A design pattern provides a general reusable solution for the common problems that occur in software design. The pattern typically shows relationships and interactions between classes or objects.

1)Creational design patterns

These design patterns are all about class instantiation or object creation.

These design patterns are used when a decision must be made at the time of instantiation of a class (i.e. creating an object of a class).

2)Structural design patterns

Structural design patterns are concerned with how classes and objects can be composed, to form larger structures.

The structural design patterns simplify the structure by identifying the relationships.

3)Behavioral Design Patterns

Behavioral design patterns are concerned with the interaction and communication of objects.

In these design patterns, the interaction between the objects should be in such a way that they can easily talk to each other and still should be loosely coupled.

# Factory Method Pattern

* It is a creational design pattern
* It is used when we have multiple sub-classes of the super class and based on input, we want to return the one class instance.
* We get that specific instance through another class Factory class.
* It removes the instantiation of actual implementation from the client code to the factory class(because client may not be aware of the instantiation of the concrete class)
* Super class can be abstract class or the interface.
* Factory class has a static method which returns the instance of subclass based on the input.

**package** factorydesignpattern;

**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** FactoryDesignPattern {

**public** **static** **void** main(String[] args) {

// Client using the Factory method

Vehicle car = VehicleFactory.*getInstance*("car", 4);

System.***out***.println(car);

Vehicle bike = VehicleFactory.*getInstance*("bike", 2);

System.***out***.println(bike);

}

}

Real-time examples:

This design pattern has been widely used in JDK, such as

1. getInstance() method of java.util.Calendar, NumberFormat, and ResourceBundle uses factory method design pattern.

2. All the wrapper classes like Integer, Boolean etc, in Java uses this pattern to evaluate the values using valueOf() method.