МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине СПП

Тема: «Классы в программах на языке программирования Java»

Выполнил

студент группы ПО-5

Романюк. В. А.

Проверил: преподаватель

Крощенко А. А.

Брест, 2021

Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования.

Задание 1

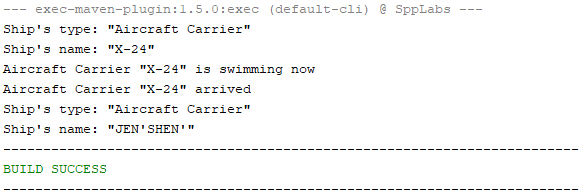
Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

7) interface Корабль 🡨 abstract class Военный Корабль 🡨 class Авианосец

Код программы

package com.company;  
  
public class Lab5Task1 {  
 public static void main(String[] args) {  
 AircraftCarrier ship1 = new AircraftCarrier("X-24");  
 ship1.print();  
 ship1.swim();  
 ship1.arrive();  
  
 AircraftCarrier ship2 = new AircraftCarrier("R4-BEE");  
 ship2.set\_name("JEN'SHEN'");  
 ship2.print();  
 }  
}  
  
interface Ship {  
 void print();  
 void set\_name(String name);  
 String get\_name();  
 String get\_type();  
 void swim();  
 void arrive();  
}  
  
abstract class MilitaryShip implements Ship {  
 String name;  
 String type;  
  
 MilitaryShip(String name) {  
 this.type = "Military Ship";  
 this.name = name;  
 }  
  
 @Override  
 public void set\_name(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String get\_name() {  
 return this.name;  
 }  
  
 @Override  
 public String get\_type() {  
 return this.type;  
 }  
  
 @Override  
 public void swim() {  
 System.out.println(this.type + " \"" + this.name + "\" is swimming now");  
 }  
  
 @Override  
 public void arrive() {  
 System.out.println(this.type + " \"" + this.name + "\" arrived");  
 }  
  
 @Override  
 public void print() {  
 System.out.println("Ship's type: \"" + this.type + "\"\nShip's name: \"" + this.name + "\"");  
 }  
}  
  
class AircraftCarrier extends MilitaryShip {  
 AircraftCarrier(String name) {  
 super(name);  
 this.type = "Aircraft Carrier";  
 }  
}

Тестирование

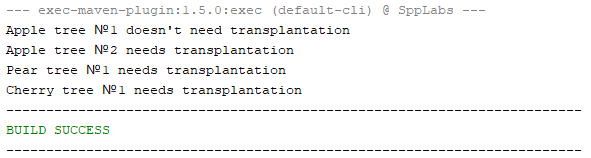


Задание 2

Код программы

package com.company;  
  
import java.util.\*;  
  
public class Lab5Task2 {  
 public static void main(String[] args) {  
 ArrayList<GardenTree> mass = new ArrayList<>();  
  
 AppleTree apple1 = new AppleTree(2, 0.6);  
 AppleTree apple2 = new AppleTree(7, 0.3);  
 PearTree pear1 = new PearTree(1, 0.3);  
 CherryTree cherry1 = new CherryTree(15, 0);  
  
 mass.add(apple1);  
 mass.add(apple2);  
 mass.add(pear1);  
 mass.add(cherry1);  
  
 mass.forEach(elem -> {  
 elem.needsTransplantation();  
 });  
 }  
}  
  
abstract class GardenTree {  
 int age;  
 double fruiting;  
  
 GardenTree(int age, double fruiting) {  
 this.age = age;  
  
 if (fruiting >= 0 && fruiting <= 1) {  
 this.fruiting = fruiting;  
 }  
 else {  
 System.out.println("Incorrect value. Try something [0.0, 1.0]");  
 }  
 }  
  
 int get\_age() {  
 return this.age;  
 }  
  
 double get\_fruiting() {  
 return this.fruiting;  
 }  
  
 void set\_age(int age) {  
 this.age = age;  
 }  
  
 void set\_fruiting(double fruiting) {  
 if (fruiting >= 0 && fruiting <= 1) {  
 this.fruiting = fruiting;  
 }  
 else {  
 System.out.println("Incorrect value. Try something [0.0, 1.0]");  
 }  
 }  
  
 void needsTransplantation() {  
 }  
}  
  
class AppleTree extends GardenTree {  
 static int number = 0;  
 int itsNumber;  
  
 AppleTree(int age, double fruiting) {  
 super(age, fruiting);  
 this.itsNumber = ++AppleTree.number;  
 }  
  
 void print() {  
 System.out.println("Apple-tree №" + this.itsNumber + " has:\n"  
 + "Age: " + this.age + "\n"  
 + "Fruiting: " + this.fruiting);  
 }  
  
 @Override  
 void needsTransplantation() {  
 if (this.age < 4 && this.fruiting > 0.5) {  
 System.out.println("Apple tree №" + this.itsNumber + " doesn't need transplantation");  
 }  
 else {  
 System.out.println("Apple tree №" + this.itsNumber + " needs transplantation");  
 }  
 }  
}  
  
class PearTree extends GardenTree {  
 static int number;  
 int itsNumber;  
  
 PearTree(int age, double fruiting) {  
 super(age, fruiting);  
 this.itsNumber = ++PearTree.number;  
 }  
  
 void print() {  
 System.out.println("Pear-tree №" + this.itsNumber + " has:\n"  
 + "Age: " + this.age + "\n"  
 + "Fruiting: " + this.fruiting);  
 }  
  
 @Override  
 void needsTransplantation() {  
 if (this.age < 6 && this.fruiting > 0.4) {  
 System.out.println("Pear tree №" + this.itsNumber + " doesn't need transplantation");  
 }  
 else {  
 System.out.println("Pear tree №" + this.itsNumber + " needs transplantation");  
 }  
 }  
}  
  
class CherryTree extends GardenTree {  
 static int number;  
 int itsNumber;  
  
 CherryTree(int age, double fruiting) {  
 super(age, fruiting);  
 this.itsNumber = ++CherryTree.number;  
 }  
  
 void print() {  
 System.out.println("Cherry-tree №" + this.itsNumber + " has:\n"  
 + "Age: " + this.age + "\n"  
 + "Fruiting: " + this.fruiting);  
 }  
  
 @Override  
 void needsTransplantation() {  
 if (this.age < 5 && this.fruiting > 0.7) {  
 System.out.println("Cherry tree №" + this.itsNumber + " doesn't need transplantation");  
 }  
 else {  
 System.out.println("Cherry tree №" + this.itsNumber + " needs transplantation");  
 }  
 }  
}

Тестирование



Задание 3

Код программы

Объявление класса

class Dispatcher {  
 ArrayList<Trip> trips;  
 ArrayList<Driver> drivers;  
 ArrayList<Car> cars;  
  
 public static void constructor (Trip[] trips.Driver[] drivers,Car[] cars) {  
 Dispatcher.trips = new ArrayLost<>Arrays.asList(trips);  
 Dispatcher.drivers = new ArrayLost<>Arrays.asList(drivers);  
 Dispatcher.cars = new ArrayLost<>Arrays.asList(cars);  
}  
  
 public static void addTrip(Trip trip) {trips.add(trip);  
  
 public static void addDriver(Driver driver) {drivers.add(driver);}  
  
 public statik void addCar(Car car) {cars.add(car)}  
  
 public void scheduleTrips() {  
 point1:  
 for (Trip trip : trips) {  
 point2:  
 if (!trip.scheduled) {  
 for (Driver driver : drivers) {  
 if (!driver.tripStarted && !driver.removed) {  
 for (Car car : cars) {  
 if (!car.taken && !car.needRepair) {  
 trip.scheduled = true;  
 driver.tripStarted = true;  
 car.taken = true;  
 driver.car = car;  
 driver.trip = trip;  
 break point2;  
 }  
 }  
  
 System.out.println("Все машины заняты или нуждаются в ремонте");  
 break point1;  
 }  
 }  
  
 System.out.println("Все водители заняты или отстранены от работы");  
 }  
 }  
 }

Использование класса

Dispatcher.constructor(new Trip []) (trip1, trip2, trip3, trip4, trip5),  
 new Driver[] {driver1, driver2, driver3, driver4, driver5},  
 new Car[] {car1, car2, car3, car4, car5});  
  
 dispatcher.scheduleTrips();  
 driver1.repairRequest();  
 driver1.finishTrip();  
 dispatcher.scheduleTrips();  
 driver1.finishTrip();  
 dispatcher.removeDriver(driver2);  
 driver2.report();  
}

Вывод: приобрел практические навыки в области объектно-ориентированного проектирования.