Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №5

По дисциплине «Современные платформы программирования»

Выполнила:

Студентка 3 курса

Группы ПО-3

Пивчик В.Г.

Проверил:

Крощенко А.А.

Брест 2020 г.

**Цель работы:**

Приобрести практические навыки в области объектно-ориентированного проектирования.

**Вариант 9**

Постановка задачи:

Задание 1:

Вариант 9

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

interface Корабль ← class Грузовой Корабль ← class Танкер

Задание 2:

В следующих заданиях требуется создать суперкласс (абстрактный класс, интерфейс) и определить общие методы для данного класса. Создать подклассы, в которых добавить специфические свойства и методы. Часть методов переопределить. Создать массив объектов суперкласса и заполнить объектами подклассов. Объекты подклассов идентифицировать конструктором по имени или идентификационному номеру. Использовать объекты подклассов для моделирования реальных ситуаций и объектов.

Создать суперкласс Транспортное средство и подклассы Автомобиль, Велосипед, Повозка. Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.

Задание 3:

В задании 3 ЛР No4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

**Ход работы**

**Текст программы:**

**Задание 1**

**Код программы**

**Main**

package com.company;

public class Main {

public static void main(String[] args) {

// by interface

Ship cargoShipOne = new CargoShip();

System.out.println("Cargo ship one: ");

cargoShipOne.swim();

cargoShipOne.load();

cargoShipOne.weight();

Ship tankerOne = new Tanker();

System.out.println("Tanker one: ");

tankerOne.swim();

tankerOne.load();

tankerOne.weight();

System.out.println("");

// by class

CargoShip cargoShipTwo = new CargoShip();

System.out.println("Cargo ship two: ");

cargoShipTwo.swim();

cargoShipTwo.load();

cargoShipTwo.weight();

cargoShipTwo.passengers();

Tanker tankerTwo = new Tanker();

System.out.println("Tanker two: ");

tankerTwo.swim();

tankerTwo.load();

tankerTwo.typeOfGoods();

tankerTwo.weight();

tankerTwo.passengers();

}

}

**Ship**

package com.company;

public interface Ship {

default void swim() {

System.out.println("I can swim in the sea!");

}

void weight();

void load();

}

**CargoShip**

package com.company;

public class CargoShip implements Ship {

@Override

public void load() {

System.out.println("I'm carrying goods");

}

@Override

public void weight() {

System.out.println("My weight is 150 tonnes");

}

public void passengers(){

System.out.println("I have no passengers!");

}

}

**Tanker**

package com.company;

public class Tanker extends CargoShip {

public void typeOfGoods() { System.out.println("My load is oil");

}

@Override

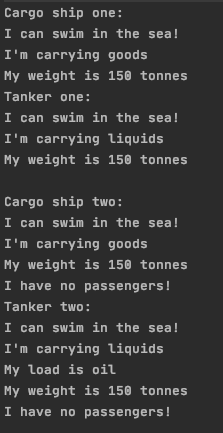
public void load() {

System.out.println("I'm carrying liquids");

}

}

**Рисунок с результатом работы программы**

****

**Задание 2**

**Код программы**

**Main**

package com.company;

import java.util.ArrayList;

public class Main {

public static final int DISTANCE = 70;

public static void main(String[] args) {

ArrayList<Vehicle> vehicles = new ArrayList<>();

Car car = new Car("Skoda", 60);

vehicles.add(car);

car.sayBeep();

Bicycle bicycle = new Bicycle("Mustang", 15);

vehicles.add(bicycle);

bicycle.sayDing();

Wagon wagon = new Wagon(12, 25);

vehicles.add(wagon);

wagon.sayCap();

for (int i = 0; i < vehicles.size(); i++) {

System.out.println("");

System.out.println("Vehicle: " + vehicles.get(i).toString());

System.out.println("Distance: " + DISTANCE + " kms");

System.out.println("Time: " + vehicles.get(i).rideTime(DISTANCE) + " hours");

System.out.println("Cost for person: " + vehicles.get(i).rideCost(DISTANCE) + " dollars");

System.out.println("Cost for goods: " + vehicles.get(i).loadCost(DISTANCE) + " dollars");

vehicles.get(i).ride();

}

}

}

**Vehicle**

package com.company;

public interface Vehicle {

default void ride() { System.out.println("Let's go!");

}

double rideTime(double distance);

double rideCost(double distance);

double loadCost(double distance);

}

**Car**

package com.company;

public class Car implements Vehicle {

private String name;

private double speed;

private double cargoCoeff;

private double personCoeff;

private double seatsAmount;

public Car(String name, double speed) { this.name = name;

this.speed = speed;

this.cargoCoeff = 2.5; this.personCoeff = 1.75; this.seatsAmount = 4;

}

public void sayBeep() { System.out.println("Beep-beep!");

}

@Override

public double rideTime(double distance) {

return distance / this.speed;

}

@Override

public double rideCost(double distance) {

return distance \* personCoeff;

}

@Override

public double loadCost(double distance) {

return distance \* cargoCoeff;

}

@Override

public String toString() {

return "Car{" + "name='" + name + '\'' + '}';

}

}

**Wagon**

package com.company;

public class Wagon implements Vehicle {

private int number;

private double speed;

private double cargoCoeff;

private double personCoeff;

private double horseAmount;

public Wagon(int number, double speed) {

this.number = number;

this.speed = speed;

this.cargoCoeff = 1.25;

this.personCoeff = 0.75;

this.horseAmount = 4;

}

public void sayCap() {

System.out.println("Cap-cap!");

}

@Override

public double rideTime(double distance) {

return distance / this.speed;

}

@Override

public double rideCost(double distance) {

return distance \* personCoeff;

}

@Override

public double loadCost(double distance) {

return distance \* cargoCoeff;

}

@Override

public String toString() {

return "Wagon{" + "number=" + number + '}';

}

}

**Bicycle**

package com.company;

public class Bicycle implements Vehicle {

private String name;

private double speed;

private double cargoCoeff;

private double personCoeff;

public Bicycle(String name,double speed){

this.name = name;

this.speed = speed;

this.cargoCoeff = 0.75;

this.personCoeff = 0.5;

}

public void sayDing() {

System.out.println("Ding-ding!");

}

@Override

public double rideTime(double distance) {

return distance / this.speed;

}

@Override

public double rideCost(double distance) {

return distance \* personCoeff;

}

@Override

public double loadCost(double distance) {

return distance \* cargoCoeff;

}

@Override

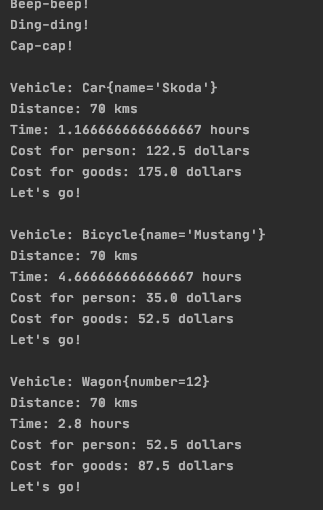
public String toString() {

return "Bicycle{" + "name='" + name + '\'' + '}';

}

}

**Рисунок с результатом работы программы**

****

**Задание 3**

**Код программы**

**Person**

package com.company;

public interface Person {

void sayHello();

}

**Admin**

package com.company;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

public class Admin implements Person {

private RailwayCash railwayCash;

@Override

public void sayHello() {

System.out.println("Hello! I'm an admin. I'm ready to help you.");

}

public Admin(RailwayCash railwayCash) { this.railwayCash = railwayCash;

}

public void addTrain(

String dayAndTime,

Integer number,

Integer seatsAmount,

ArrayList<String> stations,

Float pricePerSeat

) {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

Train train = new Train(

LocalDateTime.parse(dayAndTime, formatter),

number,

seatsAmount,

stations,

pricePerSeat

);

railwayCash.addTrains(train); }

}

**Passenger**

package com.company;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.Scanner;

public class Passenger implements Person {

@Override

public void sayHello() {

System.out.println("Hello! I'm a passenger. I'm looking for a train.");

}

public Request createRequest(String destination, String date)

{

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

Request request = new Request();

request.setDayAndTime(LocalDateTime.parse(date, formatter));

request.setDestination(destination);

return request;

}

public Bill chooseTrain(ArrayList<Train> trains) {

Scanner scanner = new Scanner(System.in);

for (Train train : trains) {

System.out.println(train);

}

System.out.println("Choose a train number: ");

Integer chosenNumber = scanner.nextInt();

boolean trainIsNotFound = true;

Train chosenTrain = null;

for (int i = 0; i < trains.size() && trainIsNotFound; i++) {

if (trains.get(i).getNumber().equals(chosenNumber)) {

chosenTrain = trains.get(i);

chosenTrain.reserveSeat();

trainIsNotFound = false;

}

}

if (chosenTrain != null) {

Bill bill = new Bill();

bill.setPrice(chosenTrain.getPricePerSeat());

bill.setSeatNumber(chosenTrain.getOccupiedSeatsAmount());

return bill;

} else {

throw new RuntimeException("Train is not found!");

}

}

}

**Main**

package com.company;

import java.util.ArrayList;

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

RailwayCash railwayCash = new RailwayCash();

Admin admin = new Admin(railwayCash);

admin.sayHello();

admin.addTrain(

"2019-10-21 14:00",

701,

500,

new ArrayList<String>(Arrays.asList("Жабинка", "Берёза", "Барановичи", "Минск")),

13.50f

);

admin.addTrain(

"2019-10-21 14:00",

703,

500,

new ArrayList<String>(Arrays.asList("Барановичи", "Минск" )),

15.50f

);

Passenger passenger = new Passenger();

passenger.sayHello();

Request request = passenger.createRequest(

"Барановичи",

"2019-10-21 14:00");

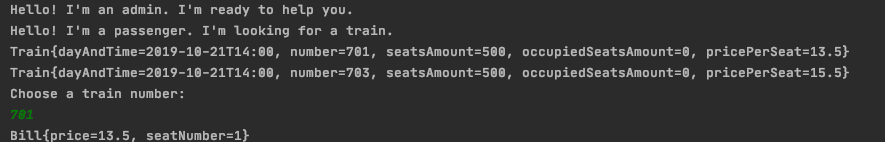
Bill bill = passenger.chooseTrain(railwayCash.findTrainsByRequest(request));

System.out.println(bill);

}

}

**Рисунок с результатом работы программы**



**Выводы**:

* приобрела практические навыки в области объектно-ориентированного проектирования.