Министерство образования Республики Беларусь Учреждение образования

«Брестский Государственный технический университет» Кафедра ИИТ

# Лабораторная работа №6

**по дисциплине “Современные платформы программирования”**

**Выполнил:** Студент 3 курса Группы ПО-8

Соколов С.Д.

**Проверил:** Крощенко А.А.

Брест 2024

**Цель работы:** приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Java.

# Ход работы Вариант20

1. **Заводы по производству автомобилей. Реализовать возможность создавать автомобили различных типов на различных заводах.**

Для данного задания я воспользуюсь паттерном Фабричный метод для создания объектов без указания их конкретных классов. Фабричный метод определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. В данном задании паттерн Фабричный метод подходит для создания различных моделей автомобилей с заранее выбранными характеристиками на разных заводах.

Код программы:

**Car.java**

public abstract class Car {  
 protected String model;  
 protected int productionYear;  
 protected String fuelType;  
 protected double engineCapacity;  
 public Car(String model, int productionYear, String fuelType, double engineCapacity) {  
 this.model = model;  
 this.productionYear = productionYear;  
 this.fuelType = fuelType;  
 this.engineCapacity = engineCapacity;  
 }  
 public abstract void printInfo();  
}

**BMW.java**

public class BMW extends Car {  
 public BMW(String model, int productionYear, String fuelType, double engineCapacity) {  
 super(model, productionYear, fuelType, engineCapacity);  
 }  
 @Override  
 public void printInfo() {  
 System.*out*.println("Автомобиль BMW модели " + super.model  
 + "\nГод выпуска: " + super.productionYear  
 + "\nТип топлива: " + super.fuelType  
 + "\nОбъем двигателя: " + super.engineCapacity + " л");  
 }  
}

**Mercedes.java**

public class Mercedes extends Car {  
 public Mercedes(String model, int productionYear, String fuelType, double engineCapacity) {  
 super(model, productionYear, fuelType, engineCapacity);  
 }  
 @Override  
 public void printInfo() {  
 System.*out*.println("Автомобиль Mercedes модели " + super.model  
 + "\nГод выпуска: " + super.productionYear  
 + "\nТип топлива: " + super.fuelType  
 + "\nОбъем двигателя: " + super.engineCapacity + " л");  
 }  
}

**BMWCreator.java**

public class BMWCreator extends CarCreator {  
 @Override  
 public Car createCar() { return new BMW("X5", 2024, "Бензин", 3.0); }  
}

**MercedesCreator.java**

public class MercedesCreator extends CarCreator {  
 @Override  
 public Car createCar() { return new Mercedes("S-Class", 2024, "Дизель", 3.5); }  
}

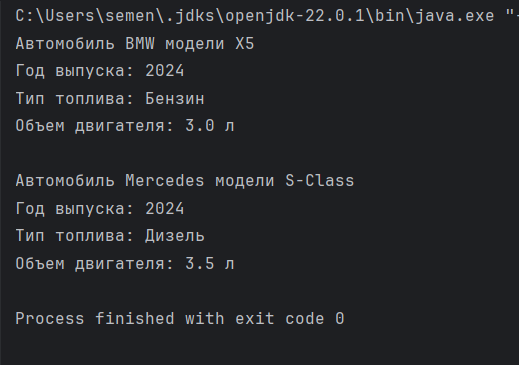
**CarCreator.java**

public abstract class CarCreator {  
 public abstract Car createCar();  
}

**Main.java**

public class Main {  
 public static void main(String[] args) {  
 BMWCreator bmwCreator = new BMWCreator();  
 Car bmw = bmwCreator.createCar();  
 bmw.printInfo();  
 System.*out*.println("");  
 MercedesCreator mercedesCreator = new MercedesCreator();  
 Car mercedes = mercedesCreator.createCar();  
 mercedes.printInfo();  
 }  
}

# Результаты работы программы:

****

**2.** Учетная запись покупателя книжного интернет-магазина. Предусмотреть различные уровни учетки в зависимости от активности покупателя. Дополнительные уровни добавляют функциональные возможности и открывают доступ к уникальным предложениям.

Для этого задания я выбрал паттерн “Стратегия”. Этот паттерн позволяет выбирать поведение объекта на лету, в зависимости от его состояния или других параметров. В данном случае, уровень активности пользователя может определять его уровень учетной записи и соответствующие привилегии.

Код программы:

**AccountLevel.java**

public interface AccountLevel {  
 void applyPrivileges();  
}

**BasicLevel.java**

public class BasicLevel implements AccountLevel {  
 @Override  
 public void applyPrivileges() {  
 System.*out*.println("Применены привилегии базового уровня");  
 }  
}

**PremiumLevel.java**

public class PremiumLevel implements AccountLevel {  
 @Override  
 public void applyPrivileges() {  
 System.*out*.println("Применены привилегии премиум уровня");  
 }  
}

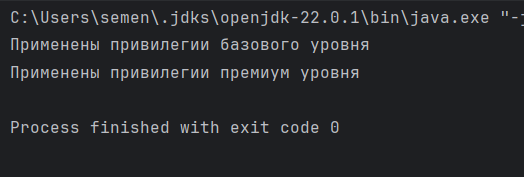
**UserAccaunt.java**

public class UserAccount {  
 private AccountLevel level;  
  
 public UserAccount(AccountLevel level) {  
 this.level = level;  
 }  
  
 public void setLevel(AccountLevel level) {  
 this.level = level;  
 }  
  
 public void applyPrivileges() {  
 level.applyPrivileges();  
 }  
}

**Main.java**

public class Main {  
 public static void main(String[] args) {  
 UserAccount user = new UserAccount(new BasicLevel());  
 user.applyPrivileges();  
  
 user.setLevel(new PremiumLevel());  
 user.applyPrivileges();  
 }  
}

# Результаты работы программы:



**3.** Проект «Банкомат». Предусмотреть выполнение основных операций (ввод пин-кода, снятие суммы, завершение работы) и наличие различных режимов работы (ожидание, аутентификация, выполнение операции, блокировка – если нет денег). Атрибуты: общая сумма денег в банкомате, ID.

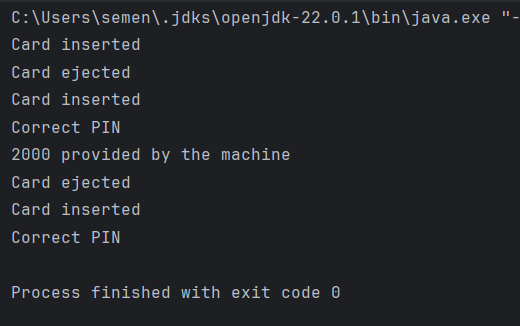
Для данного задания воспользуемся поведенческим паттерном Состояние. Этот паттерн позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. В данном случае, банкомат может находиться в различных состояниях: ожидание, аутентификация, выполнение операции, блокировка.

Код программы:

**Main.java**

interface ATMState {  
 void insertCard();  
 void ejectCard();  
 void insertPin(int pinEntered);  
 void requestCash(int cashToWithdraw);  
}  
  
class ATMMachine {  
 ATMState hasCard;  
 ATMState noCard;  
 ATMState hasCorrectPin;  
 ATMState atmOutOfMoney;  
  
 ATMState atmState;  
 int cashInMachine = 2000;  
 boolean correctPinEntered = false;  
  
 public ATMMachine(){  
 hasCard = new HasCard(this);  
 noCard = new NoCard(this);  
 hasCorrectPin = new HasPin(this);  
 atmOutOfMoney = new NoCash(this);  
  
 atmState = noCard;  
 if(cashInMachine < 0){  
 atmState = atmOutOfMoney;  
 }  
 }  
  
 void setATMState(ATMState newATMState){  
 atmState = newATMState;  
 }  
  
 public void setCashInMachine(int newCashInMachine){  
 cashInMachine = newCashInMachine;  
 }  
  
 public void insertCard() {  
 atmState.insertCard();  
 }  
  
 public void ejectCard() {  
 atmState.ejectCard();  
 }  
  
 public void requestCash(int cashToWithdraw) {  
 atmState.requestCash(cashToWithdraw);  
 }  
  
 public void insertPin(int pinEntered){  
 atmState.insertPin(pinEntered);  
 }  
  
 public ATMState getYesCardState() { return hasCard; }  
 public ATMState getNoCardState() { return noCard; }  
 public ATMState getHasPin() { return hasCorrectPin; }  
 public ATMState getNoCashState() { return atmOutOfMoney; }  
}  
  
class HasCard implements ATMState {  
 ATMMachine atmMachine;  
  
 public HasCard(ATMMachine newATMMachine){  
 atmMachine = newATMMachine;  
 }  
  
 public void insertCard() {  
 System.*out*.println("You can only insert one card at a time");  
 }  
  
 public void ejectCard() {  
 System.*out*.println("Card ejected");  
 atmMachine.setATMState(atmMachine.getNoCardState());  
 }  
  
 public void requestCash(int cashToWithdraw) {  
 System.*out*.println("Enter PIN first");  
 }  
  
 public void insertPin(int pinEntered) {  
 if(pinEntered == 1234){  
 System.*out*.println("Correct PIN");  
 atmMachine.correctPinEntered = true;  
 atmMachine.setATMState(atmMachine.getHasPin());  
 } else {  
 System.*out*.println("Wrong PIN");  
 atmMachine.correctPinEntered = false;  
 System.*out*.println("Card Ejected");  
 atmMachine.setATMState(atmMachine.getNoCardState());  
 }  
 }  
}  
  
class NoCard implements ATMState {  
 ATMMachine atmMachine;  
  
 public NoCard(ATMMachine newATMMachine){  
 atmMachine = newATMMachine;  
 }  
  
 public void insertCard() {  
 System.*out*.println("Card inserted");  
 atmMachine.setATMState(atmMachine.getYesCardState());  
 }  
  
 public void ejectCard() {  
 System.*out*.println("No card to eject");  
 }  
  
 public void requestCash(int cashToWithdraw) {  
 System.*out*.println("Enter card first");  
 }  
  
 public void insertPin(int pinEntered) {  
 System.*out*.println("Enter card first");  
 }  
}  
  
class HasPin implements ATMState {  
 ATMMachine atmMachine;  
  
 public HasPin(ATMMachine newATMMachine){  
 atmMachine = newATMMachine;  
 }  
  
 public void insertCard() {  
 System.*out*.println("You already entered a card");  
 }  
  
 public void ejectCard() {  
 System.*out*.println("Card ejected");  
 atmMachine.setATMState(atmMachine.getNoCardState());  
 }  
  
 public void requestCash(int cashToWithdraw) {  
 if(cashToWithdraw > atmMachine.cashInMachine){  
 System.*out*.println("Not enough cash in machine");  
 System.*out*.println("Card ejected");  
 atmMachine.setATMState(atmMachine.getNoCardState());  
 } else {  
 System.*out*.println(cashToWithdraw + " provided by the machine");  
 atmMachine.setCashInMachine(atmMachine.cashInMachine - cashToWithdraw);  
 System.*out*.println("Card ejected");  
 atmMachine.setATMState(atmMachine.getNoCardState());  
 }  
 }  
  
 public void insertPin(int pinEntered) {  
 System.*out*.println("Already entered PIN");  
 }  
}  
  
class NoCash implements ATMState {  
 ATMMachine atmMachine;  
  
 public NoCash(ATMMachine newATMMachine){  
 atmMachine = newATMMachine;  
 }  
  
 public void insertCard() {  
 System.*out*.println("Machine is out of money");  
 }  
  
 public void ejectCard() {  
 System.*out*.println("Machine is out of money");  
 }  
  
 public void requestCash(int cashToWithdraw) {  
 System.*out*.println("Machine is out of money");  
 }  
  
 public void insertPin(int pinEntered) {  
 System.*out*.println("Machine is out of money");  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 ATMMachine atmMachine = new ATMMachine();  
  
 atmMachine.insertCard();  
 atmMachine.ejectCard();  
 atmMachine.insertCard();  
 atmMachine.insertPin(1234);  
 atmMachine.requestCash(2000);  
 atmMachine.insertCard();  
 atmMachine.insertPin(1234);  
 }  
}

# Результаты работы программы:

****

**Вывод:** приобрели навыки применения паттернов проектирования при решении практических задач с использованием языка Java.