Name: -Aditya Raj Roll No: - 11212714 Section: - A1

Design Patterns
W5A1

Topic: - WAP to implement the working of Factory Design Pattern in any Programming Language.

A Factory Pattern or Factory Method Pattern says that just **define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.** In other words, subclasses are responsible to create the instance of the class.

The Factory Method Pattern is also known as **Virtual Constructor.**

Advantage of Factory Design Pattern

- Factory Method Pattern allows the sub-classes to choose the type of objects to create.
- It promotes the **loose-coupling** by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

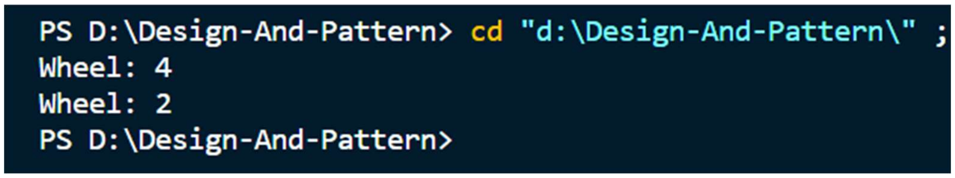
Usage of Factory Design Pattern

- When a class doesn't know what sub-classes will be required to create
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.

Code: -

```
abstract class Vehicle {
  public abstract int getWheel();

  public String toString() {
    return "Wheel: " + this.getWheel();
  }
 }

 class Car extends Vehicle {
  int wheel;

  Car(int wheel) {
    this.wheel = wheel;
  }

  @Override
  public int getWheel() {
    return this.wheel;
  }
}

class Bike extends Vehicle {
  int wheel;

  Bike(int wheel) {
    this.wheel = wheel;
  }

  @Override
  public int getWheel() {
    return this.wheel;
  }

}

class VehicleFactory {
  public static Vehicle getInstance(String type, int wheel) {
    if(type == "car") {
      return new Car(wheel);
    } else if(type == "bike") {
      return new Bike(wheel);
    }

    return null;
  }
}

public class FactoryPatternExample {

  public static void main(String[] args) {
    Vehicle car = VehicleFactory.getInstance("car", 4);
    System.out.println(car);

    Vehicle bike = VehicleFactory.getInstance("bike", 2);
    System.out.println(bike);
    }

  }
```

Output: -



```
PS D:\Design-And-Pattern> cd "d:\Design-And-Pattern\" ;
Wheel: 4
Wheel: 2
PS D:\Design-And-Pattern>
```