Вложенные классы

- статические вложенные классы
- «обычные» вложенные классы
- локальные классы
- анонимные классы

Если теорию или примеры по теме вложенные классы Вы сейчас не поймете, это не страшно:) К данной теме мы еще раз вернемся при обработке событий в GUI

Вложенные static-классы

- внутри класса объявляется другой класс, который видит его скрытые элементы
- в отличие от нестатических вложенных классов не имеет привязки к конкретному объекту (при необходимости можно добавить вручную)

Вложенные классы (не static)

- объекты таких классов имеют ссылку на экземпляр внешнего класса
- идеально подходят при обработке событий для создания listener-ов (классы, которые с одной стороны имеют заданный интерфейс, а с другой стороны имеют доступ к приватным элементам внешнего класса)

Пример 1 (static inner classes)

```
class Outer {
   private static int staticInt = 1;
   private int nonstaticInt = 2;

   static class Inner {
       Outer o;
       Inner(Outer o) {
           this.o = o;
       }

       void doSomething () {
           staticInt++;
           o.nonstaticInt++;
       }
   }
}
```

```
public class Main {

   public static void main(String[] args) {
       Outer o = new Outer();
       Outer.Inner i = new Outer.Inner(o);
       i.doSomething();
   }
}
```

Пример 2 (non-static inner classes)

```
class Outer {

   private static int staticInt = 1;
   private int nonstaticInt = 2;

   class Inner {

      void doSomething() {
         staticInt++;
         nonstaticInt++;
         // Outer “this” from Inner
         Outer.this.nonstaticInt++;
      }
   }
}
```

```
public class Main {

   public static void main(String[] args) {
      Outer o = new Outer();

      // create Inner outside Outer
      Outer.Inner i = o.new Inner();
      i.doSomething();
   }
}
```

Вложенные классы: пример 3 (часть 1/2)

```
import java.awt.*;
import java.awt.event.*;

class MyBase {}

public class Main extends MyBase{
    private Frame f;

    public static void main(String[] args) {
        Main m = new Main();
    }
```

Вложенные классы: пример 3 (часть 2/2)

```
    Main() {
        f  = new Frame("Inner classes");
        f.setSize(200,100);
        f.setVisible(true);
        f.addWindowListener(new MyWindowAdapter());
    }

    class MyWindowAdapter extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            f.dispose();
        }
    }

}
```

Локальные классы

- Описывается внутри метода
- Имеет доступ к полям внешнего объекта
- Имеет доступ к final-переменным этого метода

```
abstract class A {
    abstract void print();
}

public class Main {
    private int instanceVariable=42;

    public A createA(int x) {
        final int localVariable1 = x;
        final int localVariable2 = x+1;
        int localVariable3 = x+2;

        class B extends A {
            int myOwnVariable = 12; // Only one variable!!!
            void print() {
                System.out.println("instanceVariable = " +
                        instanceVariable);
                System.out.println("localVariable1 = " +
                        localVariable1);
```

```
                    System.out.println("localVariable2 = " +
                            localVariable2);
                    /*System.out.println("localVariable2 = " +
                            localVariable3); //!ERROR*/
                }
            }
            return new B();
        }

        public static void main(String[] args) {
            Main m = new Main();
            A a = m.createA(1);
            a.print();
        }
}
```

Анонимные классы

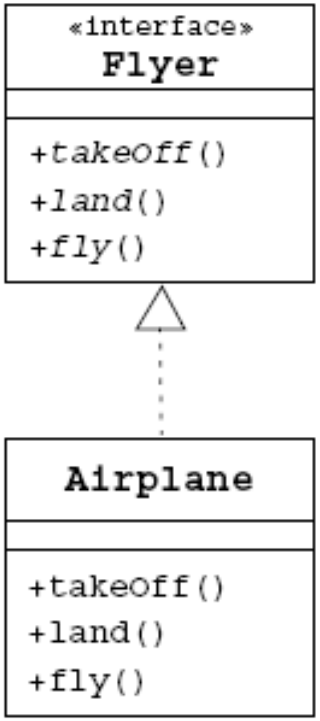
- класс-наследник, описываемый при создании объекта
- в исходном тексте программы имя в явном виде не присутствует – поэтому анонимный
- как всякий класс имеет уникальное имя, которое можно узнать с помощью getClass().getName()
- идеально подходят для создания небольших listener-ов (обработка событий)

```
class A {
    void printClassName() {
        System.out.println("My class name is 'A'");
    }
    void printRealClassName() {
         System.out.println("My real class name is " +
                 getClass().getName());
    }
}

public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.printClassName();
        a.printRealClassName();

        A b = new A() {
            void printClassName() {
                System.out.println("My class name is anonymous");
            }
        };
        b.printClassName();
        b.printRealClassName();
    }
```

ИНТЕРФЕЙСЫ

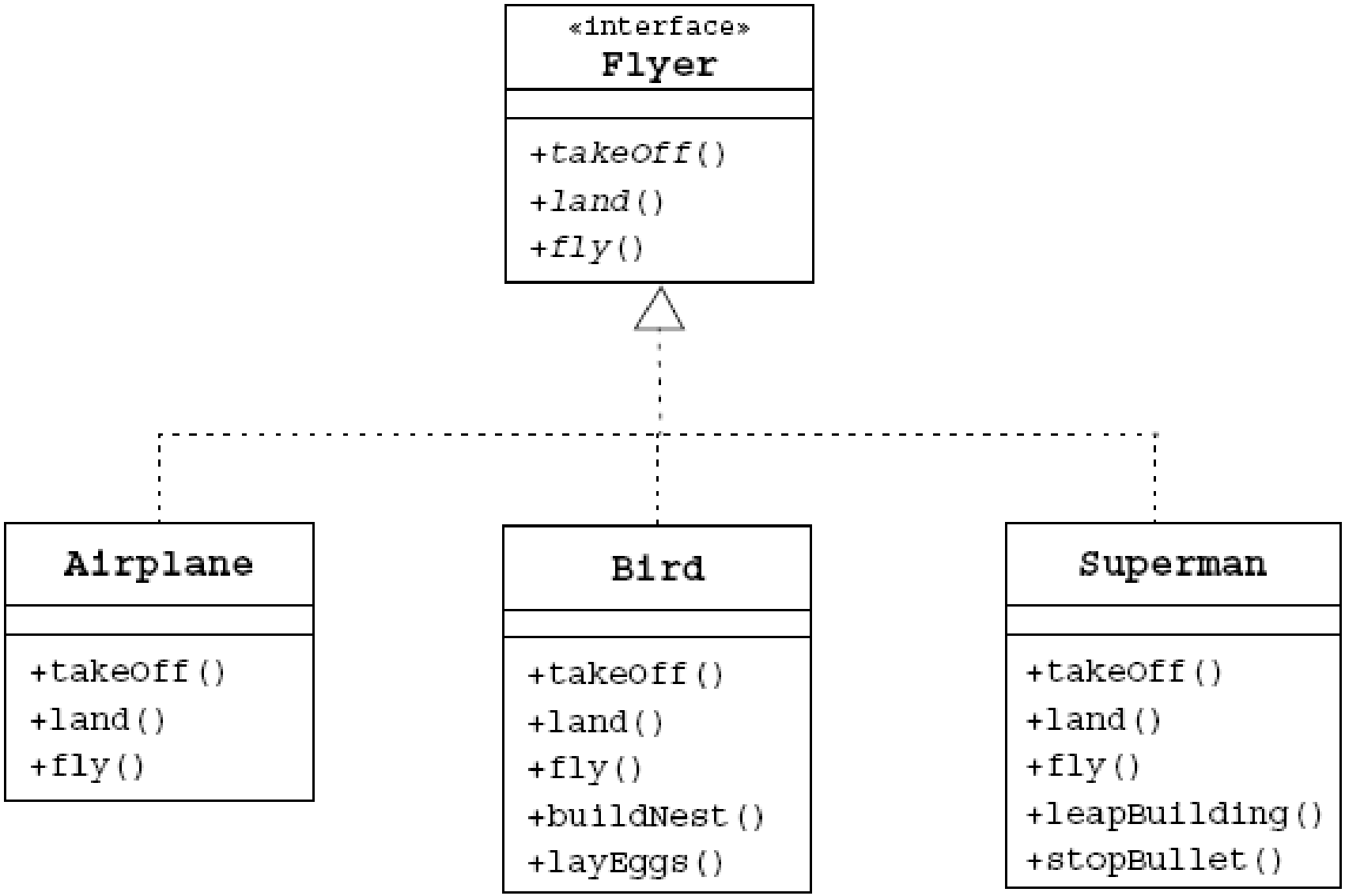
- множественного наследования в Java нет
- класс может иметь только одного непосредственного предка
- при этом класс может реализовывать любое количество интерфейсов

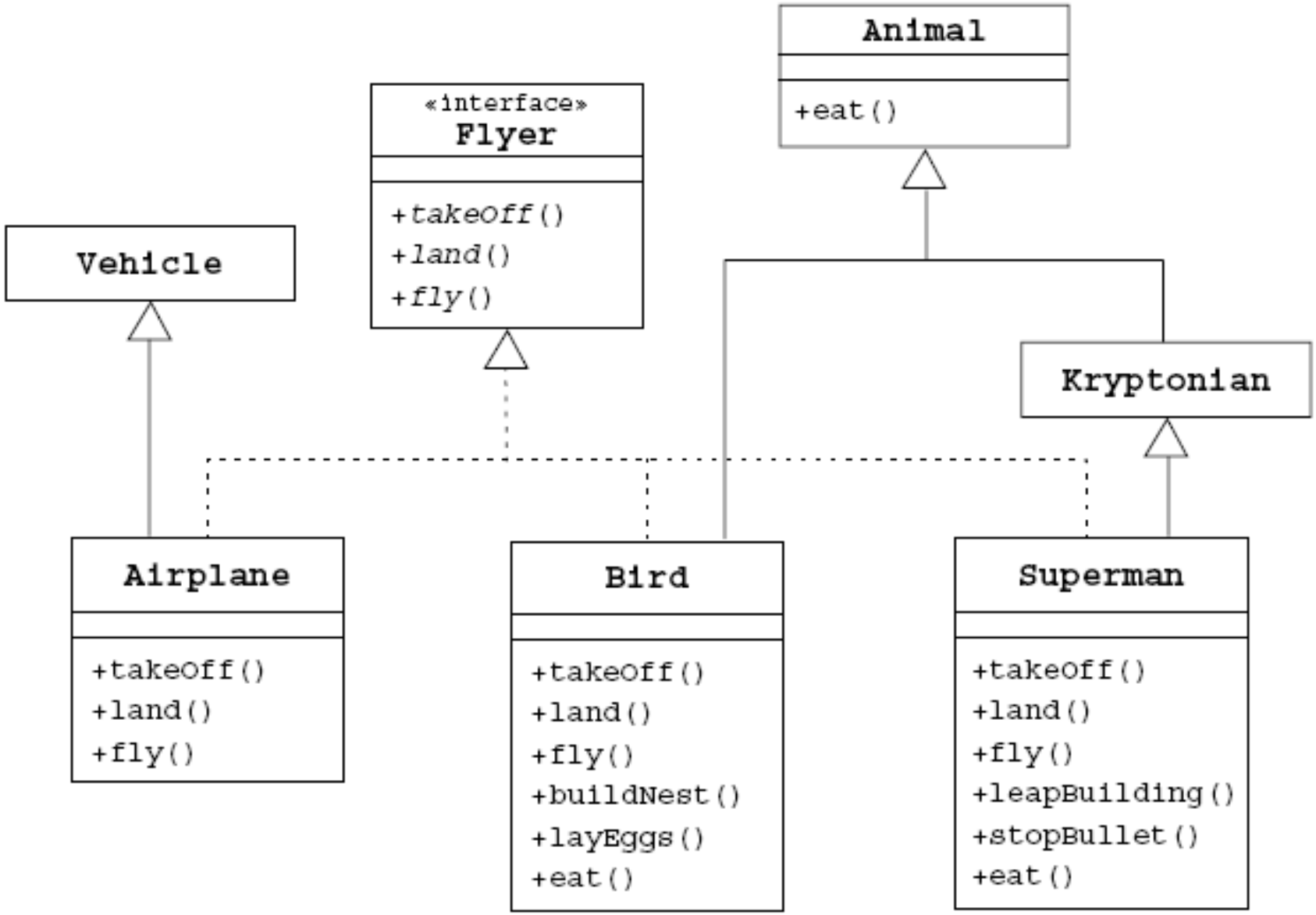
```
<modifier> interface <name> [extends <interface>
[,<interface>]* ] {
   <member_declaration>*
}
```




```
public interface Flyer {
  public void takeOff();
  public void land();
  public void fly();
}
```

```
public class Airplane implements Flyer {
  public void takeOff() {
    // accelerate until lift-off
    // raise landing gear
  }
  public void land() {
    // lower landing gear
    // decelerate and lower flaps until touch-down
    // apply brakes
  }
  public void fly() {
    // keep those engines running
  }
}
```


«interface»
Flyer
+takeOff()
+land()
+fly()
Airplane
+takeOff()
+land()
+fly()
Bird
+takeOff()
+land()
+fly()
+buildNest()
+layEggs()
Superman
+takeOff()
+land()
+fly()
+leapBuilding()
+stopBullet()


Animal
+eat()
«interface»
Flyer
+takeOff()
+land()
+fly()
Vehicle
Kryptonian
Airplane
+takeOff()
+land()
+fly()
Bird
+takeOff()
+land()
+fly()
+buildNest()
+layEggs()
+eat()
Superman
+takeOff()
+land()
+fly()
+leapBuilding()
+stopBullet()
+eat()

- интерфейс может иметь любое количество предков

```
interface X {}
interface Y {}
interface Z extends X, Y {}
```

- все поля в интерфейсе (даже если это явно не указано) – **public final static**
- все методы в интерфейсе (даже если это явно не указано) – **public abstract**

- нельзя создать экземпляр интерфейса
- можно создать анонимный класс на базе интерфейса используя сокращенную форму записи:

```
interface MyInterface {
    void myMethod();
}

public class Main {

    public static void main(String[] args) {

        MyInterface mi = new MyInterface() {
            public void myMethod() {
                System.out.println(
this.getClass().getSuperclass().getName());
            }
        };
        mi.myMethod();
    }
}
```

Задания

1. Построить три класса (базовый и 2 потомка), описывающих некоторых работников с почасовой оплатой (один из потомков) и фиксированной оплатой (второй потомок). Описать в базовом классе абстрактный метод для расчета среднемесячной заработной платы. Для «повременщиков» формула для расчета такова: «среднемесячная заработная плата = 20.8 * 8 * почасовую ставку», для работников с фиксированной оплатой «среднемесячная заработная плата = фиксированной месячной оплате»

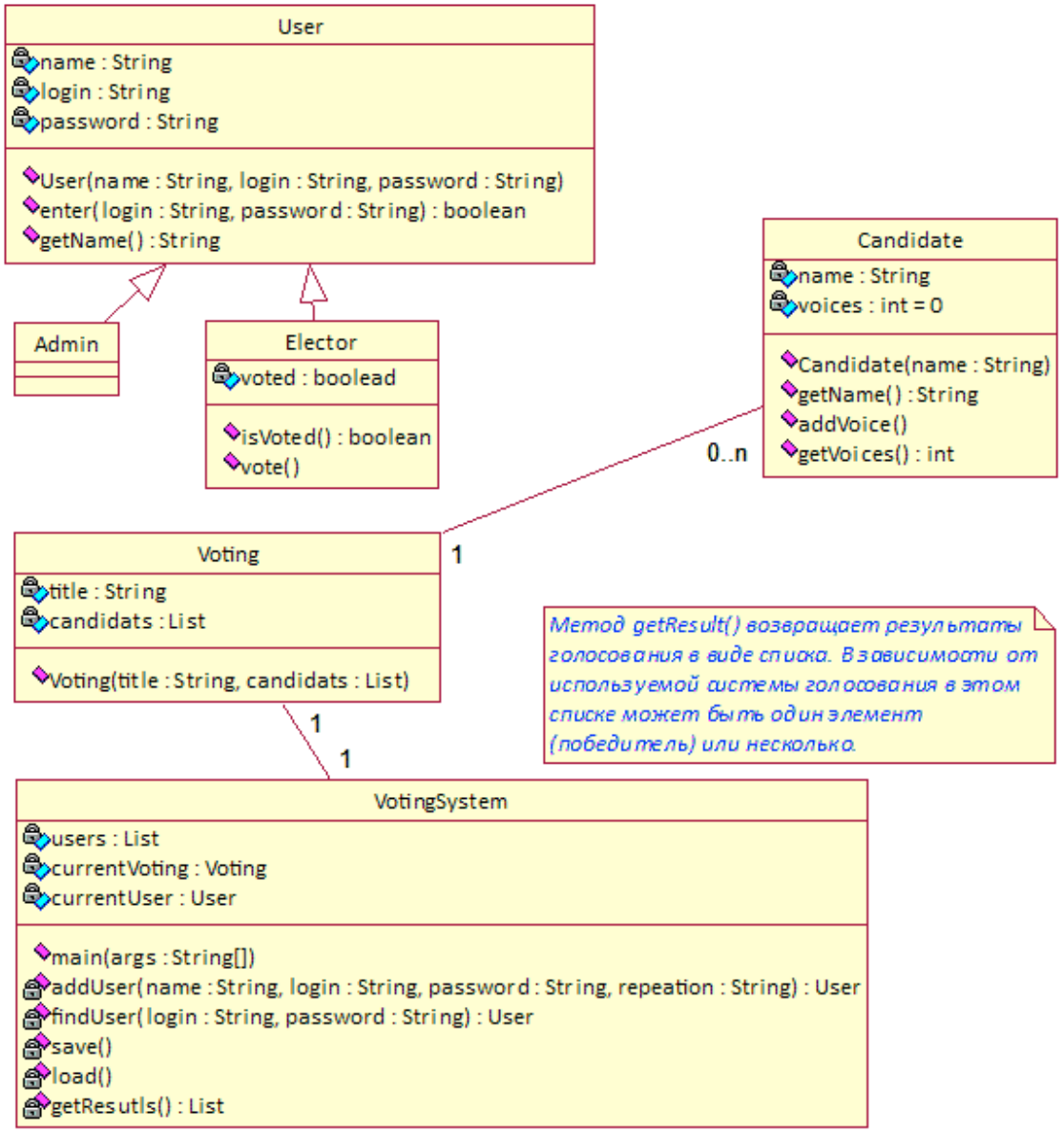
a) Упорядочить всю последовательность работников по убыванию среднемесячного заработка. При совпадении зарплаты – упорядочивать данные по алфавиту по имени. Вывести идентификатор работника, имя и среднемесячный заработок для всех элементов списка.
b) Вывести первые 5 имен работников из полученного в пункте a) списка.
c) Вывести последние 3 идентификатора работников из полученного в пункте a) списка.

2. Написать программу для проведения выборов. Избиратели должны предварительно зарегистрироваться в системе. Администратор заполняет список кандидатов. Каждый участник (по сети) или с того же самого компьютера входит в систему и голосует. Данные о проголосовавших накапливаются в базе данных. Дважды проголосовать нельзя. По окончании периода голосования администратор запускает процедуру подсчета голосов и система выдает результат.

Основные алгоритмы

Подведение итогов выборов.

Рекомендуемая диаграмма классов


User
name : String
login : String
password : String
User(name : String, login : String, password : String)
enter(login : String, password : String) : boolean
getName() : String
Admin
Elector
voted : boolead
isVoted() : boolean
vote()
Candidate
name : String
voices : int = 0
Candidate(name : String)
getName() : String
addVoice()
getVoices() : int
0..n
1
Voting
title : String
candidats : List
Voting(title : String, candidats : List)
1
1
Метод getResult() возвращает результаты голосования в виде списка. В зависимости от используемой системы голосования в этом списке может быть один элемент (победитель) или несколько.
VotingSystem
users : List
currentVoting : Voting
currentUser : User
main(args : String[])
addUser(name : String, login : String, password : String, repeation : String) : User
findUser(login : String, password : String) : User
save()
load()
getResutls() : List