Factory Method Design Pattern

The Factory Method Pattern 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. It delegates the responsibility of object instantiation to the subclasses, enabling flexibility in the types of objects that are instantiated.

Use Case:

Imagine you are developing a logistics system. There are different types of transportation modes like trucks, ships, and airplanes. The logistics system should be able to create the correct transportation mode based on the context (e.g., land, sea, or air). The Factory Method Pattern can be used to delegate this responsibility.

Components:

1. Product Interface (Transport): Declares the interface for the objects that the factory will create.
2. Concrete Products (Truck, Ship): Implement the Transport interface.
3. Creator Class (Logistics): Declares the factory method that returns a Transport object.
4. Concrete Creators (RoadLogistics, SeaLogistics): Implement the factory method to return a specific type of Transport.

Example: Logistics System

1. Product Interface (Transport):

```java
public interface Transport {
    void deliver();
}
```

2. Concrete Products (Truck, Ship):

```java
// Concrete Product: Truck
```

```
public class Truck implements Transport {
    @Override
    public void deliver() {
        System.out.println("Delivering by land in a truck.");
    }
}

// Concrete Product: Ship
public class Ship implements Transport {
    @Override
    public void deliver() {
        System.out.println("Delivering by sea in a ship.");
    }
}
```

3. Creator Class (Logistics):

```java
public abstract class Logistics {
    // Factory Method
    public abstract Transport createTransport();

    // Some other business logic that uses the product
    public void planDelivery() {
        Transport transport = createTransport();
        transport.deliver();
    }
```

}

```

4. Concrete Creators (RoadLogistics, SeaLogistics):

```java
// Concrete Creator: RoadLogistics
public class RoadLogistics extends Logistics {
    @Override
    public Transport createTransport() {
        return new Truck();  // Creates a truck for road delivery
    }
}

// Concrete Creator: SeaLogistics
public class SeaLogistics extends Logistics {
    @Override
    public Transport createTransport() {
        return new Ship();  // Creates a ship for sea delivery
    }
}
```

5. Client Code:

```java
public class FactoryMethodDemo {
    public static void main(String[] args) {
        // Plan a road delivery

```
        Logistics roadLogistics = new RoadLogistics();
        roadLogistics.planDelivery();  // Outputs: Delivering by land in a truck.

        // Plan a sea delivery
        Logistics seaLogistics = new SeaLogistics();
        seaLogistics.planDelivery();  // Outputs: Delivering by sea in a ship.
    }
}
```

Output:

```
Delivering by land in a truck.
Delivering by sea in a ship.
```

Key Points:

- Factory Method: The Factory Method Pattern allows the instantiation of objects to be delegated to subclasses, promoting flexibility and adherence to the Open/Closed Principle.
- Decoupling: The pattern decouples the client code from the specific classes of products, making the system more extensible.
- When to Use: This pattern is ideal when the exact type of object to create is determined at runtime or when subclasses are responsible for the creation of particular instances.

When to Use the Factory Method Pattern:

1. When you need to delegate the responsibility of instantiating specific types of objects to subclasses.
2. When the exact type of object is not known until runtime.

3. When you want to provide flexibility for creating objects while still adhering to a common interface.