**Report: Factory Method and Abstract Factory Patterns**

Saqyp Dias

SE-2436

# 1. Introduction

This report demonstrates the implementation of two key creational design patterns in Java: the Factory Method and the Abstract Factory. The project is divided into two main parts: logistics application using the Factory Method, and user interface components using the Abstract Factory.

# 2. Factory Method Pattern

The Factory Method pattern is used to define an interface for creating objects, but lets subclasses decide which class to instantiate. In this project, the Factory Method is applied to the creation of transport objects.

## Code Example

Transport.java

1. package com.example.logistics;  
     
   public interface Transport {  
    void deliver();  
   }

Packaging.java

1. package com.example.logistics;  
     
   public interface Packaging {  
    void pack();  
   }

Truck.java

1. package com.example.logistics;  
     
   public class Truck implements Transport {  
    @Override  
    public void deliver() {  
    System.*out*.println("delivery on Truck");  
    }  
   }

Ship.java

1. package com.example.logistics;  
     
   public class Ship implements Transport {  
    @Override  
    public void deliver() {  
    System.*out*.println("delivery on Ship ");  
    }  
   }

LogisticsFactoru.java

1. package com.example.logistics;  
     
   public abstract class LogisticsFactory {  
    public abstract Transport createTransport();  
    public abstract Packaging createPackaging();  
   }

RoadLogisticsFactory.java

1. package com.example.logistics;  
     
   public class RoadLogisticsFactory extends LogisticsFactory {  
    @Override  
    public Transport createTransport() {  
    return new Truck();  
    }  
     
    @Override  
    public Packaging createPackaging() {  
    return new RoadPackaging();  
    }  
   }

SeaLogisticsFactory.java

1. package com.example.logistics;  
     
   public class SeaLogisticsFactory extends LogisticsFactory {  
    @Override  
    public Transport createTransport() {  
    return new Ship();  
    }  
     
    @Override  
    public Packaging createPackaging() {  
    return new SeaPackaging();  
    }  
   }

SeaPackaging.java

1. package com.example.logistics;  
     
   public class SeaPackaging implements Packaging {  
    @Override  
    public void pack() {  
    System.*out*.println("Using a waterproof sea packaging.");  
    }  
   }

RoadPackaging.java

1. package com.example.logistics;  
     
   public class RoadPackaging implements Packaging {  
    @Override  
    public void pack() {  
    System.*out*.println("Using a durable road packaging.");  
    }  
   }

# 2. Main Class

1. import com.example.logistics.\*;  
     
    public class Main {  
    public static void main(String[] args) {  
     
   // RoadFabric  
    LogisticsFactory roadFactory = new RoadLogisticsFactory();  
    Transport roadTransport = roadFactory.createTransport();  
    Packaging roadPackaging = roadFactory.createPackaging();  
     
    roadTransport.deliver();  
    roadPackaging.pack();  
     
    System.*out*.println("--------------------");  
     
   // SeaFabric  
    LogisticsFactory seaFactory = new SeaLogisticsFactory();  
    Transport seaTransport = seaFactory.createTransport();  
    Packaging seaPackaging = seaFactory.createPackaging();  
     
    seaTransport.deliver();  
    seaPackaging.pack();  
     
     
     
    }  
    }

# 5. Conclusion

The implementation clearly demonstrates the work with the Factory Method and Abstract Factory patterns. The code adheres to the principles of Clean Code by maintaining meaningful names, separation of concerns, and readability.

# 6. GitHub Repository

*https://github.com/aaituu/SoftDP/tree/main/DISDOFT2*