|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа,**

**обработки и интерпретации больших данных.**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 3 |

**Название:**

Классы, наследование, полиморфизм

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Е.Е. Шморгун |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

**Вариант 1 Задача 9.** Определить класс Квадратное уравнение. Класс должен содержать несколько конструкторов. Реализовать методы для поиска корней, экстремумов, а также интервалов убывания/возрастания. Создать массив объектов и определить наибольшие и наименьшие по значению корни.

Main.java

public class Main {

public static void main(String[] args) {

QuadraticEquation[] arr = {

new QuadraticEquation(1, 1, 0),

new QuadraticEquation(-2, 3),

new QuadraticEquation(0.4, 0, 5),

new QuadraticEquation(-0.4, 0.5, 5)

};

double min\_solution = arr[0].solve()[0];

double max\_solution = min\_solution;

for (int i = 0; i < arr.length; i++) {

System.out.printf("%d: ", i);

double[] solutions = arr[i].solve();

for (int j = 0; j < solutions.length; j++) {

System.out.printf("%s ", solutions[j]);

if (solutions[j] > max\_solution) {

max\_solution = solutions[j];

}

if (solutions[j] < min\_solution) {

min\_solution = solutions[j];

}

}

System.out.printf("\n");

}

System.out.printf("Max solution: %s\n", max\_solution);

System.out.printf("Min solution: %s\n", min\_solution);

Double[] increase\_interval = arr[0].increase\_interval();

Double[] decrease\_interval = arr[0].decrease\_interval();

System.out.printf("Extreme point for first: %s\n", arr[0].find\_extreme());

System.out.printf("Increase interval for first: (%s, %s)\n", increase\_interval[0], increase\_interval[1]);

System.out.printf("Decrease interval for first: (%s, %s)\n", decrease\_interval[0], decrease\_interval[1]);

}

}

QuadraticEquation.java

public class QuadraticEquation {

private double a;

private double b;

private double c;

public QuadraticEquation(double a, double b, double c) {

this.a = a;

this.b = b;

this.c = c;

}

public QuadraticEquation(double a) {

this.a = a;

this.b = 0;

this.c = 0;

}

public QuadraticEquation(double a, double b) {

this.a = a;

this.b = b;

this.c = 0;

}

public double[] solve() {

double discriminant = b \* b - 4 \* a \* c;

if (discriminant < 0) {

System.out.printf("Equation doesn't have real solutions");

return (new double[0]);

} else if (discriminant == 0) {

double[] solution = { -b / (2 \* a) };

return solution;

} else {

double[] solution = {

(-b + Math.sqrt(discriminant)) / (2 \* a),

(-b - Math.sqrt(discriminant)) / (2 \* a)

};

return solution;

}

}

public double find\_extreme() {

return -b / (2 \* a);

}

public Double[] decrease\_interval() {

Double[] result;

if (a > 0) {

result = new Double[]{Double.NEGATIVE\_INFINITY, find\_extreme()};

} else {

result = new Double[]{find\_extreme(), Double.POSITIVE\_INFINITY};

}

return result;

}

public Double[] increase\_interval() {

Double[] result;

if (a < 0) {

result = new Double[]{Double.NEGATIVE\_INFINITY, find\_extreme()};

} else {

result = new Double[]{find\_extreme(), Double.POSITIVE\_INFINITY};

}

return result;

}

}

**Вариант 1 Задача 2.** Определить класс Булева матрица (BoolMatrix) размерности (n x m). Класс должен содержать несколько конструкторов. Реализовать методы для логического сложения (дизъюнкции), умножения и инверсии матриц. Реализовать методы для подсчета числа единиц в матрице и упорядочения строк в лексикографическом порядке

Main.java

public class Main {

public static void main(String[] args) {

boolean[][] source\_matrix\_1 = {

{ true, true, false },

{ true, true, false },

{ false, true, false },

{ true, true, false }

};

boolean[][] source\_matrix\_2 = {

{ true, true, false },

{ false, false, true },

{ false, true, false },

{ true, true, true }

};

BoolMatrix matrix\_1 = new BoolMatrix(source\_matrix\_1);

BoolMatrix matrix\_2 = new BoolMatrix(source\_matrix\_2);

BoolMatrix matrix\_3 = new BoolMatrix(4, 3);

BoolMatrix matrix\_4 = matrix\_1

.logic\_multiply(matrix\_2.invert())

.logic\_add(matrix\_3);

System.out.println("Operations result:");

matrix\_4.print();

System.out.printf("Count: %d\n", matrix\_4.count());

System.out.println("Sort second matrix:");

matrix\_2.sort();

matrix\_2.print();

}

}

BoolMatrix.java

public class Main {

public static void main(String[] args) {

boolean[][] source\_matrix\_1 = {

{ true, true, false },

{ true, true, false },

{ false, true, false },

{ true, true, false }

};

boolean[][] source\_matrix\_2 = {

{ true, true, false },

{ false, false, true },

{ false, true, false },

{ true, true, true }

};

BoolMatrix matrix\_1 = new BoolMatrix(source\_matrix\_1);

BoolMatrix matrix\_2 = new BoolMatrix(source\_matrix\_2);

BoolMatrix matrix\_3 = new BoolMatrix(4, 3);

BoolMatrix matrix\_4 = matrix\_1

.logic\_multiply(matrix\_2.invert())

.logic\_add(matrix\_3);

System.out.println("Operations result:");

matrix\_4.print();

System.out.printf("Count: %d\n", matrix\_4.count());

System.out.println("Sort second matrix:");

matrix\_2.sort();

matrix\_2.print();

}

}

**Вариант 2 Задача 9.** Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы setТип(), getТип(), toString(). Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль.

Product: id, Наименование, UPC, Производитель, Цена, Срок хранения, Количество. Создать массив объектов. Вывести: a) список товаров для заданного наименования; b) список товаров для заданного наименования, цена которых не превосходит заданную; c) список товаров, срок хранения которых больше заданного.

Main.java

import java.util.Date;

import java.util.List;

public class Main {

public static void main(String[] args) {

List<Product> products = Product.createSample();

System.out.printf("Generated Sample:\n%s\n", products);

System.out.printf("Apple products:\n%s\n", Product.getByName(products, "apple"));

System.out.printf("Cheap apple products:\n%s\n", Product.getByNameAndPrice(products, "apple", 1));

System.out.printf("Long living products:\n%s\n", Product.getByExpirationDate(products, new Date(2023, 4, 1)));

}

}

Product.java

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Date;

import java.util.List;

import java.util.stream.Collectors;

public class Product {

private int id;

private String name;

private String upc;

private double price;

private String manufacturer;

private Date expirationDate;

public Product(int id, String name, String upc, double price, String manufacturer, Date expirationDate) {

this.id = id;

this.name = name;

this.upc = upc;

this.price = price;

this.manufacturer = manufacturer;

this.expirationDate = expirationDate;

}

public int getId() { return this.id; }

public void setId(int id) { this.id = id; }

public String getName() { return this.name; }

public void setName(String name) { this.name = name; }

public String getUpc() { return this.upc; }

public void setUpc(String upc) { this.upc = upc; }

public double getPrice() { return this.price; }

public void setPrice(double price) { this.price = price; }

public String getManufacturer() { return this.manufacturer; }

public void setManufacturer(String manufacturer) { this.manufacturer = manufacturer; }

public Date getExpirationDate() { return this.expirationDate; }

public void setExpirationDate(Date expirationDate) { this.expirationDate = expirationDate; }

public String toString() {

return String.format("\nID: %s\n" +

"Name: %s\n" +

"UPC: %s\n" +

"Price: %s\n" +

"Manufacturer: %s\n" +

"Expiration Date: %s\n",

id, name, upc, price, manufacturer, expirationDate

);

}

public static List<Product> createSample() {

Product[] products = {

new Product(1, "apple", "111", 1, "garden", new Date(2023, 3, 20)),

new Product(2, "apple", "112", 1.25, "garden", new Date(2023, 3, 22)),

new Product(3, "tomato", "113", 2, "garden+", new Date(2023, 3, 23)),

new Product(4, "potato", "114", 1.5, "garden-", new Date(2023, 4, 15)),

new Product(5, "potato", "115", 1.5, "garden+", new Date(2023, 4, 11)),

new Product(6, "pepper", "116", 3, "garden+", new Date(2023, 5, 25)),

};

return Arrays.asList(products);

}

public static List<Product> getByName(List<Product> products, String name) {

return products

.stream()

.filter(product -> product.getName() == name)

.collect(Collectors.toList());

}

public static List<Product> getByNameAndPrice(List<Product> products, String name, double price) {

return products

.stream()

.filter(product -> product.getName() == name && product.getPrice() <= price)

.collect(Collectors.toList());

}

public static List<Product> getByExpirationDate(List<Product> products, Date expirationDate) {

return products

.stream()

.filter(product -> product.getExpirationDate().compareTo(expirationDate) >= 0)

.collect(Collectors.toList());

}

}

**Вариант 2 Задача 10.** Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы setТип(), getТип(), toString(). Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль.

Train: Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс). Создать массив объектов. Вывести: a) список поездов, следующих до заданного пункта назначения; b) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа; c) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места

Main.java

import java.util.Date;

import java.util.List;

public class Main {

public static void main(String[] args) {

List<Train> trains = Train.createSample();

System.out.printf("Generated sample:\n%s\n", trains);

System.out.printf("Trains to Moscow:\n%s\n", Train.getByDestination(trains, "Moscow"));

System.out.printf(

"Trains to Moscow after 18:00:\n%s\n",

Train.getByDestinationAndTime(trains, "Moscow", new Date(2023, 3, 20, 18, 0 ,0))

);

System.out.printf("Trains to Moscow:\n%s\n", Train.getByDestinationAndAvailableCommonSeats(trains, "Moscow"));

}

}

Train.java

import java.util.Arrays;

import java.util.Date;

import java.util.List;

import java.util.stream.Collectors;

public class Train {

private String destination;

private String number;

private Date departureTime;

private int commonSeats;

private int coupeSeats;

private int reservedSeats;

private int luxurySeats;

public Train(

String destination,

String number,

Date departureTime,

int commonSeats,

int coupeSeats,

int reservedSeats,

int luxurySeats

) {

this.destination = destination;

this.number = number;

this.departureTime = departureTime;

this.commonSeats = commonSeats;

this.coupeSeats = coupeSeats;

this.reservedSeats = reservedSeats;

this.luxurySeats = luxurySeats;

}

public String getDestination() { return destination; }

public void setDestination(String destination) { this.destination = destination; }

public String getNumber() { return number; }

public void setNumber(String number) { this.number = number; }

public Date getDepartureTime() { return departureTime; }

public void setDepartureTime(Date departureTime) { this.departureTime = departureTime; }

public int getCommonSeats() { return commonSeats; }

public void setCommonSeats(int commonSeats) { this.commonSeats = commonSeats; }

public int getCoupeSeats() { return coupeSeats; }

public void setCoupeSeats(int coupeSeats) { this.coupeSeats = coupeSeats; }

public int getReservedSeats() { return reservedSeats; }

public void setReservedSeats(int reservedSeats) { this.reservedSeats = reservedSeats; }

public int getLuxurySeats() { return luxurySeats; }

public void setLuxurySeats(int luxurySeats) { this.luxurySeats = luxurySeats; }

public String toString() {

return String.format(

"\nDestination: %s\n" +

"Number: %s\n" +

"Departure Time: %s\n" +

"Common Seats: %s\n" +

"Coupe Seats: %s\n" +

"Reserved Seats: %s\n" +

"Luxury Seats: %s\n",

destination, number, departureTime, commonSeats, coupeSeats, reservedSeats, luxurySeats

);

}

public static List<Train> createSample() {

Date[] dates = {

new Date(2023, 3, 20, 17, 24, 0),

new Date(2023, 3, 20, 19, 24, 0),

new Date(2023, 3, 18, 13, 15, 0),

new Date(2023, 3, 17, 17, 24, 0),

new Date(2023, 3, 16, 17, 10, 0),

};

Train[] trains = {

new Train("Moscow", "m100", dates[0], 10, 10, 10, 10),

new Train("Moscow", "m101", dates[1], 0, 0, 10, 5),

new Train("St. Petersburg", "s100", dates[2], 0, 1, 2, 3),

new Train("Kazan", "k100", dates[3], 0, 10, 5, 3),

new Train("Perm", "p100", dates[4], 5, 10, 5, 3),

};

return Arrays.asList(trains);

}

public static List<Train> getByDestination(List<Train> trains, String destination) {

return trains

.stream()

.filter(train -> train.getDestination() == destination)

.collect(Collectors.toList());

}

public static List<Train> getByDestinationAndTime(List<Train> trains, String destination, Date departureTime) {

return trains

.stream()

.filter(train -> train.getDestination() == destination &&

train.getDepartureTime().compareTo(departureTime) > 0

)

.collect(Collectors.toList());

}

public static List<Train> getByDestinationAndAvailableCommonSeats(List<Train> trains, String destination) {

return trains

.stream()

.filter(train -> train.getDestination() == destination && train.getCommonSeats() > 0)

.collect(Collectors.toList());

}

}

**Вариант 3 Задача 9.** Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы equals(), hashCode(), toString().

Создать объект класса Год, используя классы Месяц, День. Методы: задать дату, вывести на консоль день недели по заданной дате, рассчитать количество дней, месяцев в заданном временном промежутке.

Main.java

public class Main {

public static void main(String[] args) {

Year begin = new Year(2023, 1, 12);

Year end = new Year(2023, 2, 24);

System.out.println(begin);

int[] periods = Year.calcPeriod(begin, end);

System.out.printf("Day of week: %s\n", begin.dayOfWeek());

System.out.printf("Passed months and days: %s, %s\n", periods[0], periods[1]);

}

}

Year.java

import java.util.Date;

public class Year {

private int number;

private Month month;

private Day day;

public Year(int year, int month, int day) {

this.setDate(year, month, day);

}

public Month getMonth() { return month; }

public Day getDay() { return day; }

public void setDate(int year, int month, int day) {

this.number = year;

this.month = Year.MONTHS[month - 1];

this.day = new Day(day);

}

public String dayOfWeek() {

return Day.NAMES[toDate().getDay()];

}

public Date toDate() {

return new Date(number, month.getNumber(), day.getNumber());

}

public static final Month[] MONTHS = {

new Month("January", 1,31),

new Month("February", 2,28),

new Month("March", 3,31),

new Month("April", 4, 30),

new Month("May", 5,31),

new Month("June", 6,30),

new Month("July", 7,31),

new Month("August", 8,31),

new Month("September", 9, 30),

new Month("October", 10, 31),

new Month("November", 11,30),

new Month("December", 12, 31)

};

public static int[] calcPeriod(Year begin, Year end) {

int passedDays = 0;

int passedMonthes = 0;

Month begin\_month = begin.getMonth();

Month end\_month = end.getMonth();

if (begin\_month.getNumber() == end\_month.getNumber()) {

passedDays = end.getDay().getNumber() - begin.getDay().getNumber();

} else {

passedDays += begin.getMonth().getTotalDays() - begin.getDay().getNumber();

passedDays += end.getDay().getNumber();

passedMonthes += 1;

for (int i = begin.getMonth().getNumber() + 1; i < end.getMonth().getNumber(); i++) {

passedDays += Year.MONTHS[i - 1].getTotalDays();

passedMonthes += 1;

}

}

int[] result = { passedMonthes, passedDays };

return result;

}

}

Month.java

public class Month {

private int totalDays;

private String name;

private int number;

public int getNumber() { return number; }

public int getTotalDays() { return totalDays; }

public boolean equals(Month month) {

return this.number == month.number &&

this.name.equals(month.name) &&

this.totalDays == month.totalDays;

}

public String toString() {

return String.format("\nName: %s\n" +

"Number of month: %s\n" +

"Total days: %s\n",

name, number, totalDays

);

}

@Override

public int hashCode() {

return number \* totalDays \* name.hashCode();

}

public Month(String name, int number, int totalDays) {

this.name = name;

this.number = number;

this.totalDays = totalDays;

}

}

Day.java

public class Day {

private int number;

public Day(int number) {

this.number = number;

}

public int getNumber() { return number; }

public int hashCode() {

return number;

}

public String toString() {

return String.format("Day: %s", number);

}

public boolean equals(Day day) {

return this.number == day.number;

}

public static String[] NAMES = {

"Monday",

"Tuesday",

"Wednesday",

"Thursday",

"Friday",

"Saturday",

"Sunday"

};

}

**Вариант 3 Задача 10.** Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы equals(), hashCode(), toString().

Создать объект класса Сутки, используя классы Час, Минута. Методы: вывести на консоль текущее время, рассчитать время суток (утро, день, вечер, ночь).

Main.java

public class Main {

public static void main(String[] args) {

Day day = new Day(17, 30);

System.out.printf("Day:\n%s\n", day);

System.out.printf("Part of day: %s\n", day.partOfDay());

}

}

Day.java

public class Day {

private Hour hour;

private Minute minute;

public Day(int hour, int minute) {

this.hour = new Hour(hour);

this.minute = new Minute(minute);

}

public boolean equals(Day day) {

return hour.equals(day.hour) && minute.equals(day.minute);

}

@Override

public int hashCode() {

return hour.hashCode() \* minute.hashCode();

}

public String toString() {

return hour.toString() + "\n" + minute.toString();

}

public String partOfDay() {

if (hour.getNumber() >= 0 && hour.getNumber() < 6) {

return "Night";

} else if (hour.getNumber() >= 6 && hour.getNumber() < 12) {

return "Morning";

} else if (hour.getNumber() >= 12 && hour.getNumber() < 18) {

return "Day";

} else {

return "Evening";

}

}

}

Hour.java

public class Hour {

private int number;

public Hour(int hour) { this.number = hour; }

public int getNumber() { return number; }

public boolean equals(Hour hour) {

return number == hour.number;

}

@Override

public int hashCode() {

return number;

}

public String toString() {

return String.format("Hour: %s", number);

}

}

Minute.java

public class Minute {

private int number;

public int getNumber() { return number; }

public Minute(int minute) { this.number = minute; }

public boolean equals(Minute minute) {

return number == minute.number;

}

@Override

public int hashCode() {

return number;

}

public String toString() {

return String.format("Minute: %s", number);

}

}

**Вариант 4. Задача 9.** Система Интернет-магазин. Администратор добавляет информацию о Товаре. Клиент делает и оплачивает Заказ на Товары. Администратор регистрирует Продажу и может занести неплательщиков в «черный список».

Main.java

public class Main {

public static void main(String[] args) {

App app = new App();

app.run();

}

}

App.java

import java.io.IOException;

import java.util.Scanner;

public class App {

public App() {

this.in = new Scanner(System.in);

}

@Override

protected void finalize() {

try {

in.close();

} catch (Exception e){

return;

}

}

public void run() {

App.initScanner();

Actor actor = new Actor(Actor.Role.CLIENT);

Shop shop = new Shop();

boolean actor\_chosen = false;

while(true) {

App.commandPrompt();

if (!actor\_chosen) {

print\_chose\_actor\_prompt();

int command = App.getCommand();

if (command == 1) {

actor.setRole(Actor.Role.CLIENT);

actor\_chosen = true;

} else if (command == 2) {

actor.setRole(Actor.Role.ADMIN);

actor\_chosen = true;

} else if (command == 3) {

break;

} else {

App.wrongCommandPrompt();

}

} else {

actor.printCommandPrompt();

int command = App.getCommand();

boolean exited = actor.executeCommand(command, shop);

if (exited) { actor\_chosen = false; }

}

}

App.closeScanner();

}

public static void print\_chose\_actor\_prompt() {

System.out.print("Who are you?: \n" +

"1: Client \n" +

"2. Admin \n" +

"3. Exit\n" +

"Your choice: "

);

}

static public int getCommand() {

int command = in.nextInt();

in.nextLine();

return command;

}

static public String getStringValue() {

return in.nextLine();

}

static public double getNumericValue() {

return in.nextDouble();

}

static public void commandPrompt() {

System.out.println("Enter command:");

}

static public void wrongCommandPrompt() {

System.out.println("Wrong command");

}

private static Scanner in;

public static void initScanner() { in = new Scanner(System.in); }

public static void closeScanner() { in.close(); }

}

Actor.java

public class Actor {

private Role role;

private boolean identified;

private int clientId;

public Actor(Role role) {

this.role = role;

this.identified = false;

};

public void setRole(Role role) { this.role = role; }

public void printCommandPrompt() {

if (role == Role.ADMIN) {

printAdminCommandPrompt();

} else {

printClientCommandPrompt();

}

}

public boolean executeCommand(int command, Shop shop) {

if (role == Role.ADMIN) {

return executeAdminCommand(command, shop);

} else {

return executeClientCommand(command, shop);

}

}

private boolean executeAdminCommand(int command, Shop shop) {

if (command == 1) {

shop.commandCreateNewItem();

} else if (command == 2) {

shop.commandCompletePayment();

} else if (command == 3) {

shop.commandAddClientToBlacklist();

} else if (command == 4) {

return true;

} else {

App.wrongCommandPrompt();

}

return false;

}

private boolean executeClientCommand(int command, Shop shop) {

if (identified) {

if (command == 1) {

shop.commandCreateOrder(clientId);

} else if (command == 2) {

shop.commandPayOrder(clientId);

} else if (command == 3) {

this.identified = false;

return true;

}

} else {

executeIdentificationCommand(command, shop);

}

return false;

}

private void printAdminCommandPrompt() {

System.out.print("1. Add item\n" +

"2. Register payment\n" +

"3. Add client to black list\n" +

"4. Chose another role\n" +

"Your choice: "

);

}

private void printClientCommandPrompt() {

if (identified) {

System.out.print("1. Create order\n" +

"2. Pay order\n" +

"3. Chose another role\n" +

"Your choice: ");

} else {

System.out.print("1. Chose client\n" +

"2. Create client\n" +

"Your choice: ");

}

}

private void executeIdentificationCommand(int command, Shop shop) {

if (command == 1) {

Client client = shop.commandChoseClient();

this.clientId = client.getId();

this.identified = true;

} else if (command == 2) {

Client client = shop.commandCreateClient();

this.clientId = client.getId();

this.identified = true;

} else {

App.wrongCommandPrompt();

}

}

static public enum Role {

CLIENT,

ADMIN

}

}

Order.java

import java.util.List;

import java.util.stream.Collectors;

import java.util.stream.IntStream;

public class Order {

private int id;

private List<Integer> itemIds;

private int clientId;

public Order(int id, int clientId, List<Integer> itemIds) {

this.id = id;

this.clientId = clientId;

this.itemIds = itemIds;

}

public int getId() { return id; }

public String toString() {

return String.format("Order ID: %s, Client ID: %s, items: %s", id, clientId, itemIds);

}

static void printList(List<Order> orders) {

IntStream.range(0, orders.size()).forEach(

i -> System.out.println((i + 1) + ". " + orders.get(i).toString())

);

}

static List<Order> unpaidClientOrders(int clientId, List<Order> orders, List<Payment> payments) {

List<Integer> orderIds = payments

.stream()

.filter(payment -> payment.getClientId() == clientId && !payment.getPaid())

.map(payment -> payment.getOrderId())

.collect(Collectors.toList());

return orders

.stream()

.filter(order -> orderIds.contains(order.getId()))

.collect(Collectors.toList());

}

}

Item.java

import java.util.List;

import java.util.stream.IntStream;

public class Item {

private int id;

private String name;

private double price;

public Item(int id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

static void printList(List<Item> items) {

IntStream.range(0, items.size()).forEach(

i -> System.out.println((i + 1) + ". " + items.get(i).toString())

);

}

public String toString() {

return String.format("ID: %s, Name: %s, Price: %s", id, name, price);

}

}

**Вариант 4. Задача 10.** Система Железнодорожная касса. Пассажир делает Заявку на станцию назначения, время и дату поездки. Система регистрирует Заявку и осуществляет поиск подходящего Поезда. Пассажир делает выбор Поезда и получает Счет на оплату. Администратор вводит номера Поездов, промежуточные и конечные станции, цены

Main.java

public class Main {

public static void main(String[] args) {

App app = new App();

app.run();

}

}

App.java

import java.io.IOException;

import java.util.Scanner;

public class App {

public App() {

this.in = new Scanner(System.in);

}

@Override

protected void finalize() {

try {

in.close();

} catch (Exception e){

return;

}

}

public void run() {

App.initScanner();

Actor actor = new Actor(Actor.Role.CLIENT);

Railway railway = new Railway();

boolean actor\_chosen = false;

while(true) {

App.commandPrompt();

if (!actor\_chosen) {

print\_chose\_actor\_prompt();

int command = App.getCommand();

if (command == 1) {

actor.setRole(Actor.Role.CLIENT);

actor\_chosen = true;

} else if (command == 2) {

actor.setRole(Actor.Role.ADMIN);

actor\_chosen = true;

} else if (command == 3) {

break;

} else {

App.wrongCommandPrompt();

}

} else {

actor.printCommandPrompt();

int command = App.getCommand();

boolean exited = actor.executeCommand(command, railway);

if (exited) { actor\_chosen = false; }

}

}

App.closeScanner();

}

public static void print\_chose\_actor\_prompt() {

System.out.print("Who are you?: \n" +

"1: Client \n" +

"2. Admin \n" +

"3. Exit\n" +

"Your choice: "

);

}

static public int getCommand() {

int command = in.nextInt();

in.nextLine();

return command;

}

static public String getStringValue() {

return in.nextLine();

}

static public double getNumericValue() {

return in.nextDouble();

}

static public void commandPrompt() {

System.out.println("Enter command:");

}

static public void wrongCommandPrompt() {

System.out.println("Wrong command");

}

private static Scanner in;

public static void initScanner() { in = new Scanner(System.in); }

public static void closeScanner() { in.close(); }

}

Actor.java

public class Actor {

private Role role;

private boolean identified;

private int clientId;

public Actor(Role role) {

this.role = role;

this.identified = false;

};

public void setRole(Role role) { this.role = role; }

public void printCommandPrompt() {

if (role == Role.ADMIN) {

printAdminCommandPrompt();

} else {

printClientCommandPrompt();

}

}

public boolean executeCommand(int command, Railway railway) {

if (role == Role.ADMIN) {

return executeAdminCommand(command, railway);

} else {

return executeClientCommand(command, railway);

}

}

private boolean executeAdminCommand(int command, Railway railway) {

if (command == 1) {

railway.commandCreateNewStation();

} else if (command == 2) {

railway.commandCreateNewTrain();

} else if (command == 3) {

return true;

} else {

App.wrongCommandPrompt();

}

return false;

}

private boolean executeClientCommand(int command, Railway railway) {

if (identified) {

if (command == 1) {

railway.commandCreateApplication(clientId);

} else if (command == 2) {

railway.commandPayBill(clientId);

} else if (command == 3) {

this.identified = false;

return true;

}

} else {

executeIdentificationCommand(command, railway);

}

return false;

}

private void printAdminCommandPrompt() {

System.out.print("1. Add station\n" +

"2. Add train\n" +

"3. Chose another role\n" +

"Your choice: "

);

}

private void printClientCommandPrompt() {

if (identified) {

System.out.print("1. Create Application\n" +

"2. Pay Bill\n" +

"3. Chose another role\n" +

"Your choice: ");

} else {

System.out.print("1. Chose client\n" +

"2. Create client\n" +

"Your choice: ");

}

}

private void executeIdentificationCommand(int command, Railway railway) {

if (command == 1) {

Client client = railway.commandChoseClient();

this.clientId = client.getId();

this.identified = true;

} else if (command == 2) {

Client client = railway.commandCreateClient();

this.clientId = client.getId();

this.identified = true;

} else {

App.wrongCommandPrompt();

}

}

static public enum Role {

CLIENT,

ADMIN

}

}

Train.java

import java.util.Date;

import java.util.List;

import java.util.stream.IntStream;

public class Train {

public int id;

public String number;

public List<Integer> stationIds;

public double price;

public Date departureDate;

public Train(int id, String number, List<Integer> stationIds, double price, Date departureDate) {

this.id = id;

this.number = number;

this.stationIds = stationIds;

this.price = price;

this.departureDate = departureDate;

}

public List<Integer> getStationIds() { return this.stationIds; }

public Date getDepartureDate() { return this.departureDate; }

public double getPrice() { return price; }

static void printList(List<Train> trains) {

IntStream.range(0, trains.size()).forEach(

i -> System.out.println((i + 1) + ". " + trains.get(i).toString())

);

}

public String toString() {

return String.format(

"ID: %s, Number: %s, price: %s, Departure Date: %s, Stations: %s",

id, number, price, departureDate, stationIds

);

}

}

Railway.java

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.LinkedList;

import java.util.List;

import java.util.stream.Collectors;

public class Railway {

private List<Station> stations;

private List<Client> clients;

private List<Train> trains;

private List<Application> applications;

private List<Bill> bills;

public Railway() {

this.stations = new LinkedList<>();

this.clients = new LinkedList<>();

this.trains = new LinkedList<>();

this.applications = new LinkedList<>();

this.bills = new LinkedList<>();

}

public void commandCreateNewStation() {

System.out.println("Enter station name");

String name = App.getStringValue();

System.out.println("Enter 1 if station is last and 2 if its intermediate");

double choice = App.getNumericValue();

int stationId = stations.size() + 1;

Station station = new Station(stationId, name, choice == 1);

stations.add(station);

System.out.println("Current stations:");

Station.printList(stations);

}

public void commandCreateNewTrain() {

System.out.println("Enter train number");

String number = App.getStringValue();

String dateFormat = "dd/MM/yyyy HH:mm";

SimpleDateFormat parser = new SimpleDateFormat(dateFormat);

System.out.println("Enter departure date in format:" + dateFormat);

Date departureDate = null;

try {

departureDate = parser.parse(App.getStringValue());

} catch (ParseException e) {

throw new RuntimeException(e);

}

System.out.println("Enter price");

double price = App.getNumericValue();

Station chosenStation;

List<Integer> chosenStationIds = new LinkedList<Integer>();

do {

List<Station> leftStations = stations

.stream()

.filter(station -> !(chosenStationIds.contains(station.getId())))

.collect(Collectors.toList());

System.out.println("Chose station till last:");

Station.printList(leftStations);

int stationChoice = App.getCommand();

chosenStation = stations.get(stationChoice - 1);

chosenStationIds.add(chosenStation.getId());

} while (!chosenStation.isLast());

int trainId = trains.size() + 1;

Train train = new Train(trainId, number, chosenStationIds, price, departureDate);

trains.add(train);

System.out.println("Current trains");

Train.printList(trains);

}

public Client commandChoseClient() {

System.out.println("Chose client: ");

Client.printList(clients);

int clientChoice = App.getCommand();

return clients.get(clientChoice - 1);

}

public Client commandCreateClient() {

System.out.println("Enter client name: ");

String clientName = App.getStringValue();

int clientId = clients.size() + 1;

Client client = new Client(clientId, clientName);

clients.add(client);

return client;

}

public void commandCreateApplication(int clientId) {

System.out.println("Chose station");

Station.printList(stations);

int stationChoice = App.getCommand();

Station station = stations.get(stationChoice - 1);

String dateFormat = "dd/MM/yyyy HH:mm";

SimpleDateFormat parser = new SimpleDateFormat(dateFormat);

System.out.println("Enter departure date in format: " + dateFormat);

Date departureDate = null;

try {

departureDate = parser.parse(App.getStringValue());

} catch (ParseException e) {

throw new RuntimeException(e);

}

int applicationId = applications.size() + 1;

Application application = new Application(applicationId, clientId, station, departureDate);

applications.add(application);

List<Train> possibleTrains = application.possibleTrains(trains);

if (possibleTrains.size() == 0) {

System.out.println("There are no possible trains");

return;

}

System.out.println("Chose from possible trains");

Train.printList(possibleTrains);

int trainChoice = App.getCommand();

Train train = possibleTrains.get(trainChoice - 1);

application.setTrain(train);

Bill bill = new Bill(bills.size() + 1, application);

bills.add(bill);

System.out.println("Your current application:");

Application.printList(Application.selectByClientId(clientId, applications));

}

public void commandPayBill(int clientId) {

List<Bill> unpaidBills = Bill.selectClientUnpaidBills(clientId, bills);

if (unpaidBills.size() == 0) {

System.out.println("There are no unpaid bills");

return;

}

System.out.println("Your unpaid bills:");

Bill.printList(unpaidBills);

int billChoice = App.getCommand();

unpaidBills.get(billChoice - 1).pay();

}

}

**Вывод:** В данной работе я изучил работу с классами, наследованием и полиморфизмом. Создал несколько приложений, которые используют данные понятия.