รายงาน

เรื่อง Builder Pattern

จัดทำโดย

นางสาว ปิยารัตน์ สังฆคุณ รหัสนิสิต 6030300636

สาขา วิศวกรรมคอมพิวเตอร์และสารสนเทศศาสตร์

เสนอ

อาจารย์ กาญจนา เอี่ยมสอาด

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชาSoftware Engineering ปีการศึกษา 2563

คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเกษตรศาตร์ วิทยาเขตศรีราชา

คำนำ

รายงานฉบับนี้แป็นส่วนหนึ่งของวิชา Software Engineering เพื่อให้ได้ศึกษาหาความรู้มีเนื้อหาเกี่ยวกับ Builder Pattern ซึ่งปีนเรื่องเกี่ยวกับการจัดการDesignให้ง่ายต่อการใช้งานมากยิ่งขึ้น และทำเป็นเอกสารสรุปประกอบความเข้าใจ

ผู้จัดทำหวังว่ารายงานฉบับนี้จะทำให้ผู้อ่านหรือผู้ที่กำลังศึกษาสามารถเข้าในเรื่อง Builder Pattern ได้มากยิ่งขึ้น หากมีข้อผิดพลาดประการใดก็ขออภัยมา ณ ที่นี้ด้วย

ผู้จัดทำ

ปิยารัตน์ สังฆคุณ

Builder Pattern

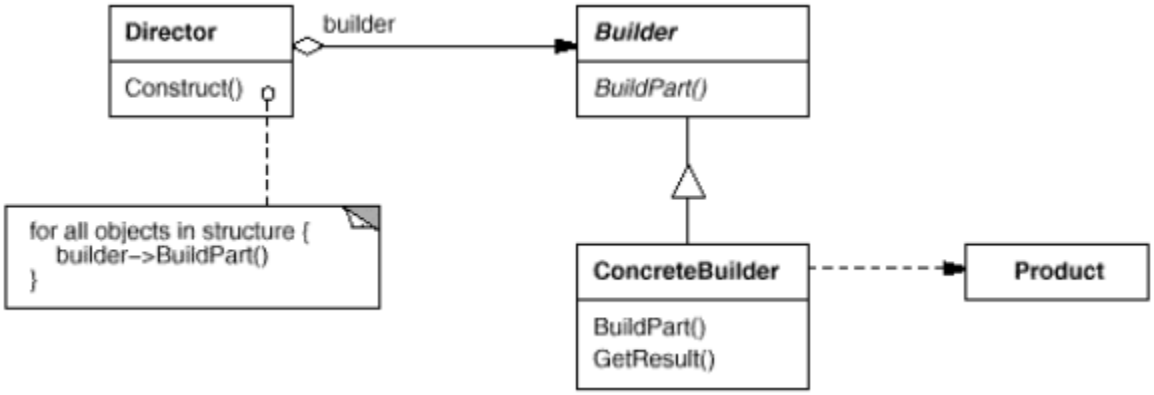
คือ Design Pattern ที่อยู่ในกลุ่มของ Creation Pattern เราสามารถสร้างObject ทีละส่วนเองได้ จะช่วยให้เราไม่ต้องใส่parameterเยอะๆ ทำให้เราสามารถใช้Method Chaninging ได้

แก้ปัญหา

ทำให้สิ่งที่มีขั้นตอนการสร้างเหมือนๆกันแต่มาสามารถออกมาเป็นobjectที่แตกต่างกันได้

กรณีศึกษา

เราเปิดร้านขายเค้กโดยเค้กแต่ละแบบจะมีรูปแบบที่แตกต่างกันที่ ส่วนประกอบ รูปแบบเค้กให้เลือกว่าต้องการอะไรเพื่อประกอบให้เค้กเสร็จสมบูรณ์แบบที่ต้องการ โดยเค้กก็จะมีหลายแบบ แบบแต่ละเค้กจะมีรูปแบบที่แตกต่างกันเช่นมีแป้งแบบไหน มีรสชาติแบบไหน และมีท๊อปปิ้งอะไรบ้าง



Builder Pattern UML Diagram

Director : เป็นตัวที่เอาไว้ควบคุมขั้นตอนวิธีการทำ

Builder : เป็นตัวกำหนดทุกขั้นตอนที่จะต้องดำเนินการเพื่อให้การสร้างดำเนินไปตามขั้นตอน

ConcreteBuilder : จะสืบทอดมาจากBuilderใช้เพื่อให้ลดความซับซ้อนของBuilder

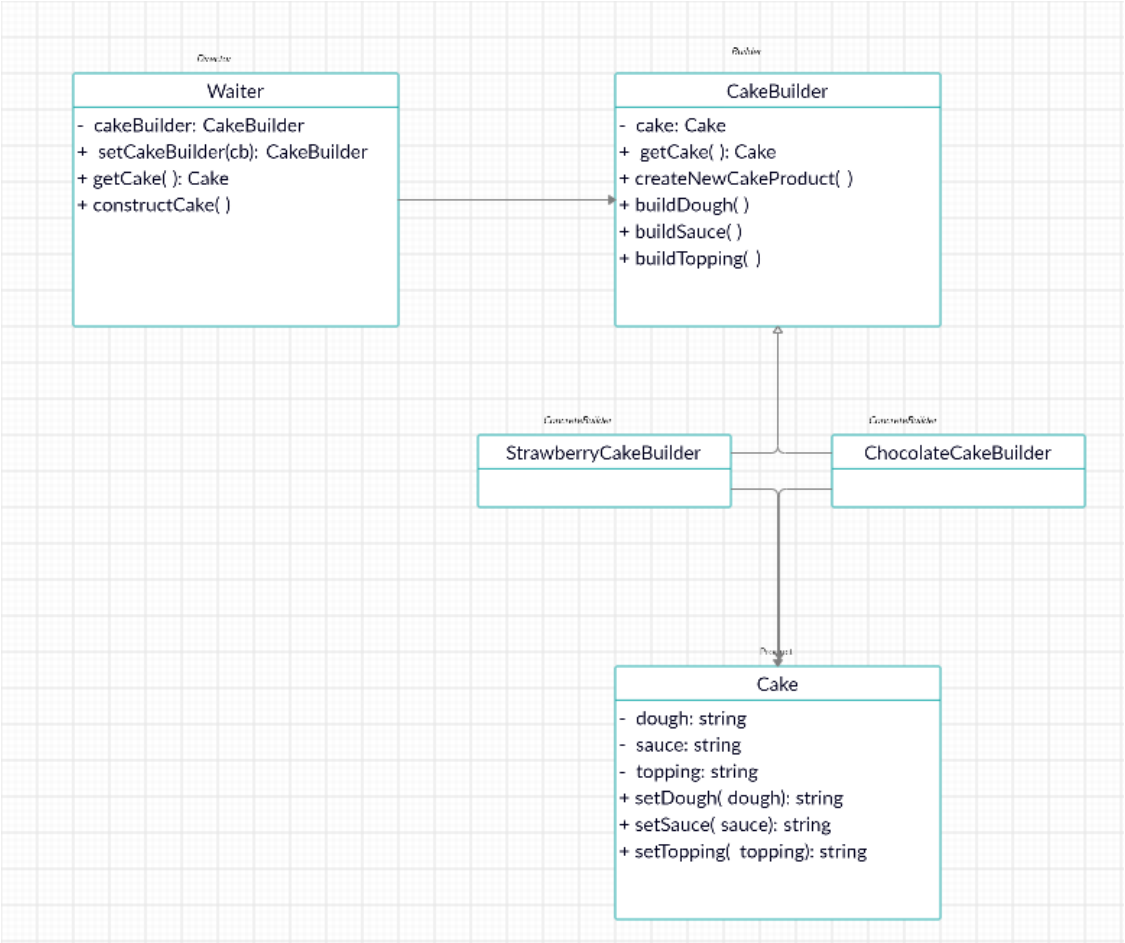
Product : objectที่ถูกสร้างขึ้น

ข้อดี

ลดการเชื่อมกันของcodeทำให้สามรถเปลี่ยนแปลงแก้ไขได้ง่ายมากยิ่งขึ้น

มีความยืดหยุ่นสูงสามารถสร้างBuilderเพิ่มขึ้นได้ในอนาคต

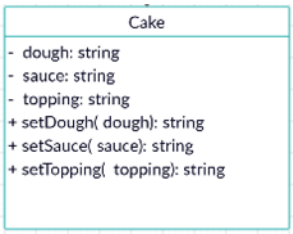
ลดความยาวและความยุ่งยากด้วยBuilder



UMLของกรณีศึกษา

Cake.java

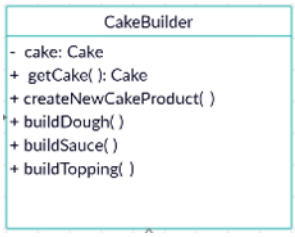
ทำหน้าที่เป็นProductภายในจะมีoptionในกรณีนี้คือแป้ง ซอส และท๊อปปิ้ง




```java
package n1;
class Cake {
    private String dough = "";
    private String sauce = "";
    private String topping = "";

    public void setDough(String dough) {
        this.dough = dough;
    }

    public void setSauce(String sauce) {
        this.sauce = sauce;
    }

    public void setTopping(String topping) {
        this.topping = topping;
    }
}
```

CakeBuilder.java

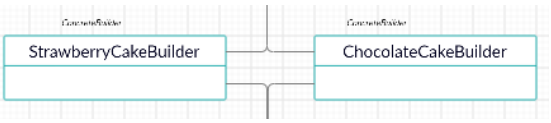
ทำหน้าที่เป็นBuilderซึ่งจะทำให้เราสามารถเพิ่มoptionเข้าไปได้ในส่วนนี้ถือว่าเป็นส่วนสำคัญของBuilder Pattrenซึ่งคือหัวใจหลักที่ทำให้codeมีความซับซ้อนน้อยลง




```java
package n1;
abstract class CakeBuilder {
    protected Cake cake;

    public Cake getCake() {
        return cake;
    }

    public void createNewCakeProduct() {
        cake = new Cake();
    }

    public abstract void buildDough();
    public abstract void buildSauce();
    public abstract void buildTopping();
}
```

StrawberryCakeBuilder.java ChocolateCakeBuilder.java

ทำหน้าที่เป็นConcreteBuilderใช้สำหรับใส่ค่าที่เราต้องการเช่นต้องการเค้กเนื้อนี้ ซอสรสนี้ จะเม็นได้ว่าเราไม่ได้ไปสร้างใหม่เราสามารถนำCakeBuilder.javaมาใช้ได้เลยทำให้classของเราไม่เยอะ

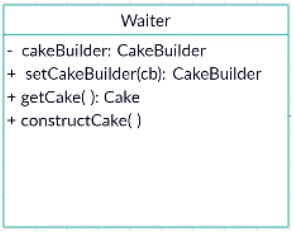



```java
package n1;
class StrawberryCakeBuilder extends CakeBuilder {
    public void buildDough() {
        cake.setDough("Soft Cake");
    }

    public void buildSauce() {
        cake.setSauce("Vanila");
    }

    public void buildTopping() {
        cake.setTopping("Strawberry");
    }
}
```

```java
package n1;
class ChocolateCakeBuilder extends CakeBuilder {
    public void buildDough() {
        cake.setDough("pan baked");
    }

    public void buildSauce() {
        cake.setSauce("Chocolate");
    }

    public void buildTopping() {
        cake.setTopping("Chocolateship");
    }
}
```

Waiter.java

ทำหน้าที่เป็นDirectorควบคุมการทำงานของการสร้างเค้กก้อนนี้



```java
package n1;
class Waiter {
    private CakeBuilder cakeBuilder;

    public void setCakeBuilder(CakeBuilder cb) {
        cakeBuilder = cb;
    }

    public Cake getCake() {
        return cakeBuilder.getCake();
    }

    public void constructCake() {
        cakeBuilder.createNewCakeProduct();
        cakeBuilder.buildDough();
        cakeBuilder.buildSauce();
        cakeBuilder.buildTopping();
    }
}
```

CakeBuilderDemo.java

เป็นMainที่ใช้สั่งเค้ก

```java
package n1;
public class CakeBuilderDemo {
    public static void main(String[] args) {
        Waiter waiter = new Waiter();
        CakeBuilder strawberryCakeBuilder = new StrawberryCakeBuilder();
        CakeBuilder chocolateCakeBuilder = new ChocolateCakeBuilder();

        waiter.setCakeBuilder( strawberryCakeBuilder );
        waiter.constructCake();

        Cake cake = waiter.getCake();
    }
}
```

อ้างอิง

- https://www.borntodev.com/2020/04/05/design-patterns-%E0%B9%81%E0%B8%9A%E0%B8%9A%E0%B8%9E%E0%B8%B7%E0%B9%89%E0

%B8%99%E0%B8%90%E0%B8%B2%E0%B8%99%E0%B8%AA%E0%B8%B8%E0%B8%94-%E0%B9%86/

- http://manit-tree.blogspot.com/2012/07/design-pattern-builder-pattern.html

ใช้เพื่อศึกษาตัวอย่างเนื่องจากตัวอย่างที่ยกมามีความเข้าใจง่าย

- https://www.geeksforgeeks.org/builder-design-pattern/

ใช้เพื่อดูความหมายของแต่ละโครงสร้าง

- https://www.saladpuk.com/beginner-1/design-patterns/creational/builder-pattern

ใช้เพื่อดูภาพรวมทั้งหมดเนื่องจากอธิบายอย่างละเอียดและมีการยกตัวอย่างที่เห็นภาพได้ชัดเจน

- https://sites.google.com/site/softeng07/software-analysis-and-design/patterns-design-and-adventages/structure-of-patterns/creational?tmpl=%2Fsystem%2Fapp%2Ftemplates%2Fprint%2F&showPrintDialog=1

ใช้เนื่องจากนำเอา Builder Pattern UML Diagramมาใช้