Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Объектно-ориентированное программирование

**ОТЧЁТ**

к лабораторной работе

на тему

Шаблоны поведения. Шаблон Chain of responsibility

Выполнил: студент группы 053506 Слуцкий Никита Сергеевич

Проверил: Летохо Александр Сергеевич

Минск 2022

Оглавление

[Цели выполнения задания 3](#_Toc65424824)

[Краткие теоретические сведения 3](#_Toc65424825)

[Задание 4](#_Toc65424826)

[Результат выполнения](#_Toc65424828) 4

[Вставки кода](#_Toc65424829) 6

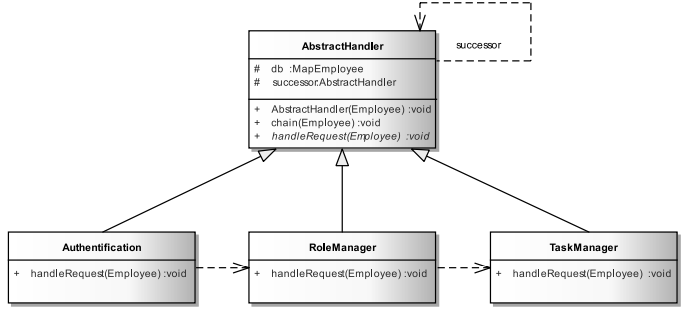
Цели выполнения задания:

Изучить поведенческий паттерн Chain Of Responsibility. Реализовать программный продукт с использованием данного шаблона согласно заданию.

Краткие теоретические сведения

**Шаблон Chain of Responsibility (Цепочка Обязанностей) организует независимую от объекта-отправителя цепочку не знающих возможностей друг друга объектов-получателей.**

**Он позволяет принять решение по обработке объекта на том уровне, на котором оно получено, или передаваться дальше по цепочке обработчиков. Применение шаблона связывает объекты составных частей приложения между собой по цепочке для передачи запроса на обработку от более низких****, детализированных слоёв системы к более высоким базовым. Или вдоль одного уровня иерархии, как на представленной схеме.**

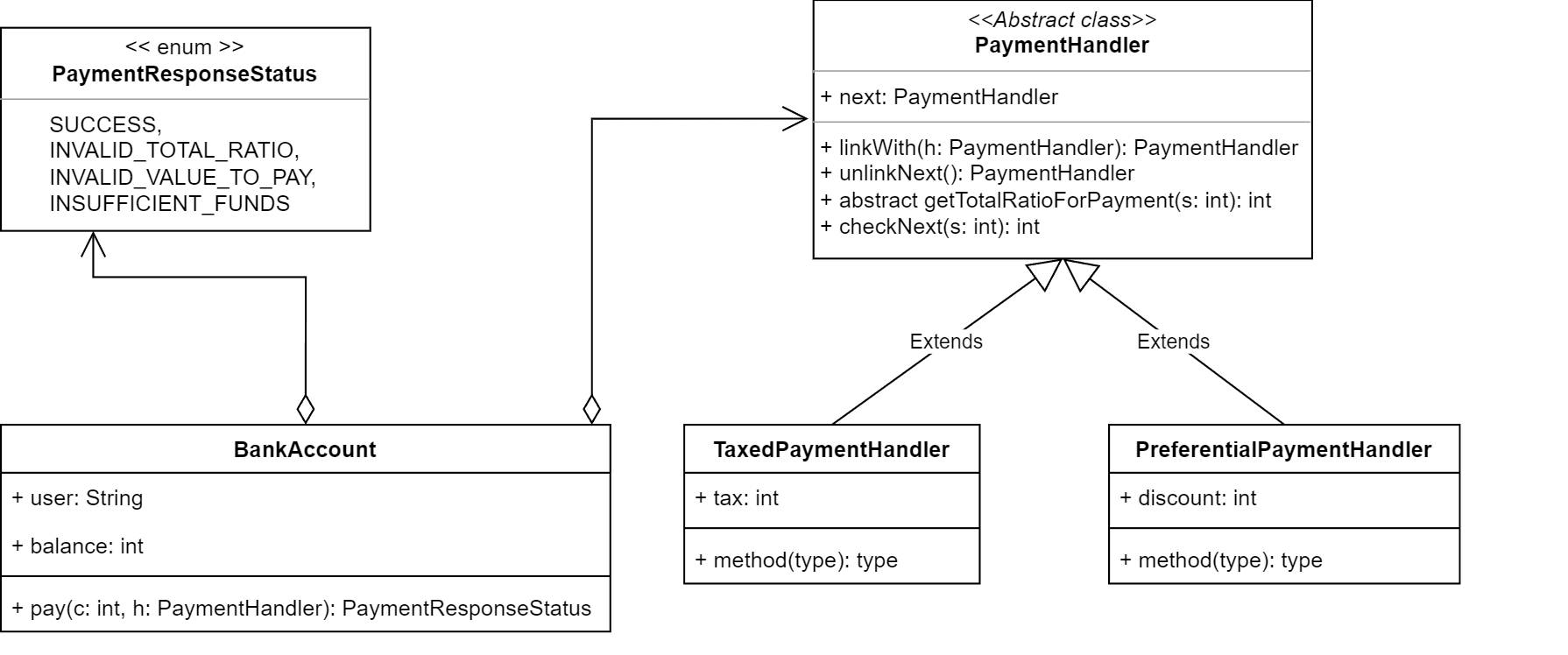


Задание

Паттерн Chain of Responsibility. Прохождение платежа через банковскую систему сопровождается целым рядом действий: фиксирующих, контролирующих, снимающих процент банка и прочие действия. Построить цепочки для различного вида платежей. Разработать модель системы.

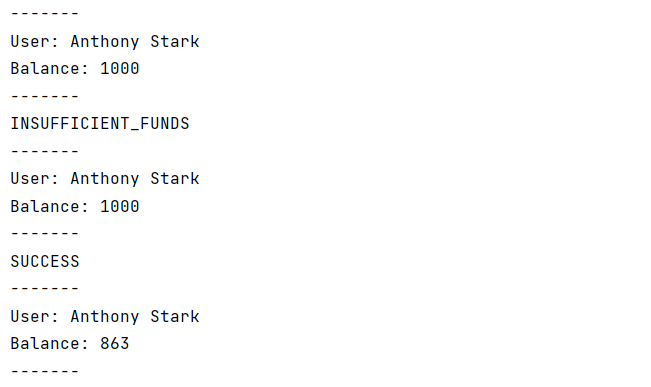
Необходимо описать иерархию классов предметной области в виде UML диаграммы и классов на языке Java, Kotlin или другом, предложенном преподавателем, в соответствии с выбранным вариантом задания. Предметная сущность – “Компьютеры”.

Ход выполнения:



Результат вывода программы

Разработано в среде IntelliJ IDEA на языке программирования Java.



Вставки кода классов из иерархии, а также кода класса для фабричного метода

**package** Bank.PaymentHandler;  
  
  
**public abstract class** PaymentHandler {  
 **protected** PaymentHandler **next**;  
  
 **public** PaymentHandler linkWith(PaymentHandler next) {  
 **this**.**next** = next;  
 **return** next;  
 }  
  
 **public** PaymentHandler unlinkNext() {  
 **var** temp = **this**.**next**;  
 **this**.**next** = **null**;  
 **return** temp;  
 }  
  
 **public abstract int** getTotalRatioForPayment(**int** startSum);  
  
 **protected int** checkNext(**int** startRatio) {  
 **return this**.**next** == **null** ? startRatio : **next**.getTotalRatioForPayment(startRatio);  
 }  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** Bank.PaymentHandler;  
  
**public class** PreferentialPaymentHandler **extends** PaymentHandler {  
 **private final int discount**;  
  
 **public** PreferentialPaymentHandler(**int** discount) {  
 **this**.**discount** = discount;  
 }  
  
 @Override  
 **public int** getTotalRatioForPayment(**int** startRatio) {  
 **return this**.checkNext(startRatio - **discount**);  
 }  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** Bank.PaymentHandler;  
  
  
**public class** TaxedPaymentHandler **extends** PaymentHandler {  
 **private final int tax**;  
  
 **public** TaxedPaymentHandler(**int** tax) {  
 **this**.**tax** = tax;  
 }  
  
 @Override  
 **public int** getTotalRatioForPayment(**int** startRatio) {  
 **return this**.checkNext(startRatio + **tax**);  
 }  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** Bank;  
  
**import** Bank.PaymentHandler.PaymentHandler;  
  
  
**public class** BankAccount {  
 **private final** String **user**;  
 **private int balance**;  
  
 **public** BankAccount(String name, **int** startBalance) {  
 **this**.**balance** = Math.*max*(startBalance, 0);  
 **this**.**user** = name;  
 }  
  
 **public** PaymentResponseStatus pay(**int** cost, PaymentHandler baseHandler) {  
 **if** (cost <= 0) **return** PaymentResponseStatus.***INVALID\_VALUE\_TO\_PAY***;  
  
 **double** totalRatio = (baseHandler.getTotalRatioForPayment(0) + 100) / 100.0;  
  
 **if** (totalRatio > 100 || totalRatio < 0) **return** PaymentResponseStatus.***INVALID\_TOTAL\_RATIO***;  
  
 **int** totalCost = (**int**) ((**double**) cost \* totalRatio);  
  
 **if** (totalCost <= **this**.**balance**) {  
 **this**.**balance** -= totalCost;  
 **return** PaymentResponseStatus.***SUCCESS***;  
 } **else** {  
 **return** PaymentResponseStatus.***INSUFFICIENT\_FUNDS***;  
 }  
 }  
  
 @Override  
 **public** String toString() {  
 **return "-------\nUser: "** + **this**.**user** + **"\nBalance: "** + **this**.**balance** + **"\n-------"**;  
 }  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**package** Bank;  
  
  
**public enum** PaymentResponseStatus {  
 ***SUCCESS***,  
 ***INVALID\_TOTAL\_RATIO***,  
 ***INVALID\_VALUE\_TO\_PAY***,  
 ***INSUFFICIENT\_FUNDS***}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**import** Bank.BankAccount;  
**import** Bank.PaymentHandler.PreferentialPaymentHandler;  
**import** Bank.PaymentHandler.TaxedPaymentHandler;  
**import** Bank.PaymentResponseStatus;  
  
  
**public class** TestPayment {  
 **public static void** main(String[] args) {  
 *//TestPayment.showExampleWithDiscounts();* TestPayment.*showExampleWithTaxes*();  
 }  
  
 **private static void** showExampleWithDiscounts() {  
 PaymentResponseStatus paymentStatus;  
 **var** studentDiscount = **new** PreferentialPaymentHandler(20);  
 **var** excellentStudentDiscount = **new** PreferentialPaymentHandler(10);  
 **var** competitionWinnerDiscount = **new** PreferentialPaymentHandler(5);  
  
 **var** loggedStudent = **new** BankAccount(**"John Smith"**, 1000);  
 System.***out***.println(loggedStudent);  
  
 paymentStatus = loggedStudent.pay(100, studentDiscount);  
 System.***out***.println(paymentStatus);  
 System.***out***.println(loggedStudent);  
  
 studentDiscount.linkWith(excellentStudentDiscount).linkWith(competitionWinnerDiscount);  
  
 paymentStatus = loggedStudent.pay(100, studentDiscount);  
 System.***out***.println(paymentStatus);  
 System.***out***.println(loggedStudent);  
 }  
  
 **private static void** showExampleWithTaxes() {  
 PaymentResponseStatus paymentStatus;  
 **var** carWeight = **new** TaxedPaymentHandler(30);  
 **var** enginesVolume = **new** TaxedPaymentHandler(7);  
  
 **var** carDealer = **new** BankAccount(**"Anthony Stark"**, 1000);  
 System.***out***.println(carDealer);  
  
 paymentStatus = carDealer.pay(1000, carWeight);  
 System.***out***.println(paymentStatus);  
 System.***out***.println(carDealer);  
  
 carWeight.linkWith(enginesVolume);  
  
 paymentStatus = carDealer.pay(100, carWeight);  
 System.***out***.println(paymentStatus);  
 System.***out***.println(carDealer);  
 }  
}