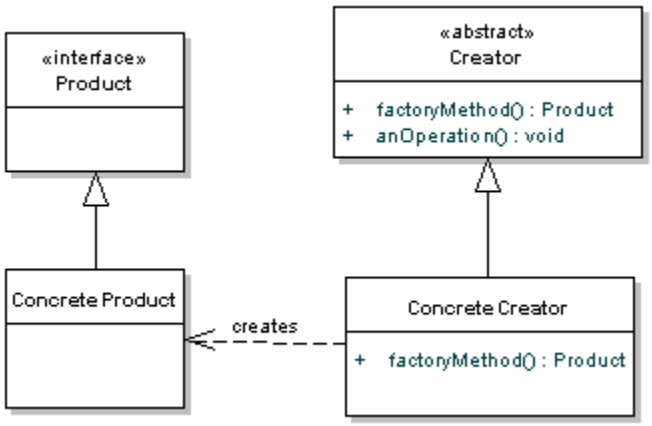
The Factory Pattern

The Factory Pattern อยู่ในกลุ่ม Creational Patterns เป็น Pattern ที่นิยามเมธอดสำหรับสร้างอ็อบเจกต์ของคลาสชนิดหนึ่งๆ แต่ไม่ได้เจาะจงชนิด โดย Factory Pattern จะเป็นรูปแบบที่จำลองโรงงานสร้างสิ่งของขึ้นมา โดยที่เราสามารถสั่งสร้างของได้โดยไม่ต้องสนใจโลจิกการสร้างของชิ้นนั้นๆ ทำให้ง่ายต่อการสร้างobject

การใช้ Factory Patternทำให้สามารถสร้าง object ที่พร้อมสำหรับใช้งาน โดยไม่ต้องระบุ data type ที่แท้จริงของ object ที่ต้องการสร้าง แต่จะให้sub classเป็นส่วนตัดสินใจในการสร้าง object ที่เหมาะสมแทน



โครงสร้าง factory pattern

Creator เป็นAbstract Class ที่มีหน้าที่กำหนดคุณสมบัติของ *Factory* โดยภายในจะมี Abstract Method ที่มีไว้ให้sub class ใช้สร้าง Object เรียกว่า factoryMethod()

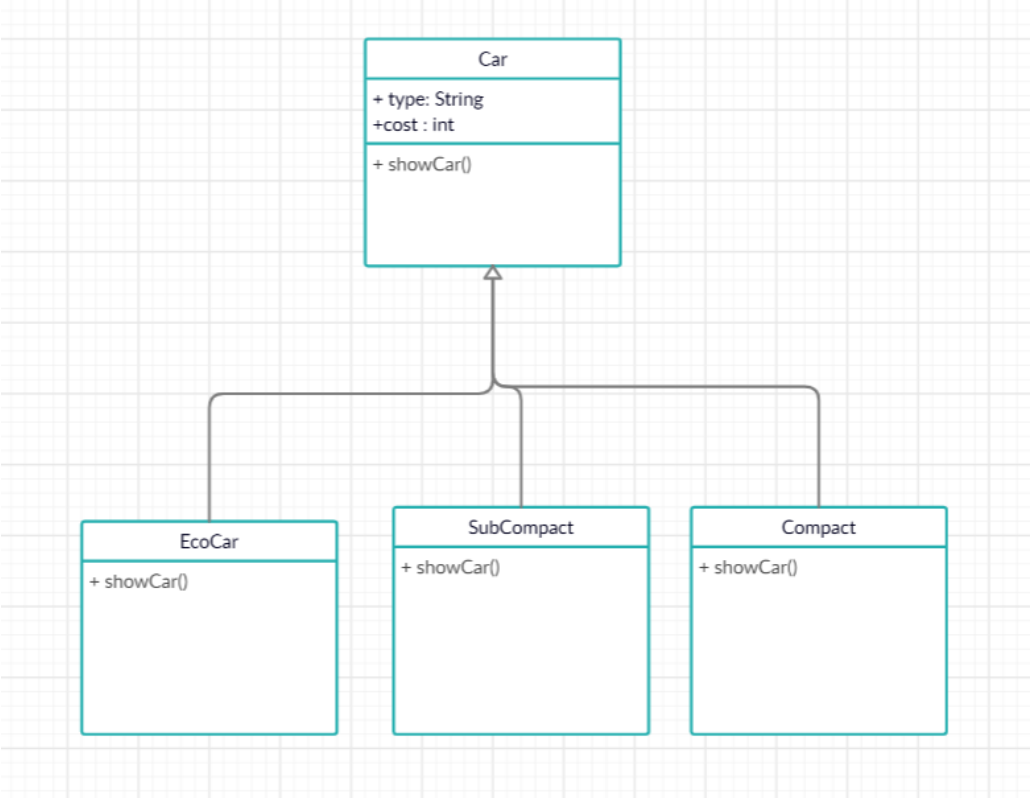
ConcreteCreator เป็นsub classของ Creator เป็น Class ที่มีการoverride abstract method โดยในส่วนนี้จะมีการกำหนดการสร้างObject

Product เป็นClassที่เป็นตัวกำหนดคุณสมบัติของ Object ที่จะถูกสร้างขึ้นจาก ConcreteCreator

ConcreteProduct เป็นsub classของProduct ที่มีการทำงานของ Method ที่ Override มาจาก Product

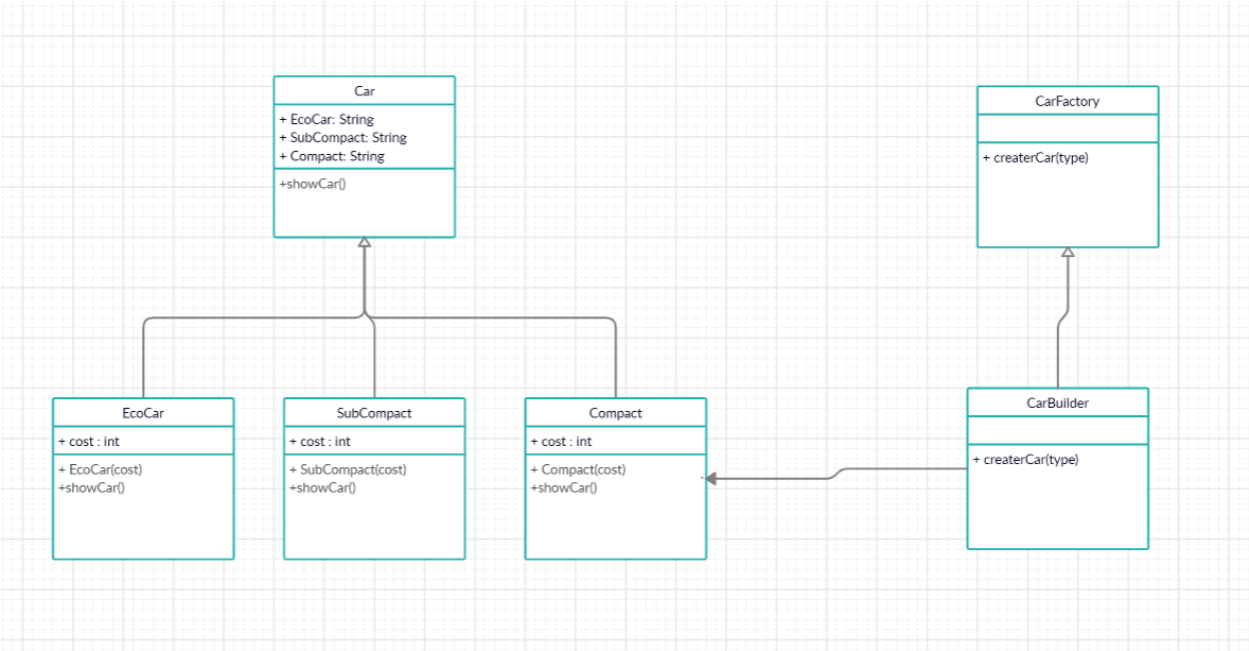
<u>กรณีศึกษา</u>

“โรงงานผลิตรถยนต์” เราต้องการจะสร้างรถยนต์ทั้งหมด 3 แบบ คือ 1.Eco Car 2.Sub Compact 3.Compact ที่มีการกำหนด โดยถ้าดูแบบภาพรวมก็จะเห็นว่าทุกๆแบบนั้นล้วนเป็นรถยนต์เหมือนกัน จะสามารถออกแบบได้ตามภาพด้านล่าง



จากภาพด้านบนเมื่อเรานำไปเขียนcodeจะเห็นได้ว่า ถ้าในอนาคตมีการเปลี่ยนเงื่อนไขในรถต่างๆแบบใหม่ อาจจะทำให้เราต้องทำการแก้ไขหรือเพิ่มcodeใหม่ทั้งหมด ซึ่งจะทำให้codeมีขนาดที่ใหญ่ขึ้นเรื่อยๆ โดยเราสามารถทำการป้องกันปัญหานี้ได้ด้วยการใช้Factory Pattern

ซึ่งเมื่อลองแปลงClass Diagram **“โรงงานผลิตรถยนต์”** ก่อนหน้า ให้ออกมาเป็น Class Diagram ตามรูปโครงสร้างFactory Patternที่กล่าวไว้ก่อนหน้าแล้วจะได้ตามภาพด้านล่าง



โดยจากDiagramข้างต้นจะเห็นว่าเมื่อทำการปรับรูปแบบให้เป็นแบบ The Factory Pattern แล้ว จะสังเกตเห็นได้ว่าในอนาคตหากมีการเพิ่มหรือปรับเปลี่ยนรูปแบบการผลิตรถนั้น ก็สามารถเพิ่มได้โดยไม่จำเป็นต้องแก้ไขcodeทั้งหมด โดยจาก Class Diagram ข้างต้นจะมีรายละเอียด code ดังนี้

Car.java

src > v1 > Car.java > Car > showCar()

```java
package v1;
public interface Car{
    public String EcoCar="EcoCar";
    public String SubCompact="SubCompact";
    public String Compact="Compact";
    public abstract void showCar();
}
```

รูปที่ 1 คลาส Car

EcoCar.java

src > v1 > EcoCar.java > EcoCar > showCar()

```java
package v1;

public class EcoCar implements Car{
    private int cost;
    public EcoCar(int cost){
        this.cost = cost;
    }

    @Override
    public void showCar() {
        System.out.println("Eco car cost "+this.cost);
    }

}
```

รูปที่ 2 คลาส EcoCar

SubCompact.java

src > v1 > SubCompact.java > SubCompact > showCar()

```java
package v1;

public class SubCompact implements Car{
    private int cost;
    public SubCompact(int cost){
        this.cost = cost;
    }

    @Override
    public void showCar() {
        System.out.println("SubCompact car cost "+this.cost);
    }

}
```

รูปที่ 3 คลาส SubCompact

Compact.java

src > v1 > Compact.java > Compact > showCar()

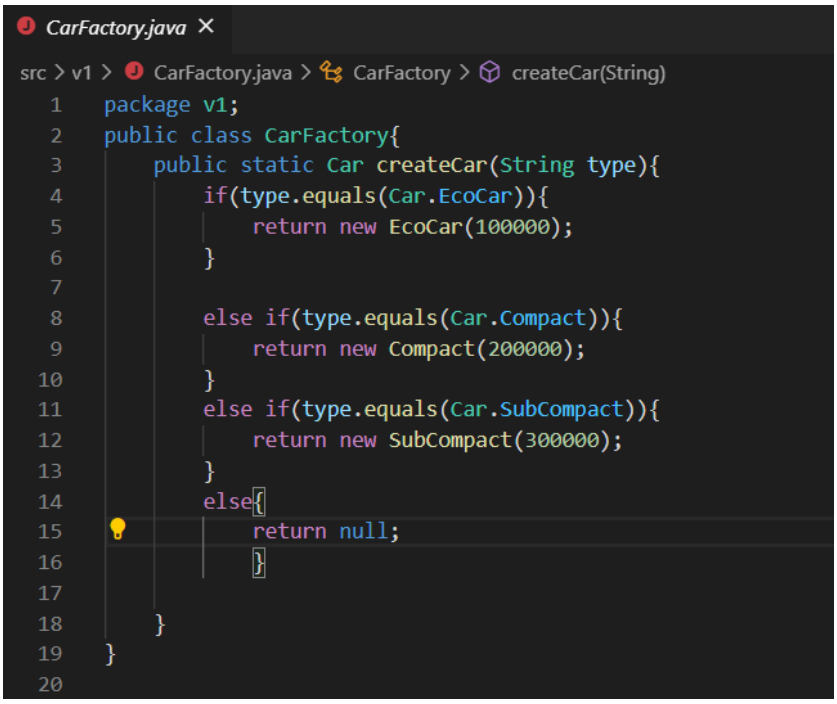
```java
package v1;

public class Compact implements Car{
    private int cost;
    public Compact(int cost){
        this.cost = cost;
    }

    @Override
    public void showCar() {
        System.out.println("Compact car cost "+this.cost);
    }

}
```

รูปที่ 4 คลาส Compact



รูปที่ 5 คลาส CarFactory

รูปที่ 6 คลาส CarBuilder

แหล่งอ้างอิง

1. https://th.wikipedia.org/wiki/%E0%B9%81%E0%B8%9A%E0%B8%9A%E0%B9%82%E0%B8%A3%E0%B8%87%E0%B8%87%E0%B8%B2%E0%B8%99%E0%B8%A2%E0%B9%88%E0%B8%AD
2. https://www.saladpuk.com/beginner-1/design-patterns/creational/factory-method-pattern#undefined-1
3. https://medium.com/thipwriteblog/design-patterns-series-1-factory-method-%E0%B8%8A%E0%B8%A7%E0%B8%99%E0%B8%A1%E0%B8%B2%E0%B8%94%E0%B8%B9%E0%B9%82%E0%B8%A3%E0%B8%87%E0%B8%87%E0%B8%B2%E0%B8%99%E0%B9%84%E0%B8%AD%E0%B8%A8%E0%B8%81%E0%B8%A3%E0%B8%B5%E0%B8%A1-5feb099298e2

นาย ณัฐธัญ ฐติวรสิริ 6030300300