**Singleton Design Pattern**

A **singleton class** is a design pattern that ensures a class has only one instance and provides a global point of access to that instance. It's commonly used when you need to control access to shared resources, such as a configuration object, a logging service, or a database connection, where multiple instances would be wasteful or could lead to inconsistent results.

**Key Characteristics:**

1. **Single Instance**: The class will have only one instance throughout the lifetime of the application.
2. **Global Access Point**: It provides a global point to access that single instance.

**When to Use Singleton Class:**

1. **Global Access Point**: You need to provide a global point of access to an object.
2. **Single Shared Resource**: You have a resource (e.g., database connection, configuration, etc.) that should only be accessed by one instance across your entire application.
3. **Memory Efficiency**: You want to ensure that only one instance of a class is created, saving memory or other resources.

**Common Use Cases:**

* **Logger classes**: Logging services are often implemented as singletons to ensure that all parts of the program log through a single instance.
* **Configuration objects**: A singleton can hold the application's configuration settings, so that all components can access them in a consistent way.
* **Database connections**: You may want to ensure only one connection is active throughout the life of an application.

**package codingquestion;**

**class Singleton1{**

**private static Singleton1 s;**

**private Singleton1() {**

**}**

**public static Singleton1 getSingleTon() {**

**if(s==null) {**

**s= new Singleton1();**

**return s;**

**}**

**else**

**return s;**

**}**

**}**

**public class Singleton {**

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

**Singleton1 s1 =Singleton1.getSingleTon();**

**Singleton1 s2 =Singleton1.getSingleTon();**

**System.out.println(s1==s2);**

**}**

**}**

**Factory Method Pattern:-**

The Factory Method pattern is a creational design pattern that provides an interface for creating objects in a super class, but allows subclasses to alter the type of objects that will be created. It allows the instantiation of objects without specifying the exact class of object that will be created. This pattern is particularly useful when the exact type of the object is not known until runtime or when the system needs to be independent of how the objects are created.

**Typical Usage of the Factory Method Pattern:**

1. **When a class cannot anticipate the type of objects it needs to create**:
   * The Factory Method is often used when a class needs to create instances of objects of different types, but the exact type is determined at runtime, or based on user input, configuration, or other variables.
2. **To promote loose coupling**:
   * It decouples the client class from the concrete classes it uses. The client class interacts only with the factory method, not the specific implementation, thereby making the system more flexible and extensible.

### Example Scenario:

Consider an application where different types of documents (PDF, Word, and Excel) need to be created, but the application should remain unaware of the exact type of document being generated. The Factory Method pattern can be used to define a factory interface and different subclasses that implement the document creation process.

//product

interface Document{

void open();

}

//concrete creation of Product

class PdfDocument implements Document{

@Override

public void open() {

// TODO Auto-generated method stub

System.out.println("Create pdf Document");

}}

class ExcelDocument implements Document{

@Override

public void open() {

// TODO Auto-generated method stub

System.out.println("Create Excel Document");

}}

class WordDocument implements Document{

@Override

public void open() {

// TODO Auto-generated method stub

System.out.println("Create Word Document");

}}

//abstract factory method

abstract class DocumentFactory {

public abstract Document createDocument();

}

class CreatePdfDocument extends DocumentFactory{

@Override

public Document createDocument() {

// TODO Auto-generated method stub

return new PdfDocument();

}

}

class CreateExcelDocument extends DocumentFactory{

@Override

public Document createDocument() {

// TODO Auto-generated method stub

return new ExcelDocument();

}

}

class CreateWordDocument extends DocumentFactory{

@Override

public Document createDocument() {

// TODO Auto-generated method stub

return new WordDocument();

}

}

public class FactoryMethodDesignPattern {

public static void main(String[] args) {

DocumentFactory pdf = new CreatePdfDocument();

Document pdfOpen =pdf.createDocument();

pdfOpen.open();

DocumentFactory excel = new CreateExcelDocument();

Document ExcelOpen =excel.createDocument();

ExcelOpen.open();

DocumentFactory word = new CreateWordDocument();

Document wordOpen =word.createDocument();

wordOpen.open();}}