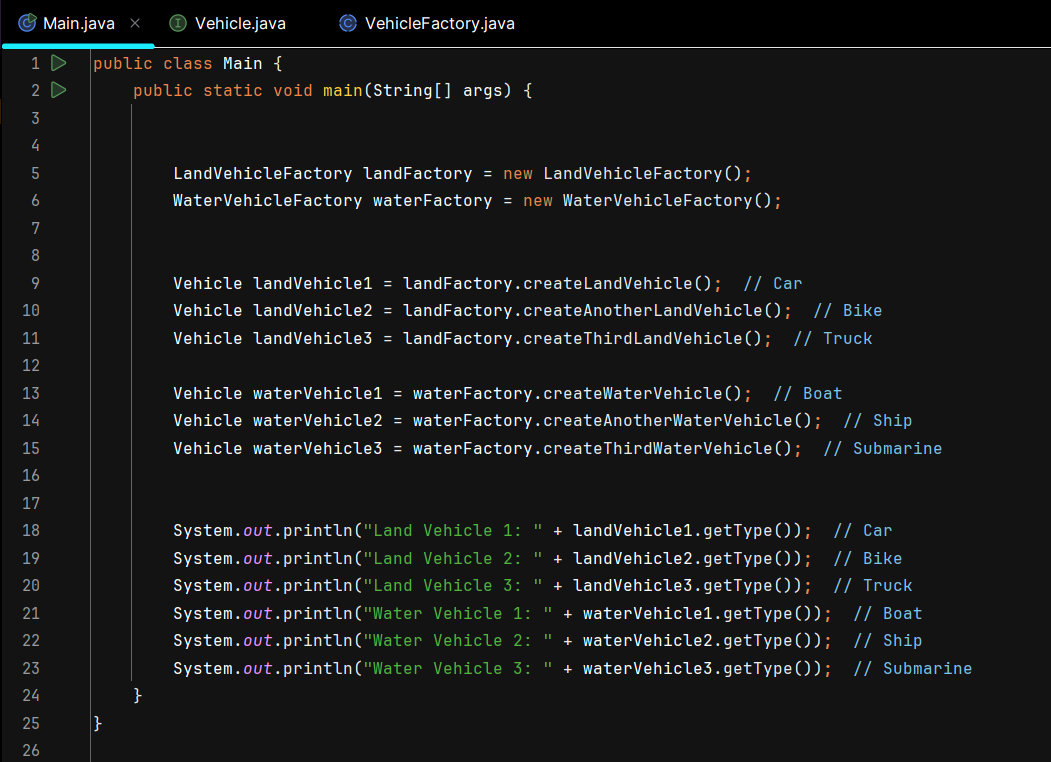
Maksatova Akniyet Manatkyzy

SE-2409

Software Design Patterns

Assignment 2



**Class Initialization:**

LandVehicleFactory landFactory = new LandVehicleFactory();

WaterVehicleFactory waterFactory = new WaterVehicleFactory();

* Two factory objects are created: landFactory for creating land vehicles and waterFactory for creating water vehicles. These are instantiated from their respective classes (LandVehicleFactory and WaterVehicleFactory).

**Creating Land Vehicles:**

Vehicle landVehicle1 = landFactory.createLandVehicle(); // Car

Vehicle landVehicle2 = landFactory.createAnotherLandVehicle(); // Bike

Vehicle landVehicle3 = landFactory.createThirdLandVehicle(); // Truck

* The landFactory is used to create three different types of land vehicles:
  + **Land Vehicle 1:** Created by createLandVehicle(), which returns a Car.
  + **Land Vehicle 2:** Created by createAnotherLandVehicle(), which returns a Bike.
  + **Land Vehicle 3:** Created by createThirdLandVehicle(), which returns a Truck.

**Creating Water Vehicles:**

Vehicle waterVehicle1 = waterFactory.createWaterVehicle(); // Boat

Vehicle waterVehicle2 = waterFactory.createAnotherWaterVehicle(); // Ship

Vehicle waterVehicle3 = waterFactory.createThirdWaterVehicle(); // Submarine

* The waterFactory is used to create three different types of water vehicles:
  + **Water Vehicle 1:** Created by createWaterVehicle(), which returns a Boat.
  + **Water Vehicle 2:** Created by createAnotherWaterVehicle(), which returns a Ship.
  + **Water Vehicle 3:** Created by createThirdWaterVehicle(), which returns a Submarine.

**Displaying Vehicle Types:**

System.out.println("Land Vehicle 1: " + landVehicle1.getType()); // Car

System.out.println("Land Vehicle 2: " + landVehicle2.getType()); // Bike

System.out.println("Land Vehicle 3: " + landVehicle3.getType()); // Truck

System.out.println("Water Vehicle 1: " + waterVehicle1.getType()); // Boat

System.out.println("Water Vehicle 2: " + waterVehicle2.getType()); // Ship

System.out.println("Water Vehicle 3: " + waterVehicle3.getType()); // Submarine

* For each vehicle, the getType() method is called to output the type of each vehicle:
  + **Land Vehicle 1:** Outputs the type of the first land vehicle (Car).
  + **Land Vehicle 2:** Outputs the type of the second land vehicle (Bike).
  + **Land Vehicle 3:** Outputs the type of the third land vehicle (Truck).
  + **Water Vehicle 1:** Outputs the type of the first water vehicle (Boat).
  + **Water Vehicle 2:** Outputs the type of the second water vehicle (Ship).
  + **Water Vehicle 3:** Outputs the type of the third water vehicle (Submarine).

A screenshot of a computer program

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.

**1. Vehicle Interface**

public interface Vehicle {

String getType();

}

* **Vehicle Interface**: This defines a contract for the getType() method, which will be implemented by all concrete vehicle classes. This ensures that every vehicle has a way to describe itself by returning its type.

**2. Concrete Vehicle Classes**

class Car implements Vehicle {

@Override

public String getType() {

return "Land Vehicle: Car";

}

}

class Bike implements Vehicle {

@Override

public String getType() {

return "Land Vehicle: Bike";

}

}

class Truck implements Vehicle {

@Override

public String getType() {

return "Land Vehicle: Truck";

}

}

class Boat implements Vehicle {

@Override

public String getType() {

return "Water Vehicle: Boat";

}

}

class Ship implements Vehicle {

@Override

public String getType() {

return "Water Vehicle: Ship";

}

}

class Submarine implements Vehicle {

@Override

public String getType() {

return "Water Vehicle: Submarine";

}

}

* **Concrete Vehicle Classes (Car, Bike, Truck, Boat, Ship, Submarine)**: These classes implement the Vehicle interface and provide their own implementation of the getType() method. Each class represents a different type of vehicle (land or water).

**3. Factory Method Pattern**

* The Factory Method Pattern allows for the creation of objects without specifying the exact class of the object that will be created. Each concrete class (Car, Boat, Ship, etc.) implements the getType() method, but they are created through a factory method, rather than directly instantiating them.

**4. Benefits**

* This design pattern is used when:
  + You want to create families of related objects without specifying their exact concrete classes.
  + You want to delegate the responsibility of object creation to a factory.

A screenshot of a computer program

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.

**1. Abstract VehicleFactory Class**

abstract class VehicleFactory {

public abstract Vehicle createLandVehicle();

public abstract Vehicle createWaterVehicle();

}

* **VehicleFactory Class**: This is an abstract class that defines two abstract methods, createLandVehicle() and createWaterVehicle(). These methods are meant to be implemented by concrete factories to create specific types of vehicles (land and water).

**2. LandVehicleFactory Class**

class LandVehicleFactory extends VehicleFactory {

@Override

public Vehicle createLandVehicle() {

return new Car();

}

@Override

public Vehicle createWaterVehicle() {

return null;

}

public Vehicle createAnotherLandVehicle() {

return new Bike();

}

public Vehicle createThirdLandVehicle() {

return new Truck();

}

}

* **LandVehicleFactory Class**: This class extends VehicleFactory and provides implementations for creating land vehicles, like Car, Bike, and Truck. For the createWaterVehicle() method, it returns null, as this factory only deals with land vehicles.
  + **createLandVehicle()**: Creates a Car.
  + **createAnotherLandVehicle()**: Creates a Bike.
  + **createThirdLandVehicle()**: Creates a Truck.

**3. WaterVehicleFactory Class**

class WaterVehicleFactory extends VehicleFactory {

@Override

public Vehicle createLandVehicle() {

return null;

}

@Override

public Vehicle createWaterVehicle() {

return new Boat();

}

public Vehicle createAnotherWaterVehicle() {

return new Ship();

}

public Vehicle createThirdWaterVehicle() {

return new Submarine();

}

}

* **WaterVehicleFactory Class**: Similar to the LandVehicleFactory, this class creates water vehicles, like Boat, Ship, and Submarine. The createLandVehicle() method returns null as it only produces water vehicles.
  + **createWaterVehicle()**: Creates a Boat.
  + **createAnotherWaterVehicle()**: Creates a Ship.
  + **createThirdWaterVehicle()**: Creates a Submarine.

**4. Key Differences Between the Two Factories**

* The **LandVehicleFactory** creates land vehicles (Car, Bike, and Truck) but does not create water vehicles.
* The **WaterVehicleFactory** creates water vehicles (Boat, Ship, and Submarine) but does not create land vehicles.

**5. Benefits of the Abstract Factory Pattern**

* This pattern provides a way to create families of related objects (land and water vehicles) without specifying their exact classes. Each factory produces a set of objects, either land or water vehicles, ensuring that objects from one family (land or water) are used together.
* It also abstracts away the creation of individual vehicle types, ensuring that clients do not need to know the details of how vehicles are created.