**Tazhibayev Sultanbay**

**SE-2326**

**SDP**

**Assignment 1**

***Factory Method Pattern***

**ConcreteFactories**

package FactoryMethodPattern;

public class ConcreteFactories {  
 public static class LandTransportFactory extends TransportFactory{  
 @Override  
 public Transport createTransport() {  
 return new ConcreteTransport.Car();  
 }  
 }  
  
 public static class WaterTransportFactory extends TransportFactory{  
 @Override  
 public Transport createTransport() {  
 return new ConcreteTransport.Ship();  
 }  
 }  
  
 public static class AirTransportFactory extends TransportFactory{  
 @Override  
 public Transport createTransport() {  
 return new ConcreteTransport.Airplane();  
 }  
 }  
}

**Concrete Transport**

package FactoryMethodPattern;  
  
public class ConcreteTransport {  
 public static class Car implements Transport{  
 @Override  
 public void deliver() {  
 System.*out*.println("Delivery by land in a car");  
 }  
 }  
  
 public static class Airplane implements Transport{  
 @Override  
 public void deliver() {  
 System.*out*.println("Delivery by air in a airplane");  
 }  
 }  
  
 public static class Ship implements Transport{  
 @Override  
 public void deliver() {  
 System.*out*.println("Delivery by water in a ship.");  
 }  
 }  
}

**Main**

package FactoryMethodPattern;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 TransportFactory factory = null;  
  
 System.*out*.println("Select delivery method (land, water, air): ");  
 String deliveryType = scanner.nextLine();  
  
 switch (deliveryType.toLowerCase()) {  
 case "land":  
 factory = new ConcreteFactories.LandTransportFactory();  
 break;  
 case "water":  
 factory = new ConcreteFactories.WaterTransportFactory();  
 break;  
 case "air":  
 factory = new ConcreteFactories.AirTransportFactory();  
 break;  
 default:  
 System.*out*.println("Invalid delivery type.");  
 break;  
 }  
  
 if (factory != null) {  
 factory.manageDelivery();  
 }  
  
 scanner.close();  
 }  
}

**Transport**

package FactoryMethodPattern;  
  
public interface Transport {  
 void deliver();  
}

**TransportFactory**

package FactoryMethodPattern;  
  
public abstract class TransportFactory {  
 public abstract Transport createTransport();  
  
 public void manageDelivery(){  
 Transport transport = createTransport();  
 transport.deliver();  
 }  
}

***Logger***

**Logger**

package Logger;  
  
public class Logger {  
 // this code is private static variable to hold the single instance of the Logger.Logger  
 private static Logger *instance*;  
  
 // this code is private constructor to prevent instantiation from outside  
 private Logger() {}  
  
 // this code is public static method to provide access to the single instance  
 public static Logger getInstance() {  
 if (*instance* == null) { // if object is not created  
 *instance* = new Logger(); // to create a new object  
 }  
 return *instance*; // to return a created object  
 }  
  
 // this Method to log messages  
 public void LogToConsole(String message) {  
 System.*out*.println("Log message: " + message);  
 }  
}

**Main**

package Logger;  
  
import Logger.Logger;  
  
import java.util.Scanner;  
// i import Scanner  
public class Main {  
 public static void main(String[] args) {  
 //there i am adding scanner  
 Scanner scanner = new Scanner(System.*in*);  
 // there i will get the single instance of Logger.Logger  
 Logger logger = Logger.*getInstance*();  
  
 // there i will log messages using the Logger.Logger instance  
 // this code is for writing some message to terminal  
 String Message = scanner.nextLine();  
 // with this i will go tho the Logger.Logger and will print message to the terminal  
 logger.LogToConsole(Message);  
 }  
}

***BuilderPattern***

**House**

package BuilderPattern;  
  
public class House {  
 private int numRooms;  
 private int numFloors;  
 private boolean hasPool;  
 private boolean hasGarage;  
  
 private House(HouseBuilder builder) {  
 this.numRooms = builder.numRooms;  
 this.numFloors = builder.numFloors;  
 this.hasPool = builder.hasPool;  
 this.hasGarage = builder.hasGarage;  
 }  
  
 @Override  
 public String toString() {  
 return "House with " + numRooms + " room(s), " + numFloors + " floor(s), " +  
 (hasPool ? "a pool" : "no pool") + ", " +  
 (hasGarage ? "a garage" : "no garage");  
 }  
  
  
 public static class HouseBuilder {  
 private int numRooms;  
 private int numFloors;  
 private boolean hasPool;  
 private boolean hasGarage;  
  
 public HouseBuilder setNumRooms(int numRooms) {  
 this.numRooms = numRooms;  
 return this;  
 }  
  
 public HouseBuilder setNumFloors(int numFloors) {  
 this.numFloors = numFloors;  
 return this;  
 }  
  
 public HouseBuilder setHasPool(boolean hasPool) {  
 this.hasPool = hasPool;  
 return this;  
 }  
  
 public HouseBuilder setHasGarage(boolean hasGarage) {  
 this.hasGarage = hasGarage;  
 return this;  
 }  
  
 public House build() {  
 return new House(this);  
 }  
 }  
}

**Main**

package BuilderPattern;  
public class Main {  
 public static void main(String[] args) {  
 House house = new House.HouseBuilder()  
 .setNumRooms(3)  
 .setNumFloors(2)  
 .setHasPool(true)  
 .setHasGarage(false)  
 .build();  
  
 System.*out*.println(house);  
 }  
}

***PrototypePattern***

**Main**

package ProtatypePattern;  
  
public class Main {  
 public static void main(String[] args) {  
 Product originalProduct = new Product("Iphone 13", 1200.00);  
  
 Product clonedProduct = (Product) originalProduct.clone();  
  
 clonedProduct.setPrice(900.00);  
  
 System.*out*.println("Original Product: " + originalProduct);  
 System.*out*.println("Cloned Product: " + clonedProduct);  
 }  
}

**Product**

package ProtatypePattern;  
  
public class Product implements Cloneable{  
 private String name;  
 private double price;  
  
 public Product(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
  
 // Getter and Setter methods  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 this.price = price;  
 }  
  
 @Override  
 public Object clone() {  
 try {  
 return super.clone();  
 } catch (CloneNotSupportedException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 @Override  
 public String toString() {  
 return "Product{name='" + name + "', price=" + price + "}";  
 }  
}

***AbstractFactoryPattern***

**Application**

package AbstractFactoryPattern;  
  
public class Application {  
 private Button button;  
 private Checkbox checkbox;  
 public Application(GUIFactory factory){  
 button = factory.createButton();  
 checkbox = factory.createCheckbox();  
 }  
 public void paint(){  
 button.paint();  
 checkbox.paint();  
 }  
}

**Button**

package AbstractFactoryPattern;  
  
public interface Button {  
 void paint();  
}

**Checkbox**

package AbstractFactoryPattern;  
  
public interface Checkbox {  
 void paint();  
}

**GUIFactory**

package AbstractFactoryPattern;  
  
public interface GUIFactory {  
 Button createButton();  
 Checkbox createCheckbox();  
  
}

**MacOSButton**

package AbstractFactoryPattern;  
  
public class MacOSButton implements Button{  
 @Override  
 public void paint() {  
 System.*out*.println("You have created MacOSButton.");  
 }  
}

**MacOSCheckbox**

package AbstractFactoryPattern;  
  
public class MacOSCheckbox implements Checkbox{  
 @Override  
 public void paint() {  
 System.*out*.println("You have created MacOSCheckbox.");  
 }  
}

**MacOSFactory**

package AbstractFactoryPattern;  
  
public class MacOSFactory implements GUIFactory{  
 @Override  
 public Button createButton() {  
 return new MacOSButton();  
 }  
  
 @Override  
 public Checkbox createCheckbox() {  
 return new MacOSCheckbox();  
 }  
}

**Main**

package AbstractFactoryPattern;  
  
public class Main {  
 private static Application configureApplication(){  
 Application app;  
 GUIFactory factory;  
 String osName = System.*getProperty*("os.name").toLowerCase();  
 if(osName.contains("mac")){  
 factory = new MacOSFactory();  
 }  
 else{  
 factory = new WindowsFactory();  
 }  
 app = new Application(factory);  
 return app;  
 }  
 public static void main(String[] args){  
 Application app = *configureApplication*();  
 app.paint();  
 }  
}

**WindowsButton**

package AbstractFactoryPattern;  
  
public class WindowsButton implements Button{  
 @Override  
 public void paint() {  
 System.*out*.println("You have created WindowsButton.");  
 }  
}

**WindowsCheckbox**

package AbstractFactoryPattern;  
  
public class WindowsCheckbox implements Checkbox{  
 @Override  
 public void paint() {  
 System.*out*.println("You have created WindowsCheckbox.");  
 }  
}

**WindowsFactory**

package AbstractFactoryPattern;  
  
public class WindowsFactory implements GUIFactory{  
 @Override  
 public Button createButton() {  
 return new WindowsButton();  
 }  
  
 @Override  
 public Checkbox createCheckbox() {  
 return new WindowsCheckbox();  
 }  
}