด้วย HTML เพื่อความปลอดภัยของระบบคอมพิวเตอร์ ขอบเขตความสามารถของแอพเพล็ตจะถูกจำกัดไว้ด้วยเวปเบราเซอร์ (Web browser) แอพเพล็ต ไม่สามารถติดต่อใช้งาน หรือกระทำการใดๆกับระบบไฟล์ของเครื่อง หรือระบบคอมพิวเตอร์ปลายทางได้ เช่น การลบไฟล์ การแก้ไขไฟล์ เป็นต้น

2.4 แบบจำลองเอ็มวีซี

แบบจำลองเอ็มวีซี (MVC : Model-View-Controller) เสนอโดย Reenskaug [2] เป็นแบบจำลองสำหรับการเขียนโปรแกรมเชิงวัตถุที่ใช้กันอย่างกว้างขวาง โดยเฉพาะในโครงร่างของระบบติดต่อกับผู้ใช้งานแบบกราฟิก (Graphical user interface framework) แบบจำลองนี้เสนอให้แบ่งคลาสของโครงร่างออกเป็น 3 ชนิดดังต่อไปนี้

1. *คลาสโมเดล (Model class)*
   ใช้สำหรับเก็บข้อมูลซึ่งจะแสดงและถูกจัดการโดยโปรแกรมที่มีระบบติดต่อกับผู้ใช้งานแบบกราฟิก
2. *คลาสวิว (View class)*
   ใช้ควบคุมการแสดงข้อมูลในคลาสโมเดลบนหน้าจอ
3. *คลาสคอนโทรลเลอร์ (Controller class)*
   มีหน้าที่ในการรับข้อมูลนำเข้าจากแป้นพิมพ์และเมาส์ พร้อมทั้งส่งข้อความที่เหมาะสมให้กับคลาสโมเดล และคลาสวิวเพื่อให้สามารถเปลี่ยนแปลงแก้ไขข้อมูลในคลาสวิวได้

ตัวอย่างเช่น โปรแกรมประมวลผลคำ (Word processing) คลาสโมเดลจะถูกใช้เพื่อเก็บข้อมูลคำต่างๆ คลาสวิวจัดการการแสดงผลคำเหล่านี้บนหน้าจอ และคลาสคอนโทรลเลอร์รับข้อมูลที่ผู้ใช้ป้อนผ่านแป้นพิมพ์เป็นต้น ดังแสดงในรูปที่ 2.27 จะเป็นการแสดงการทำงานร่วมกันของคลาสต่างๆ ในแบบจำลองเอ็มวีซี จากรูปแสดงให้เห็นว่าคลาสวิว และคลาสคอนโทรลเลอร์มีคลาสโมเดลได้เพียงหนึ่งคลาส ในขณะที่คลาสโมเดลอาจมีคลาสวิวหรือคลาสคอนโทรลเลอร์ได้มากกว่าหนึ่งคลาส



รูปที่ 2.27 การทำงานร่วมกันของคลาสต่างๆในแบบจำลองเอ็มวีซี [2]

เหตุผลในการแยกคลาสต่างๆ เหล่านี้ออกจากกันเนื่องจากในบางครั้งการแสดงผลข้อมูลชุดหนึ่งๆ อาจแสดงได้ในหลายรูปแบบ เช่นอาจแสดงในรูปตาราง หรือรูปกราฟในแบบต่างๆ เมื่อมีการแก้ไขข้อมูลเพื่อแสดงผลในแบบใดแบบหนึ่ง จะทำให้เกิดความยุ่งยากในการทำให้การแสดงผลในรูปแบบอื่นๆ มีความถูกต้องตรงกันทั้งหมด ปัญหานี้สามารถแก้ไขได้โดยการแยกส่วนของข้อมูลออกจากส่วนแสดงผล พร้อมทั้งเก็บรายการรูปแบบการแสดงผลทั้งหมดไว้ การแสดงผลทุกรูปแบบจะใช้ข้อมูลจากแหล่งข้อมูลเดียวกันเพียงชุดเดียวในคลาสโมเดล ถ้ามีการแก้ไขเปลี่ยนแปลงข้อมูลดังกล่าว จะสามารถจัดการให้การแสดงผลทุกรูปแบบเปลี่ยนแปลงตามได้อย่างถูกต้อง