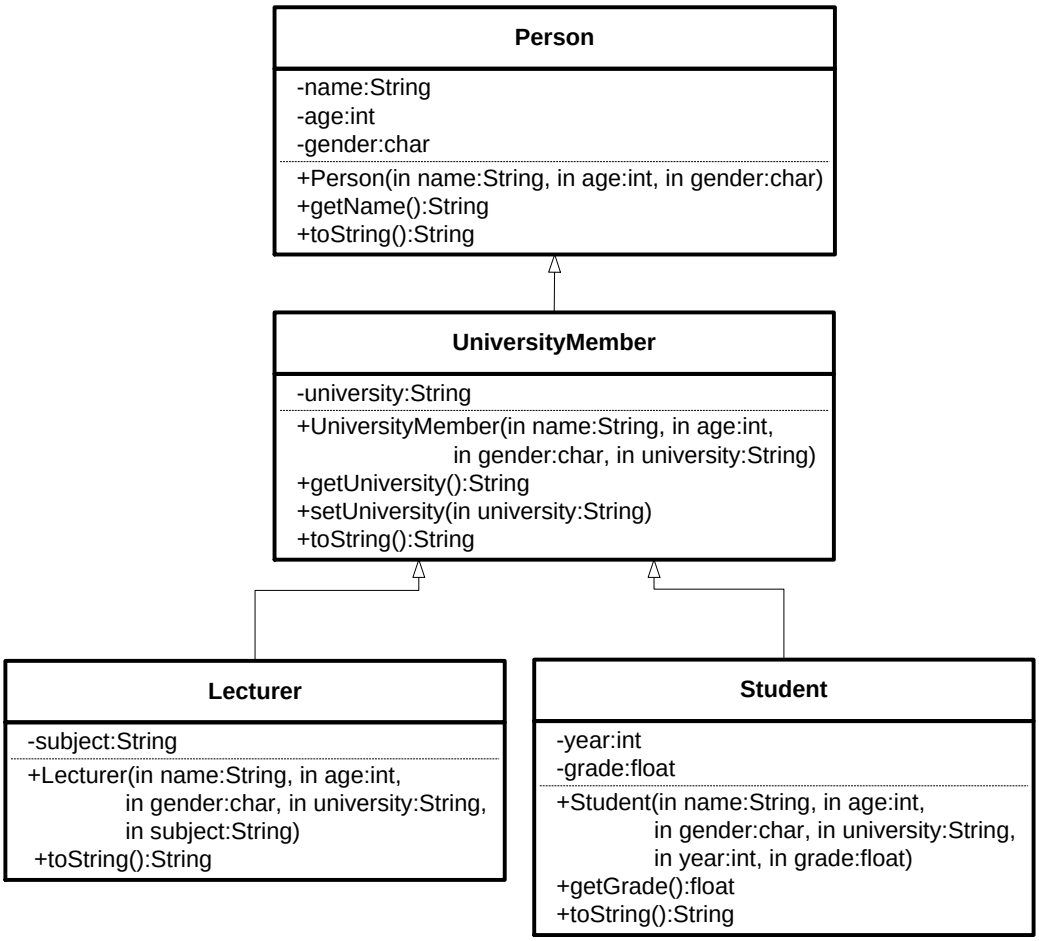
Упражнение №1

Класове. Наследяване. Полиморфизъм чрез наследяване.

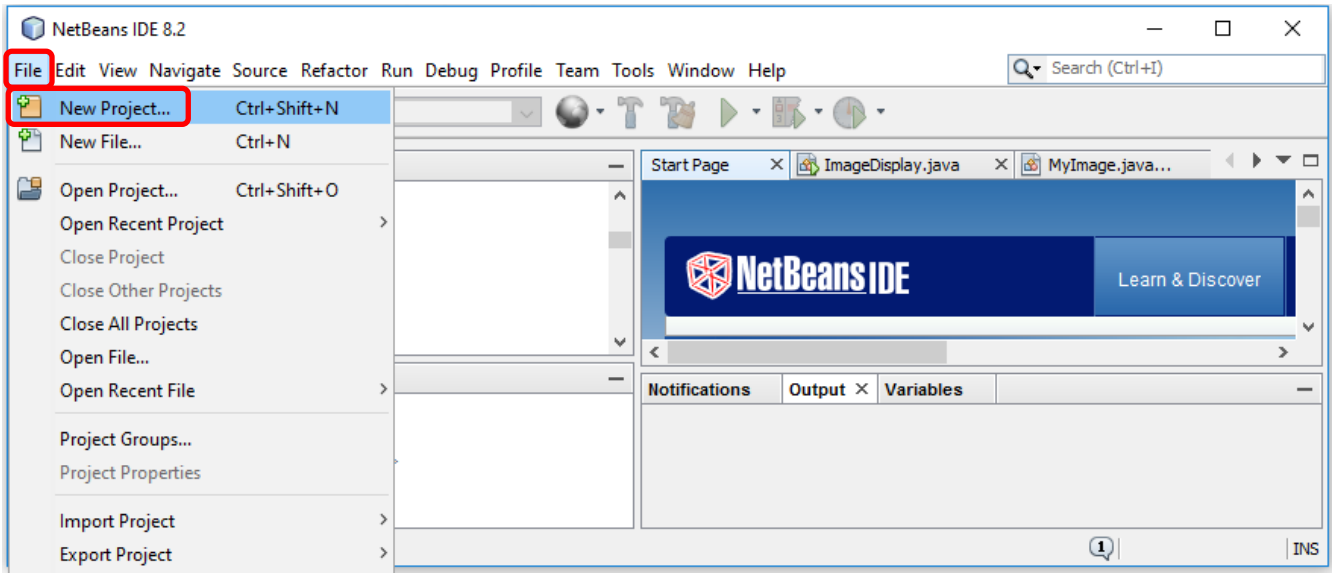
Създайте следната йерархия от класове:



I. Стартирайте NetBeans.

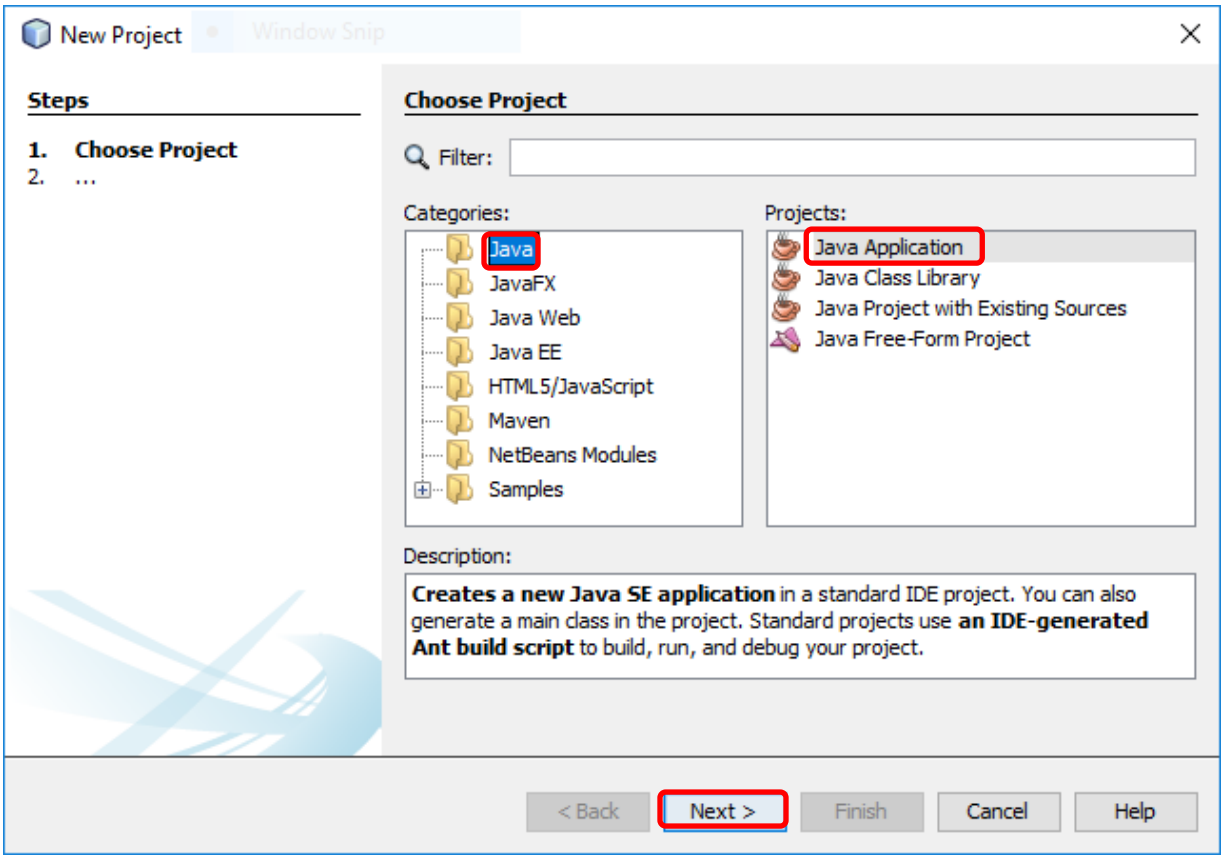
1. Създайте нов проект:

   `File -> New Project...`



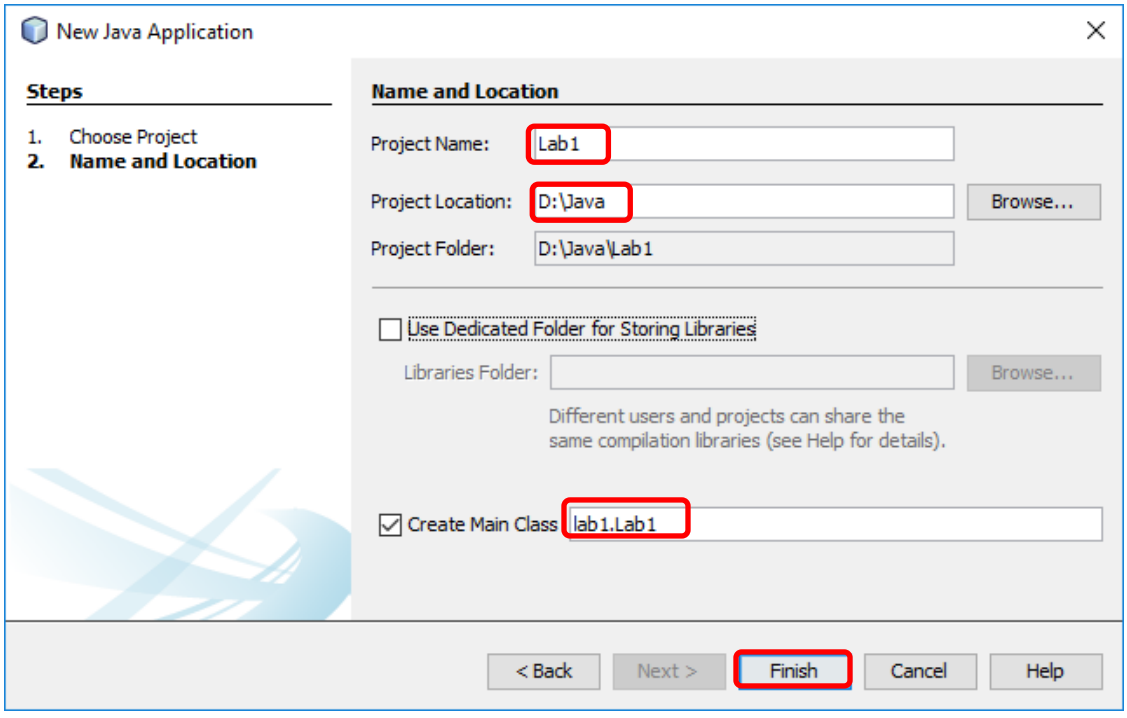
2. В диалоговия прозорец **`New Project`** изберете:

   `Categories:` **`Java`**
   `Projects:` **`Java Application`**

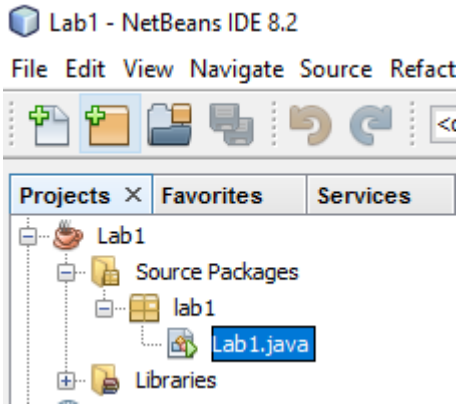


3. В диалоговия прозорец **`New Java Application`** изберете името и директорията на проекта:

   `Project Name:` **`<име на проект>`**
   `Project Location:` **`<директория на проекта>`**
   ☑ `Create Main Class` **`<име на пакет>.<име на тестов клас>`**



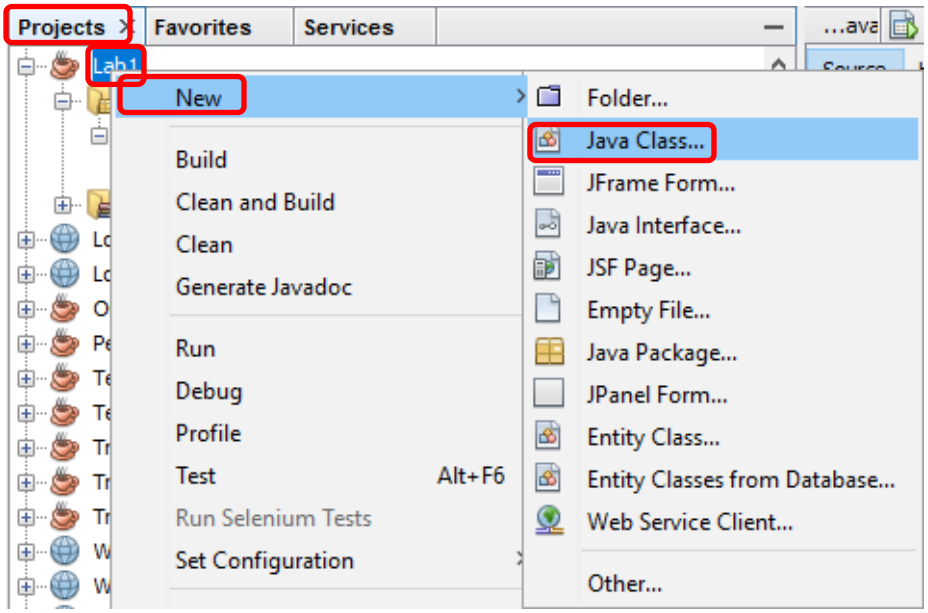
В менюто **`Projects`** се създава проект **`Lab1`**, пакет **`lab1`** и тестов клас **`Lab1`**.




```
package lab1;
public class Lab1 {
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

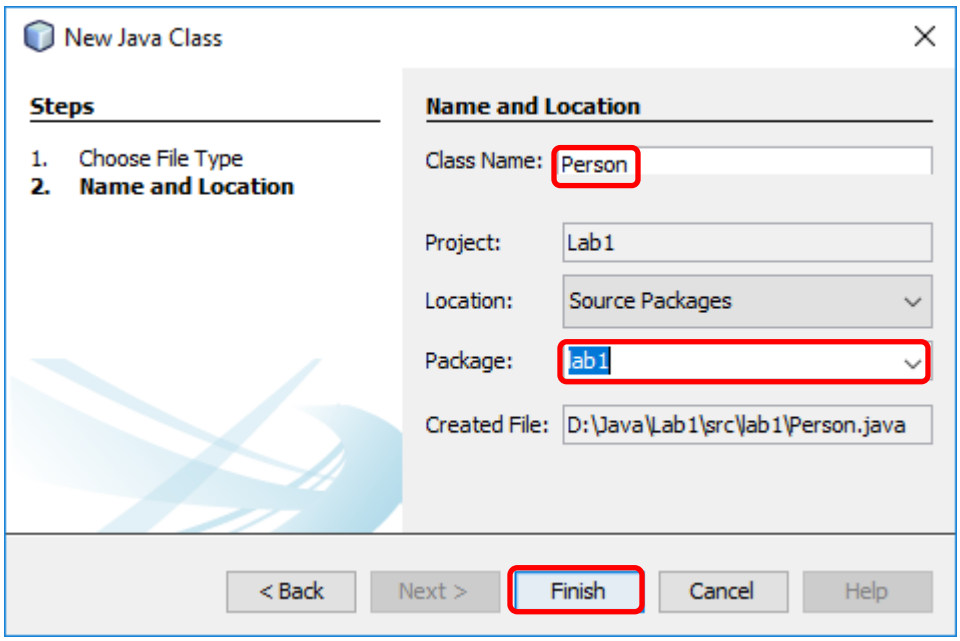
II. Създайте клас **`Person`** (**`Личност`**).

`Projects -> Lab1 -> New -> Java Class...`



1. В диалоговия прозорец **`New Java Class`** изберете името и пакета на класа:

   `Class Name:` **`<име на клас>`**
   `Package:` **`<име на пакет>`**




```
package lab1;
public class Person {

}
```

2. Декларирайте полета с **private** достъп: име **name** от тип **String**, възраст **age** от тип **int** и пол **gender** от тип **char**.
3. Добавете конструктор с три параметъра за инициализиране на полетата.
4. Напишете метод за достъп **getName()**, който връща името на личността.
5. Предефинирайте метода **toString()** на класа **Object**, който връща символното представяне на типа.

```
package lab1;
public class Person {                          // Клас Личност
    private String name;                       // име
    private int age;                           // възраст
    private char gender;                       // пол: М/Ж
    // Конструктор с три параметъра
    public Person(String name, int age, char gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
    public String getName() {                  // Метод за достъп
        return name;                           // връща името на личността
    }
    @Override
    public String toString() {                 // Символно представяне
        return "Име: " + name + ", Възраст: " + age + ", Пол: " + gender;
    }
}
```

III. Създайте клас **UniversityMember** (**Член на университет**) като наследник на класа **Person** (**Личност**).

1. Декларирайте поле с **private** достъп за име на университет **university** от тип **String**.
2. Добавете конструктор с четири параметъра за инициализиране на полетата.
3. Напишете методи за достъп **getUniversity()** и **setUniversity()**, които връщат/установяват името на университета.
4. Предефинирайте метода **toString()** на класа **Person**, който връща символното представяне на типа.

```
package lab1;
public class UniversityMember extends Person {     // Клас Член на университет
    private String university;                      // име на университет
    // Конструктор с четири параметъра
    public UniversityMember(String name, int age, char gender,
                            String university) {
        super(name, age, gender);              // извиква конструктора на Person
        this.university = university;
    }
    public String getUniversity() {              // Метод за достъп
        return university;                       // връща името на университета
    }
    public void setUniversity(String university) {// Метод за достъп
        this.university = university;      // установява името на университета
    }
    @Override
    public String toString() {              // Символно представяне
        return super.toString() + "\n" +"Университет: " + university;
    }
}
```

IV. Създайте клас **Lecturer** (**Преподавател**) като наследник на класа **UniversityMember** (**Член на университет**).

1. Декларирайте поле с **private** достъп за име на дисциплина **subject** от тип **String**.
2. Добавете конструктор с пет параметъра за инициализиране на полетата.
3. Предефинирайте метода **toString()** на класа **UniversityMember**, който връща символното представяне на типа.

```
package lab1;
public class Lecturer extends UniversityMember {     // Клас Преподвател
    private String subject;                           // име на дисциплина
    // Конструктор с пет параметъра
    public Lecturer(String name, int age, char gender, String university,
                    String subject) {
        // извиква конструктора на UniversityMember
        super(name, age, gender, university);
        this.subject = subject;
    }
    @Override
    public String toString() {                        // Символно представяне
        return super.toString() + ", Дисциплина: " + subject;
    }
}
```

V. Създайте клас **Student** (**Студент**) като наследник на класа **UniversityMember** (**Член на университет**).

1. Декларирайте полета с **private** достъп за: курс **year** от тип **int** и успех **grade** от тип **float**.
2. Добавете конструктор с шест параметъра за инициализиране на полетата.
3. Напишете метод за достъп **getGrade()**, който връща успеха на студента.
4. Предефинирайте метода **toString()** на класа **UniversityMember**, който връща символното представяне на типа.

```
package lab1;
public class Student extends UniversityMember {        // Клас Студент
    private int year;                                   // курс
    private float grade;                                //успех
    // Конструктор с шест параметъра
    public Student(String name, int age, char gender, String university,
                   int year, float grade) {
        super(name, age, gender, university);
        this.year = year;
        this.grade = grade;
    }
    public float getGrade() {                  // Метод за достъп
        return grade;                          // връща успеха на студента
    }
    @Override
    public String toString() {                 // Символно представяне
        return super.toString() + ", Курс: " +year + ", Успех: " + grade;
    }
}
```

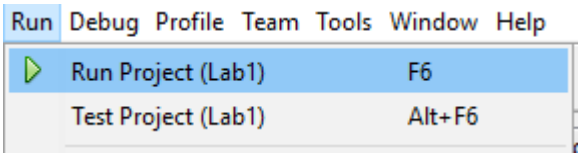
VI. Използвайте класа **Lab1** за тестване на класовете и тяхната йерархия.

1. В метода **main()** декларирайте: динамичен масив **list** от тип **ArrayList<Person>**, който ще съдържа преподаватели и студенти. Импортирайте класа **ArrayList** от пакета **java.util**. Декларирайте променлива **s** от тип **Student**, създайте инстанция и разпечатайте данните за студента.

```
package lab1;
import java.util.ArrayList;
public class Lab1 {                                          // Тестов клас
    public static void main(String[] args) {
        ArrayList<Person> list = new ArrayList<>();          // динамичен масив
        Student s =
            new Student ("Мария Георгиева", 21, 'Ж', "ТУ - София", 3, 4.57f);
        System.out.println(s);
    }
}
```

2. Изпълнете приложението:

Run -> Run Project (Lab1)



Резултатът е:

```
Име: Мария Георгиева, Възраст: 21, Пол: Ж
Университет: ТУ - София, Курс: 3, Успех: 4.57
```

Методът **println()** извиква метода **toString()** на класа **Student**, защото променливата **s** е от тип **Student** и по време на компилация се определят методите в зависимост от типа на обектите (**ранно свързване**).

3. Дефинирайте променлива **p** от супер класа **Person**, за да реализирате полиморфно обръщение към класовете **Lecturer** и **Student**. Създайте инстанции на класовете **Lecturer** и **Student**, като свържете референцията **p** на супер клас **Person** с всеки един създаден обект и добавете всеки обект към динамичния масив **list**.

```
Person p;
p = new Student ("Мария Георгиева", 21, 'Ж', "ТУ - София", 3, 4.57f);
list.add(p);
p = new Lecturer("Георги Стоянов", 45, 'М', "УНСС", "ООП");
list.add(p);
p = new Student("Иван Петров", 21, 'М', "ТУ - София", 3, 5.67f);
list.add(p);
p = new Student("Мартин Иванов", 22, 'М', "УНСС", 4, 5.34f);
list.add(p);
```

4. Към класа **Lab1** добавете статичен метод **printList()** с входен параметър **list** от тип **ArrayList<Person>** с данни за студенти и преподаватели за печат на динамичния масив с данни.

```
public class Lab1 {
    public static void printList(ArrayList<Person> list) {
    for (Person element :list)
        System.out.println(element);
    }
    public static void main(String[] args) {
        ...
    }
}
```

5. В метода **main()** извикайте статичния метод **printList()**, за да разпечатате динамичния масив.

```
System.out.println("Списък");
printList(list);
```

Тъй като **printList()** е статичен метод, за неговото извикване не е необходим обект за разлика от стандартното извикване на методите на инстанциите (**обект.метод()**).

6. Изпълнете приложението:
Резултатът е:

```
Списък
Име: Мария Георгиева, Възраст: 21, Пол: Ж
Университет: ТУ - София, Курс: 3, Успех: 4.57
Име: Георги Стоянов, Възраст: 45, Пол: М
Университет: УНСС, Дисциплина: ООП
Име: Иван Петров, Възраст: 21, Пол: М
Университет: ТУ - София, Курс: 3, Успех: 5.67
Име: Мартин Иванов, Възраст: 22, Пол: М
Университет: УНСС, Курс: 4, Успех: 5.34
```

По време на изпълнение се определя версията на извикания метод **toString()** в зависимост от типа на действителния обект (**късно свързване**).

7. Към класа **Lab1** добавете статичен метод **averageGrade()** с входни параметри: **list** от тип **ArrayList<Person>** с данни за студенти и преподаватели и университет **university** от тип **String**, който връща като резултат средноаритметичния успех от тип **float** на студентите от зададения университет, използвайки следния алгоритъм:

средноаритметично ← 0
брояч ← 0
за всеки елемент от тип **Person** от масива **list**
  ако елементът е инстанция от тип **Student**
    ако студентът е от зададения университет **university**
      средноаритметично ← средноаритметично + успеха на студента
      брояч ← брояч + 1
средноаритметично ← средноаритметично / брояч

```
public static float averageGrade(ArrayList<Person> list, String university) {
    float average = 0.0f;
    int count = 0;
    for (Person element : list) {
        if (element instanceof Student) {
            if (((Student) element).getUniversity().equals(university)) {
                average += ((Student) element).getGrade();
                count++;
            }
        }
    }
    if (0 != count)
        average /= count;
    return average;
}
```

Операторът **instanceof** връща **true**, ако обектът **element** е от тип **Student**. Тъй като **element** е дефиниран от тип **Person**, който не съдържа метода **getUniversity()**, необходимо е принудително преобрзуване на **element** в тип **Student**. Методът **equals()** на класа **String** връща **true**, ако двата низа съдържат еднакви символи.

8. В метода **main()** извикайте статичния метод **averageGrade()**, за да изчислите средноаритметичния успех на студентите от **ТУ – София**.

```
float average = averageGrade(list, "ТУ - София");
System.out.println("Средният успех на студентите от ТУ - София е " + average);
```

9. Изпълнете приложението.

Резултатът е:

```
Средният успех на студентите от ТУ - София е 5.12
```

VII. Допълнете класа **Lab1**:

1. Към класа **Lab1** добавете статичен метод **updateUniversity()** с входно/изходен параметър **list** от тип **ArrayList<Person>** с данни за студенти и преподаватели, и входни параметри: име на преподавател **lecturerName** от тип **String** и университет **university** от тип **String**, който обновява университета на преподавателя с даденото име, използвайки следния алгоритъм:

за всеки елемент от тип **Person** от масива **list**
  ако елементът е инстанция от тип **Lecturer**
    ако името на преподавателя е със зададеното име **lecturerName**
      установява зададения универстет **university** за преподавателя
      изход

2. Тествайте метода **updateUniversity()** и променете университета на даден преподавател.