Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра системотехніки

Дисципліна: «Крос-платформне програмування»

**ЛАБОРАТОРНА РОБОТА № 2**

**«ПОСТРОЕНИЕ ИЕРАРХИИ КЛАССОВ»**

|  |  |  |
| --- | --- | --- |
| Виконав:  ст. гр. ІТКН-18-1  Варламов Микита Денисович |  | Прийняла:  Ст. лаб. Каф.  Калайда Н. С. |

Харків 2019

1. **Мета роботи:** Изучение возможностей объектно-ориентрованного программирования в языке Java. Получение практических навыков при создании иерархии классов с использованием языка Java, переопределения методов классов при наследовании, реализация стандартных интерфейсов.
2. **Варіанти задач:**

Задача 16: Спроектировать и реализовать иерархию классов воздушных судов.

1. **Ход роботи**

Я проанализировал классификацию воздушных судов и составил упрощенную UML диаграмму в программе Visual Paradigm. При разработке на Java использовал абстрактные классы, интерфейсы, интерфейсы Comparable и Comparator. Создавал массивы и коллекции, @Override.

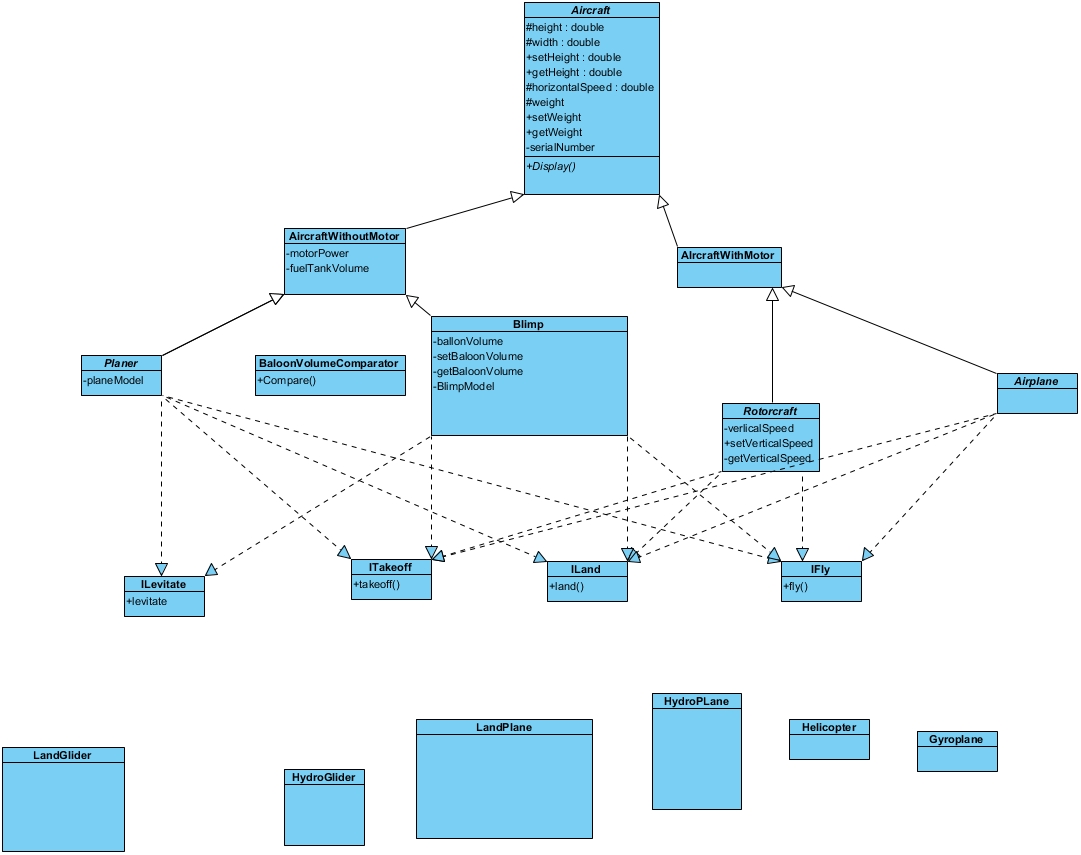


Рисунок 1. – UML діаграма

1. **Код программы:**

### Абстрактный класс, суперкласс Aircraft

package airplane\_hierarcy;

public abstract class Aircraft {

private double height;

private double width;

private double weight;

private double horizontalSpeed;

private double serialNumber;

public double getSerialNumber () {

return serialNumber;

}

public void setSerialNumber (double serial) {

this.serialNumber = serial;

}

public double getHeight () {

return height;

}

public void setHeight (double height) {

this.height = height;

}

public void setWidth (double width) {

this.width = width;

}

public double getWidth () {

return width;

}

public void setWeight (double weight) {

this.weight = weight;

}

public double getWeight () {

return weight;

}

public void setHorizontalSpeed (double speed) {

this.horizontalSpeed = speed;

}

public double getHorizontalSpeed () {

return horizontalSpeed;

}

public void Display () {

}

}

### AircraftWithoutMotor (воздушные судна без двигателя)

package airplane\_hierarcy;

public abstract class AircraftWithoutMotor extends Aircraft {

double motorPower;

double fuelTankVolume;

public double getMotorPower () {

return motorPower;

}

public void setMotorPower (double power) {

this.motorPower = power;

}

public double getfuelTankVoulme () {

return fuelTankVolume;

}

}

### BaloonVolumeComparator (объем баллона дирижабля)

package airplane\_hierarcy;

import java.util.Comparator;

public class BaloonVolumeComparator implements Comparator<Blimp>{

public int compare(Blimp b1, Blimp b2) {

if (b1.getBaloonVolume() == b2.getBaloonVolume()) {

return 0;

}

if (b1.getBaloonVolume() < b2.getBaloonVolume()) {

return -1;

}

else {

return 1;

}

}

}

### Blimp (дирижабль)

package airplane\_hierarcy;

public class Blimp extends AircraftWithoutMotor {

private double baloonVolume;

String blimpModel;

public double getBaloonVolume () {

return baloonVolume;

}

public void setBaloonVolume (double volume) {

this.baloonVolume = volume;

}

public Blimp (String blimpModel, double volume) {

this.setBaloonVolume(volume);

this.blimpModel = blimpModel;

}

@Override

public String toString () {

return "Name " + blimpModel;

}

}

### Planer

package airplane\_hierarcy;

import interfaces.IFly;

import java.util.Comparator;

public class Planer extends AircraftWithoutMotor implements IFly, Comparable<Planer>{

private String planerModel;

public Planer (String planerModel, double width, double height, double weight, double speed, double power) {

super.setWidth(width);

super.setHeight(height);

super.setWeight(weight);

super.setHorizontalSpeed(speed);

super.setMotorPower(power);

this.planerModel = planerModel;

}

public Planer (String name, double serial) {

super.setSerialNumber(serial);

this.planerModel = name;

}

public void fly() {

System.out.printf("I can fly");

}

/\*

public int compareTo(Planer p) {

return ((int)super.getWeight()).compareTo(((int)p.getWeight()));

}

\*/

@Override

public String toString () {

return "Name " + planerModel;

}

@Override

public int compareTo(Planer p) {

// TODO Auto-generated method stub

if (this.getSerialNumber() == p.getSerialNumber()) {

return 0;

} else if (this.getSerialNumber() < p.getSerialNumber()) {

return -1;

} else {

return 1;

}

}

}

### IFly (интерфейс полета)

package interfaces;

public interface IFly {

void fly();

}

### Program (функция main)

package airplane\_hierarcy;

import java.util.ArrayList;

import java.util.TreeSet;

public class Program {

public static void main(String[] args) {

Planer planer = new Planer("airbus", 5.555555, 5.5555, 10.00000, 80.000, 12.4000);

System.out.println(planer.getHorizontalSpeed());

planer.fly();

System.out.println();

System.out.println(planer.toString());

Planer [] planers = new Planer[2];

planers [0] = new Planer ("cat", 122.999);

planers [1] = new Planer ("dog", 123.999);

TreeSet<Planer> planerList = new TreeSet<Planer>();

planerList.add(planers [0]);

planerList.add(planers [1]);

System.out.println();

//Array.sort(planers);

for(Planer p: planerList) {

System.out.println(p);

}

System.out.println();

ArrayList<Blimp> blimpList = new ArrayList <Blimp>();

Blimp zilla = new Blimp ("zilla", 350.50);

Blimp golla = new Blimp ("golla", 280.00);

blimpList.add(zilla);

blimpList.add(golla);

for( Blimp b: blimpList) {

System.out.println(b);

}

System.out.println();

BaloonVolumeComparator myVolumeComp = new BaloonVolumeComparator();

blimpList.sort(myVolumeComp);

for( Blimp b: blimpList) {

System.out.println(b);

}

}

}

1. **Результат роботи програми:**

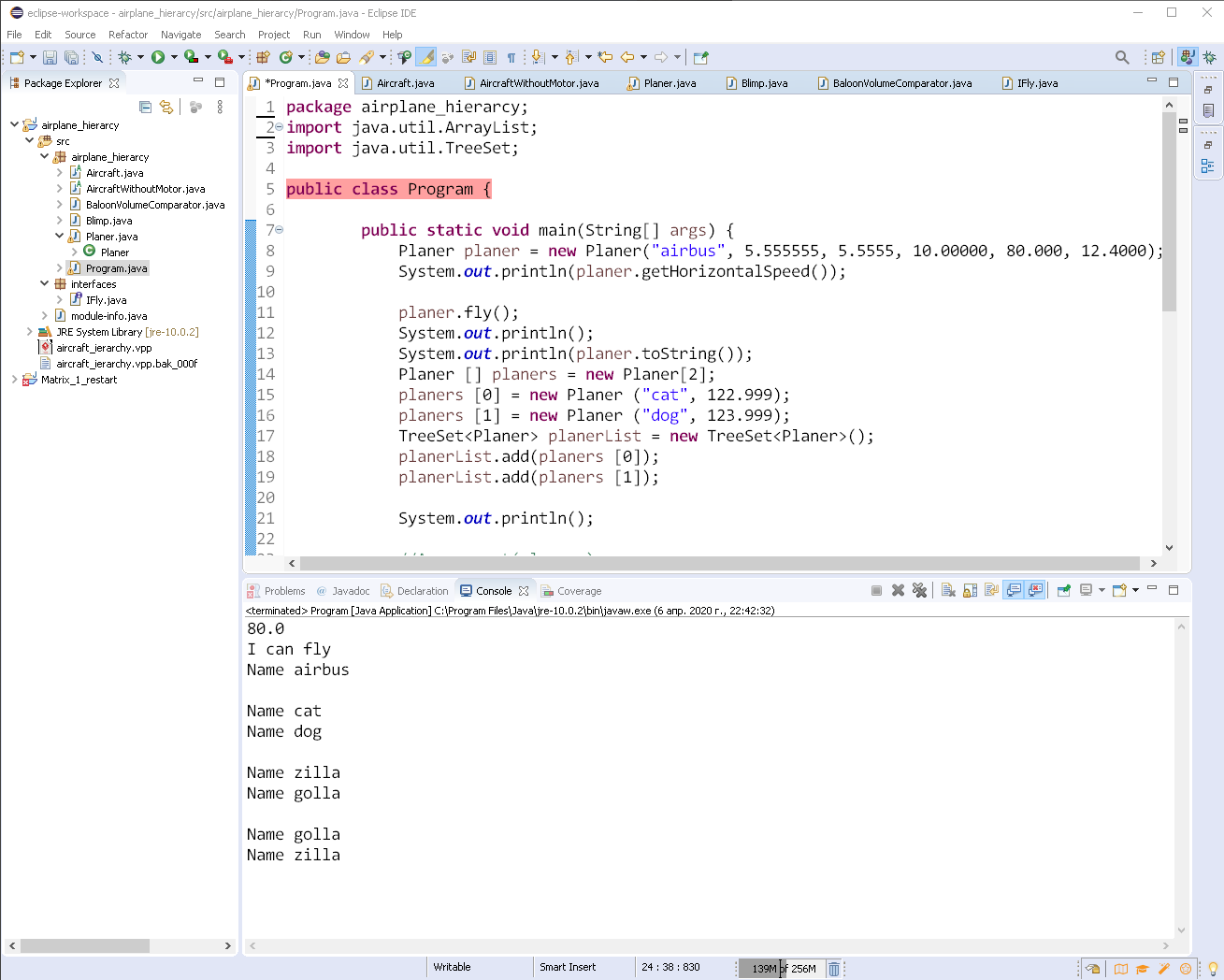


Рисунок 2. – Результат роботи програми

1. **Висновки: повторив правила ієрархії наслідування, використовував гетери та сетери для інкапсуляції, проаналізував класифікацію повітряних суден і склав спрощену UML діаграму в програмі Visual Paradigm. При розробці на Java використовував абстрактні класи, інтерфейси, інтерфейси Comparable і Comparator. Створював масиви і колекції, @Override переоприділив метод toString. повторив правила ієрархії наслідування, роботу абстрактних класів, інтерфейсів. При роботі з інтерфейсами comparable зіткнувся з проблемою що compareTo не працює з примітивними типами double, аде я вирішив проблемуреалізувавши compareTo вручну.**