design pattern

คือ รูปแบบที่ใช้แก้ปัญหาต่างๆในการออกแบบซอฟต์แวร์ซึ้งปัญหานั้นมักจะเกิดกับเหล่าprogrammerทั่วโลก ดังนั้นจึงมีการคิดdesign patterns เพื่อแก้ปัญหาที่programmerมักเจอ

ในการออกแบบซอฟต์แวร์ ซึ่งกว่าจะเกิดขึ้นมาแต่ละแบบก็มาจากการทดลองซ้ำๆจนหารูปแบบหรือวิธีที่ดีที่สุดในการแก้ปัญหาแต่ละอย่าง

และด้วยความที่มันเป็นรูปแบบที่ใช้แก้ปัญหาต่างๆ เราจึงไม่สามารถที่จะก็อปปี้เอามาใช้งานทันที แต่ต้องทำความเข้าใจของ design pattern นั้นๆ และนำไปปรับใช้กับโค้ดของเราเอง

design pattern นั้นเค้าแบ่งออกได้เป็น 3 กลุ่มที่มีเป้าหมายต่างกันออกไป ประกอบด้วย

1.Creational patterns – เป็นกลุ่มที่ไว้ใช้สร้าง object ในรูปแบบต่างๆ ให้มีความยืดหยุ่น(flexible) และนำโค้ดมาใช้ซ้ำ(reuse)ได้

2.Structural patterns – กลุ่มนี้จะเป็นวิธีการนำ object และ class มาใช้งานร่วมกัน สร้างเป็นโครงสร้างที่มีความซับซ้อนยิ่งขึ้น โดยที่ยังมีความยืดหยุ่นและทำงานได้อย่างมีประสิทธิภาพ

3.Behavioral patterns – กลุ่มสุดท้ายนี้เป็นวิธีการออกแบบการติดต่อกันระกว่าง object ให้มีความยืดหยุ่นและสามารถติดต่อกันกันได้อย่างไม่มีปัญหา

Antipattern

คือ พฤติกรรมที่ไม่ดีที่ไม่ควรเอาอย่างในการออกแบบซอฟต์แวร์หรือแก้ไขปัญหา โดยมีการวิจัยและจัดหมวดหมู่เพื่อป้องกันความผิดพลาดที่อาจจะเกิดขึ้นในอนาคต

ตัวอย่างAntipatternที่ไม่ดี

Poor management: จัดการโครงการโดยไม่มีความรู้เพียงพอในเรื่องนั้น ๆ

Mess balls: ระบบไม่มีโครงสร้างที่เป็นที่รู้จัก

Wan Yingling: วัตถุรู้มากเกินไปหรือจำเป็นต้องทำมากเกินไปราวกับว่ามันมีอำนาจทุกอย่าง.

Anti-patterns ในการออกแบบเชิงวัตถุ

Universal class: นการออกแบบคลาสมีการรวมฟังก์ชันไว้มากเกินไป

Noisy: วัตถุประสงค์ของการสร้างวัตถุคือการส่งข้อความไปยังวัตถุอื่น

Copy-n-paste programming:ชอบคัดลอก (และแก้ไข) โค้ดที่มีอยู่แทนที่จะสร้างโซลูชันสากล

Anti-refactoring: ลบฟังก์ชันและแทนที่ด้วยคำอธิบายประกอบ

Singleton Pattern

เป็นรูปแบบการออกแบบซอฟต์แวร์ที่จำกัดจำนวนของ Object ที่ถูกสร้างขึ้นในระบบ ซึ่งจะเป็นประโยชน์เมื่อระบบต้องการจะมี Object นั้นเพียงตัวเดียวเพื่อป้องกันไม่ให้เกิดการทำงานซ้ำซ้อนกันเช่น class สำหรับการเก็บข้อมูล หรือเป็น Model ที่มีการเรียกใช้งานทั้งระบบ

ถ้าไม่มีก็สร้างขึ้นมาใหม่ ถ้ามีแล้วก็เรียกใช้ตัวเดิม

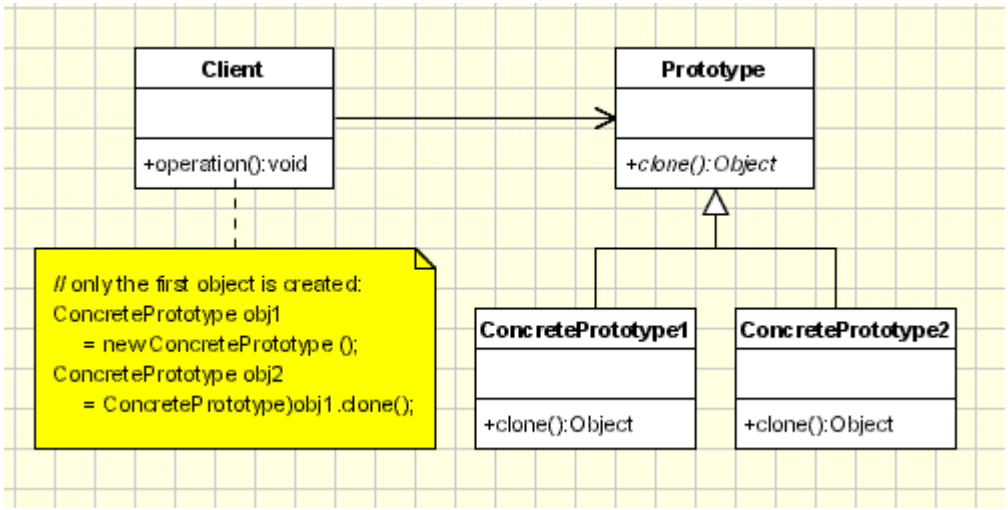

```java
public class Singleton {
    private static Singleton instance;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

Prototype Pattern

เป็นรูปแบบการออกแบบซึ่ง มีประโยชน์ทางด้านการใช้ Object ที่มีจำนวนมาก ๆ โดยที่เราไม่ต้องไป New Object ไปเรื่อย ๆ ซึ่งทำให้เปลือง ทรัพยากร โดยเรามีตัวต้นแบบแล้วก็ Clone มาเรื่อย ๆ

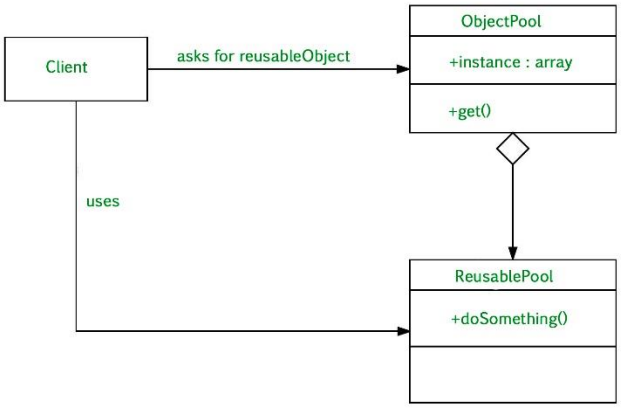


Client - จะสร้าง Object มาแล้วไปบอก Prototype Class ว่าจะคัดลอก

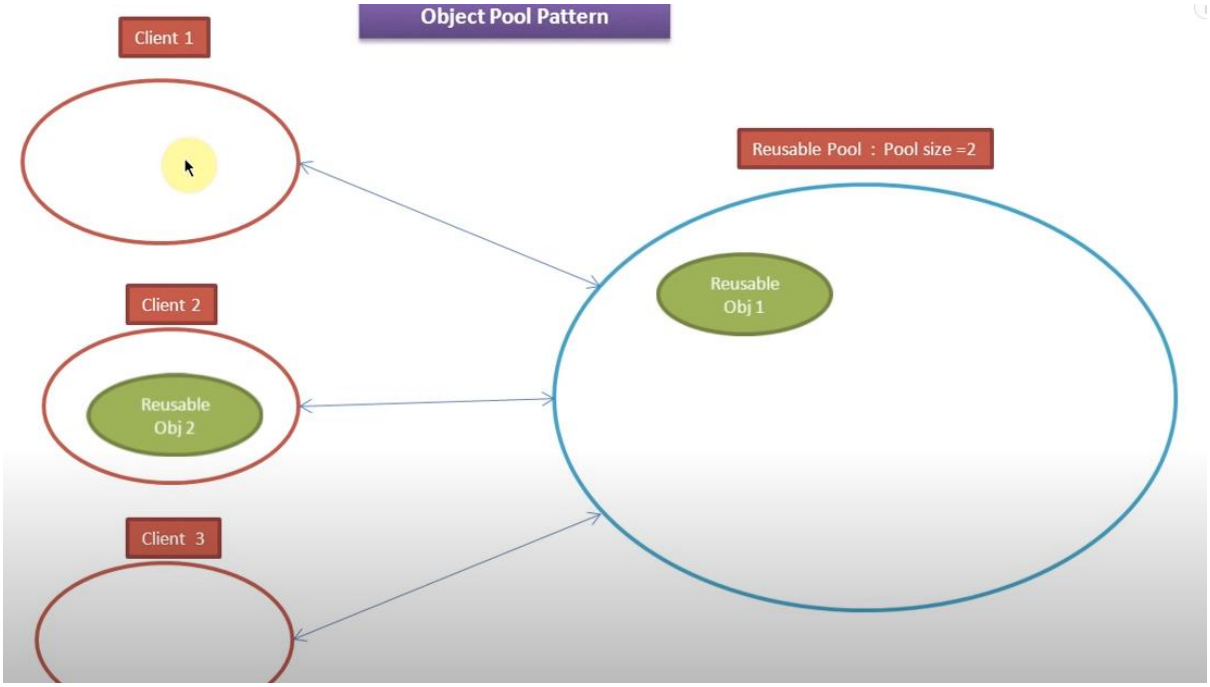
Prototype - กำหนด Interface class สำหรับการสร้าง Object และ Clone

ConcreatePrototype - เป็นรูปร่างหน้าตาของสิ่งที่สร้างขึ้นมาและสามารถ โคลนตัวเองได้

Object Pool pattern



โดยเปรียบเสมือนสระน้ำที่ในนั้นมี Object อยู่มากมาย โดย client ทำการขอ Object ไปแล้วถ้าจะมีคนอื่นมีใช้ตัวเดียวกันจะจำเป็นต้องให้ client ตัวแรกเสร็จสิ้นการใช้งานก่อนถึง client ตัวต่อไปจะสามารถนำมาใช้ได้



- ObjectPool จะมี Object อยู่ในนั้น
- Client จะเรียก Object ผ่าน ObjectPool

Functional Programming

Functional Programming นั้นคือการเขียนโปรแกรมโดยการเอาเทคนิคการทำ Composition (การประกอบร่าง) มาใช้ควบคู่กับ Pure function โดยข้อควรจำคือต้องหลีกเลี่ยงการ Share state, การแก้ไขข้อมูล (Mutate Data) และการเขียนโค้ดที่เกิด Side effect. สำหรับการเขียนโค้ดเราจะเขียนแบบ Declarative style แทนการเขียนแบบ Imperative Style

Declarative vs Imperative style

การเขียน Functional Programming เราจะใช้การเขียนโปรแกรมแบบ Declarative เป็นการเขียนโค้ดที่ไม่มี Control flow ส่วนการเขียนแบบ Imperative จะเขียนเป็นคำสั่งเพื่อบอกว่าระบบของเราทำอะไรบ้างไล่ลำดับการทำงานลงมาและมี Control flow