Assignment 1: Factory Pattern

What is Factory Design Pattern?

Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. This pattern is particularly useful when a class doesn't know what sub-classes will be required to create an object, or when a class wants to delegate the responsibility of object creation to its subclasses.

Context: *Car*

In the context of creating different types of cars, the factory design pattern can be used to create different cars without the client code needing to know the specific class of the object being created.

For example, let's say we have a car factory class that is responsible for creating different cars such as Alcazar, Swift, Creta, etc. Each car has its own class and implementation but we can create them through a factory method in the factory class. The client code can use this factory method to create objects of different car without having to know the specific class of the object being created.

Parent Class: Car

Child Class 1: Alcazar

Child Class 2: Swift

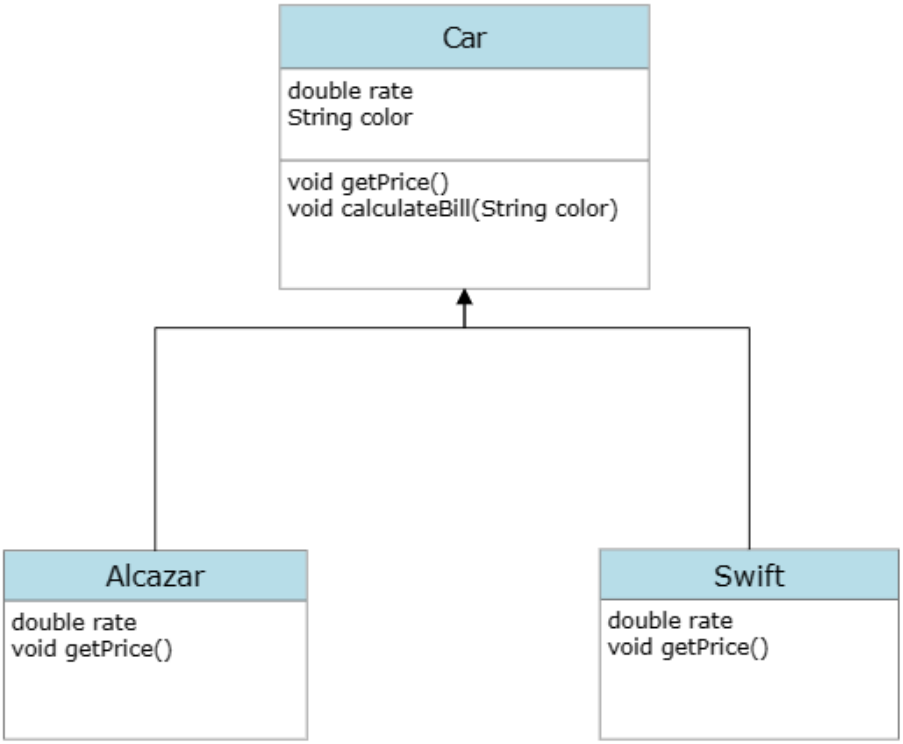

Figure: Inheritance Diagram

Implementation (Code)

Car.java

```
abstract class Car {
    protected double rate;
    public String color;
    abstract void getPrice();
    public void calculateBill(String color){
        double colorValue;
        colorValue = switch (color) {
            case ("Brown") -> 40000;
            case ("White") -> 60000;
            case ("Red") -> 75000;
            case ("Black") -> 90000;
            default -> 38000;
        };
        System.out.println(rate + colorValue);
    }
}
```

Alcazar.java

```
public class Alcazar extends Car {
    public void getPrice() {
        rate = 1500000;
    }
}
```

Swift.java

```
public class Swift extends Car {
    public void getPrice() {
        rate = 500000;
    }
}
```

GetCarFactory.java

```
public class GetCarFactory {
    public Car getCar(String carName){
        if (carName == null){
            return null;
        }
        if (carName.equalsIgnoreCase("Alcazar")){
            return new Alcazar();
        }
        else if (carName.equalsIgnoreCase("Swift")){
            return new Swift();
        }
        return null;
    }
}
```

BuyCar.java

```
import java.io.*;
public class BuyCar {
   public static void main(String[] args) throws IOException {
      GetCarFactory carFactory = new GetCarFactory();

      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

      // Name
      System.out.print("Which car you want to buy [Alcazar, Swift]: ");
      String carName =br.readLine();

      System.out.println("Ok! \nPlease Enter the Specifications you need: ");

      // Color
      System.out.print("Color [ Black,Red,White,Brown ] : ");
      String color = br.readLine();

      Car c = carFactory.getCar(carName);

      System.out.print("Your Bill amount for " + carName + " is ");
      c.getPrice();
      c.calculateBill(color);

   }
}
```

Output

```
Which car you want to buy [Alcazar, Swift]: Alcazar
Ok!
Please Enter the Specifications you need :
Color [ Black,Red,White,Brown ] : Black
Your Bill amount for Alcazar is 1590000.0
```

Applicability

1. Use the Factory Method when you don't know beforehand the exact types and dependencies of the objects your code should work with.
2. Use the Factory Method when you want to provide users of your library or framework with a way to extend its internal components.
3. Use the Factory Method when you want to save system resources by reusing existing objects instead of rebuilding them each time.