Анализ кода программы “ airport_network ”.

Минусы написанного кода.

1) В классе Airport присутствует массив вариантов названий аэропортов, его присутствие и использование вызывает следующие проблемы:

- Чтобы добавить новое название аэропорта необходимо либо изменять существующий класс, либо создавать класс потомок в котором создавать второй массив с названиями аэропортов.

- Все объекты этого класса будут содержать данный массив, то есть содержать лишнюю информацию и занимать лишний объем оперативной памяти.

- Используя только это класс, невозможно создать больше объектов класса с разными названиями, чем указано имен в массиве.

На наш взгляд, лучшем вариантом было бы хранить название аэропортов в файле, а при создании объекта класса передавать в конструктор нужное название аэропорта.

```
import java.util.*;

public class airport {
    private static String []airportsNames = new String[] { //массив имен аэропортов
        "Шереметьево",
        "Домодедово",
        "Толмачево",
        "Пулково",
        "Сокол",
        "Алтай",
        "Хитроу",
        "Адлер_Сочи",
        "Пушкино",
        "Внуково",
        "Шереметьево-2",
        "Девау"
    };
    public static int getMaxAirportNumber() {
        return airportsNames.length;
    }
```

Те же самые проблемы касаются и класса basicPlane, в котором присутствует массив названий самолетов.

```
import java.util.*;

public class basicPlane {
    private static String[] namesArray = new String[]{ //массив имен самолетов
                                "АЭРОБУС А310",
                                "АЭРОБУС А320",
                                "Лучший самолет",
                                "Ту-144",
                                "Cessna 850",
                                "An-225",
                                "Boing 747400",
                                "A380",
                                "Charrikein",
                                "Junkers",
                                "FW-190",
                                "Dornier Do X",
                                "Convair B-36J"
                               };
    private static int[] colvoSameNames = new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    private int weight;
```

2) В классе Airport присутствуют следующие поля

```
private int coordX, coordY; //координаты аэропорта
```

На наш взгляд, координаты аэропорта лучше было бы хранить в классе Point.

3) В классе Airport присутствуют следующие поля

```
private int airportType; //тип самолета: 1-военный, 0-общий
private int runwayStatus; //показатель свободности взлетно-посадочной полосы
```

Проблемы, связанные с использованием этих полей:

- Для того, чтобы программисту использовать эти поля в коде, ему необходимо постоянно помнить какая цифра что обозначает, либо открывать класс для просмотра комментариев;
- Переменная runwayStatus не содержит комментариев о том, какая целая константа что обозначает, поэтому программистом остаются только догадываться об этом;
- Программисту придется использовать магические цифры;
- **Самая большая проблема в том что в сеттерах для этих переменных никак не контролируется какие числа туда передаются через параметры, в результате если программист в каком-либо участке кода передаст неверное значение переменной, то программа начнет неправильно функционировать и/или «вылетать».**

На наш взгляд, лучшим решением было бы создать следующие конструкцию

```
public static enum TypePlane(COMMON_PLANE, WAR_PLANE);

private   TypePlane type;
```

4) Во всех сеттерах, встречающихся в программе, никак не контролируются передаваемые в параметрах значения.

В результате отсутствия проверки переменные можно случайно испортить, например: в классе Airport переменным coordX, coordY (координаты аэропорта) передать отрицательное значение или в том же классе переменной flyingPlanes (переменная хранит самолеты, летящие в данный аэропорт) передать значение NULL. Порча этих переменных может привести к неправильному функционированию и/или «вылетам» программы.

Возникает вопрос: зачем вообще нужны сеттеры в классах, если они не контролируют передаваемые в них значения? Разработчику можно было бы сделать все переменные с модификатором доступа public и от этого ничего бы не поменялось.

Эта же проблема касается и всех конструкторов, встречающихся в программе.

5) Встречаются магические числа смысл которых не объяснен комментариями.

Например: конструктор класса Airport.

```
public airport (int airportCoordX, int airportCoordY) {
    airportsName=airportsNames[airportNumber];
    coordX=airportCoordX;
    coordY=airportCoordY;
    Random rand = new Random();
    if (airportNumber>=2) { //определение типа аэропорта (как минимум 2 аэропорта должны быт
        airportType=rand.nextInt(2);
    }
    else {
        airportType=0;
    }
    runwayStatus=0;
    parkingPlacesNumber=5+rand.nextInt(31); //определение количества стояночный мест
    int startingPlanesNumber=rand.nextInt(parkingPlacesNumber);//определение количества стоя
    //начальное заполнение аэропорта
    parkingPlanes = new Vector<basicPlane>();
    flyingPlanes = new Vector<basicPlane>();
    for (int i=0; i<startingPlanesNumber; i++) {
        if (airportType==0) {
            switch (rand.nextInt(9)) { //определение типа самолета
```

конструктор класса BasicPlane.

```
public basicPlane () {
    Random rand = new Random();
    weight=rand.nextInt(90000)+10000;
    speed=rand.nextInt(1000)+100;
    maxTime=rand.nextInt(50)+10;
    stopTime=rand.nextInt(4)+1;
    int nameNumber=rand.nextInt(namesArray.length-1);
    colvoSameNames[nameNumber]++;
    name=namesArray[nameNumber] + " #" + Integer.toString(colvoSameNames[nameNumber]);
    finishAirport=null;
    startAirport=null;
}
```

6) Не все переменные имеют комментарии.

Например, в классе airport есть комментарий для каждой переменной.

```
private int coordX, coordY; //координаты аэропорта
private int airportType; //тип самолета: 1-военный, 0-общий
private int runwayStatus; //показатель свободности взлетно-посадочной полосы
private int parkingPlacesNumber; //количество стоянок для самолетов
private Vector<basicPlane> parkingPlanes; //самолеты, стоящие на данном аэропорту
private Vector<basicPlane> flyingPlanes; //самолеты, летящие в данный аэропорт
private String airportsName; //название аэропорта
```

В классе basicPlane комментарии встречаются не у каждой переменной

```
private static int[] colvoSameNames = new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
private int weight;
private int speed;
private int maxTime; //сколько максимум может лететь самолет
private int tekTime; //сколько осталось лететь до пункта назначения
private int stopTime;
private String name;
private String startAirport;
```

7) Отсутствие конструктора копирования во всех классах.

8) Все классы представляют из себя только набор конструкторов, сеттеров и геттеров.

9) Кое-где переносы длинных строк отсутствуют, приходится использовать прокрутку

Например: конструктор класса basicPlane

```
45      }
46      public basicPlane(int newWeight, int newSpeed, int newMaxTime, int newTekTime, int newStopTime, Stri
47          weight=newWeight;
48          speed=newSpeed;
```

10) Переменные во всех классах имеют модификатор доступа private, в результате чего наследуемые классы не будут видеть этих переменных.

На наш взгляд, уместней было бы использовать модификатор доступа protected.

11) Не используется спецификатор final для передаваемых в сеттеры параметров.

12) Используются сгенерированные методы и названия объектов во всех формах.

Например, в форме NetworkForm встречается следующие

```
    // Variables declaration - do not modify
    private javax.swing.JButton jButtonExit;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JList jList1;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTable jTable1;
    private javax.swing.JToggleButton jToggleButton1;
    // End of variables declaration
}
```

```
    private void jList1MouseClicked(java.awt.event.MouseEvent evt) {
        if (enabled==true) {
            int number=jList1.getSelectedIndex();
            if (number==0) new flyingPlanesForm(flyingPlanes).setVisible(true);
            else new airportForm(airports.get(number-1)).setVisible(true);
        }
    }

    private void jButtonExitActionPerformed(java.awt.event.ActionEvent evt) {
        UIManager.put("OptionPane.yesButtonText", "Да");
        UIManager.put("OptionPane.noButtonText", "Нет");
        int answer = JOptionPane.showConfirmDialog(null, "Вы уверены, что хотите выйти из программы?",
        if (answer == JOptionPane.YES_OPTION) {
            System.exit(0);

        }
```

13) Есть метод step в форме NetworkFrom имеющий размер **136 строк**.

14) Под формой NetworkForm есть описание классов ResultTableModel и planesCompare.

На наш взгляд, целесообразнее было бы оформить эти классы в отдельных файлах.

15) Краткие и не значащие ничего названия переменных в методах.

Например, в форме NetworkForm в методе step встречается на 300 строке следующее определение переменных.

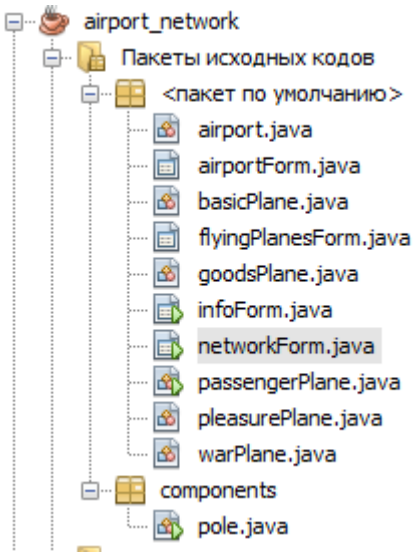
```
int dx=Math.abs(xFinish-xStart), dy=Math.abs(yFinish-yStart);
```

Также, в классе NetworkForm в методе formWindowOpened на 349 строке встречается следующее определение переменных

```
int xi, yi;
```

Стоит отметить, что этот недостаток кода встречается очень редко.

16) Все классы и формы хранятся в пакете по умолчанию.



На наш взгляд, классы и формы для удобства можно было поместить в разные пакеты.

Подведение итогов: Создается впечатление, что многие классы создавались для "галочки", так как кроме конструкторов, геттеров и сеттеров в классах ничего больше нет, в их сеттерах и конструкторах отсутствуют проверки передаваемых параметров. Также, во всех классах отсутствуют конструкторы копирования.

Плюсы написанного кода.

1) Нет закомментированных участков кода.

2) Код выдержан преимущественно в едином стиле.

3) Проект организован в виде классов.

4) Наименования классов, полей, методов, параметров и переменных информативны.

5) Отсутствует дублирование кода.

6) Практически все переменные в методах имеют информативное название.

Оценка кода: На наш субъективный взгляд, код, несмотря на ряд достоинств, имеет большое количество явных недостатков. Мы считаем, что корректно будет оценить его на 40 баллов по 100-бальной шкале.