Java classes, methods, access specifiers. Classes for arrays of characters.

Objectives:
- implementing Java classes and methods
- getting familiar with Java access specifiers
- implementing Java applications that work with arrays of variables and *String*, *StringBuffer*, *StringBuilder*, *StringTokenizer* data

Java classes

A Java class is declared using the keyword *class* followed by the class name (*CamelCase* notation).
A class can contain variables/attributes and implemented methods.
Both the class and its members can be marked using the access/visibility specifiers.
Syntax:

```
[access_specifiers] class ClassName{
        //class implementation
}
```

Class access specifiers:

- *default* (if not specified)
- *public*
- *abstract*
- *final*

The inner classes may have other access specifics.

Class members access specifiers:

- *default* (if not specified)
- *public*
- *abstract*
- *final*
- *private*
- *protected*
- *static*

Note:
If a class is declared public, the file in which it is stored must bear the same name as the class name.
If the class is not public, there is no restriction whatsoever with caution to the source file name.

Access/visibility specifiers

The Java programming language provides the following access/visibility specifics, which can be applied (as appropriate) to classes, methods or variables.
`final`: may be applied to classes, methods and variables.
`abstract`: may be applied to classes and methods.
`static`: can be applied to variables, methods and blocks of code. A static member is associated with the class to which it belongs and not with a specific instance. A static method cannot access other non-static members of the class to which it belongs. Static methods do not have access to *this* default variable.
`native`: can only be applied to methods.

`transient`: can only be applied to variables
`synchronized`: is used to control the access of multiple threads to common resources.
`volatile`: can be applied to variables
`public`: may be applied to classes, variables and methods. A public class is visible from anywhere outside (where the package to which it belongs is visible (if the class is associated with a package)). A public member (variable or method) can be accessed from anywhere outside the class through the created instances.
`protected`: may be applied to variables or methods. Members marked in this way can be accessed from anywhere in the same package as the class to which they belong and in any of the inheriting subclasses (regardless of the package to which they belong). Protected members cannot be accessed directly from outside through instances.
`default`: represents the default access specifier that is applied to classes, methods, or variables that did not mention explicitly any of the public, private, or protected specifiers. Default members can be accessed from anywhere within the same package.
`private`: can be applied to variables or methods of a class. It is the most restrictive visibility specifier. Private members can only be accessed from within the class to which they belong.

Java methods (functions)

A Java method is a software entity that groups a sequence of statements of variables, instructions, and/or program blocks and may or may not return a variable of a certain type.
Syntax:

```
[access_specifiers] returnedType methodName (parameter_list){
      //method implementation
}
```

The returned type can be *void* if the method does not return anything or any other data type (elementary or not), string/array type or not.
The method name is an alpha-numeric sequence of characters (*camelCase* notation).
The parameter list is a sequence of variable types and names, separated by a commas, that the function receives at calling time.

Special methods:

- **setter methods:** public methods that receive as parameters values to be associated with the corresponding attributes in the class;

  public void setVariableName(variable_type variable_name);

It is recommended that each attribute has its own setter.

- **getter methods**: public methods that return the values of the class attributes;

  public variable_type getVariableName();

It is recommended that each attribute has its own getter.

Constructor/destructor methods

There is a special category of methods that are members of a class, namely ***constructor methods***. These are analogous to those in C++.
In Java there are no destructors. However, there is a complementary mechanism that allows a sequence of code to be executed when an object is destroyed. This mechanism is based on the method

```
public void finalize()
```

being implicitly inherited from the *Object* class. Overriding this method takes the place of the destructor and is called by:

System.gc(); correlated with *System.runFinalization();*

The *main( )* method

Possible prototypes:

```
public static void main(String[] args){
           //implementation
           }
```

OR

```
public static void main(String args[]){
           //implementation
           }
```

OR

```
public static void main(String... args){
            //implementation
           }
```

Static methods

A static method cannot access other non-static members of the class to which it belongs

Static methods do not have access to the default variable *this*.

Inner classes

Java allows defining some classes within other classes. Such classes are called inner classes.

An inner class is a member of the class in which it was declared.

Syntax:

```
      class ClasaExterioara{
           //...
           [static] [access_specifier] class ClasaInterioara{
                //...
           }
      }
```

As a member of the class to which it belongs, an inner class has access to all members of the class that encapsulates it, regardless of the access specifier (*public*, *protected*, *private*).

An inner class can be marked with any of the visibility specifiers *public*, *protected* or *private*.

An instance of an inner class is always associated with an instance of the encapsulating class and has direct access to its members.

Syntax:

```
      ClasaExterioara obiect_exterior = new ClasaExterioara();
      ClasaExeterioara.ClasaInterioara   obiect_interior   =   obiect_exterior.new
ClasaInterioara();
```

The inner classes can also be marked with the *static* access specifier. A static inner class has access only to static members of the exterior class. A static inner class is associated with the exterior class itself and calling its methods looks like:

Syntax:

```
      ClasaExterioara.ClasaInterioara.metoda();
```

Instantiating a static inner class uses a syntax similar to the one below:

Syntax:

```
      ClasaExterioara.ClasaInterioara obiect_interior = new
ClasaExterioara.ClasaInterioara();
```

Abstract classes

A Java class must be marked with the *abstract* keyword if it has at least one abstract method (without implementation, only the declaration of the method appears in the class body).

Syntax:
```
      abstract class NumeClasa{
          public abstract void metodaAbstracta();
          public void metodaNormala(){
               //implementarea metodei
                    }
          }
```

Abstract classes cannot be instantiated.
Abstract classes can only be inherited.

Class `String`
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

=> all Java variables that contain a sequence of characters can be processed using this class.

Class `StringBuffer`
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StringBuffer.html

! Unlike a *String* variable, a *StringBuffer* object can be modified during its existence.

Class `StringBuilder`
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StringBuilder.html

=> it is intended to be a replacement for the *StringBuffer* class. The mechanisms, functionalities and methods it offers are very similar to those of the aforementioned class.

Clasa `StringTokenizer`
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/StringTokenizer.html

=> allows the separation of a *String* object into subsections (portions) based on certain separators.

All string classes offer support for regular expressions (regex, http://www.regexr.com , https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/regex/Pattern.html)

Examples

Ex. 1: Java methods

```
class Methods_01 {
      void m1(int... sir){
            for(int crt_val : sir){
                  System.out.print(crt_val+" ");
            }
            System.out.println();
      }
      void m2(int sir[]){
            for(int i=0; i<sir.length; i++){
                  System.out.print(sir[i]+" ");
            }
            System.out.println();
      }
      void m3(int sir[]){
            sir[0] = -777;
      }
      int[] m4(){
            int sir[] = {8, 88, 888};
            return sir;
      }
      /*
      void m5(int x=7){ // nu exista functii cu parametri impliciti
            System.out.println(x);
      }
      */
      static void m6(){
            System.out.println("Metoda statica recomandat a fi apelata fara
obiect");
      }
}

public class Test{
      public static void main(String... args) {
            int sir[] = {5, 3, 777};
            Methods_01 ob1 = new Methods_01();

            ob1.m1(sir); //parametru de tip tablou
            ob1.m1(10); //o singura variabila
            ob1.m1(100, 101, 102); //variabile enumerate
            ob1.m1(); //nici un parametru
            ob1.m2(sir);
            ob1.m3(sir);
            ob1.m2(sir); //-777, 3, 777
            sir = ob1.m4(); //8, 88, 888
            ob1.m1(sir);
            Methods_01.m6();  // apel metoda statica
      }
}
```

Ex. 2: inner classes

```
class InnerClass_01 {
	static int i1 = 1;
	int i2 = 2;
	void m1(){
		int i3 = 3;
		class Inner1{
			public void m1Inner(){
				System.out.println(i1);
				System.out.println(i2);
				System.out.println(i3);
			}
		}
		Inner1 ob1 = new Inner1();
		ob1.m1Inner();
	}
	class Inner2{
		public void m2Inner(){
			System.out.println(i1);
			System.out.println(i2);
		}
	}
}
public class Test{
	public static void main(String... args) {
		InnerClass_01 ob1 = new InnerClass_01();
		ob1.m1();
		//InnerClass_01.Inner1 ob2 = new InnerClass_01().new Inner1();
		// clasa Inner1 invizibila in ext, fiind interioara metodei m1()
		//ob2.m1Inner();
		InnerClass_01.Inner2 ob3 = new InnerClass_01().new Inner2();
		ob3.m2Inner();
	}
}
```

Ex. 3: StringTokenizer

```
import java.util.StringTokenizer;
class StringTokenizer_01 {
     public static void main(String[] args) {
          String tags = "pizza pepperoni food cheese";
          // convert each tag to a token
          StringTokenizer st = new StringTokenizer(tags," ");
          while ( st.hasMoreTokens() ){
            String token = (String)st.nextToken();
            System.out.println(token);
          }
     }
}
```

Ex. 4: StringBuffer

```
import java.lang.*;
public class Test{
     public static void main(String[] args) {
          System.out.println("Test StringBuffer!");
          StringBuffer sb = new StringBuffer(0);
          //inserarea la prima pozitie
          System.out.println(sb.insert(0, "început"));
          //determinarea lungimii
          int len = sb.length();
          //inserarea la sfarsit
          System.out.println(sb.insert(len, "sfarsit"));
          //inserarea la o pozitie intermediara a unei instante anonime Integer
          System.out.println(sb.insert(7, String.valueOf(777)));
          //inserarea la sfarsit a unei variabile de tipul elementar int
          System.out.println(sb.append(888));
     }
}
```

Individual work

Complexity *

1. Write a class named Methods having 3 methods: a method with a variable number of integer parameters, which returns the arithmetic mean of the parameters; another method in which will be defined a one-dimensional array of n double-type numbers (n being the parameter of the method and which will be read in main () from the keyboard), its elements will receive randomly generated values and the method will return this array; a static method that receives as a parameter a character array, converts the respective characters into uppercase letters and forms a String object from this array, which wiil be returned by the function. From the main () method, defined in another class, call the methods of the Methods class and display the results on the screen (the call values for the first method and the character string for the 3rd method will be defined in main ()).

2. Define a class *X* that will have the following attributes: a private array of characters, a protected String object and an integer without access specifier. The class will contain an explicit constructor, which will initialize the attributes, accessor and mutator methods for fields that cannot be accessed outside the class and redefine the finalize () method inherited from the Object class, which will display an appropriate message.
   In the main() method, placed in another class, create an object of X class, display the attribute values, and then prepare the object to be removed by garbage collector. Then create a new X object, change the attribute values, and display their attribute values.

3. Write a Java class that models a matrix of integer values. The dimensions and the array of elements are private attributes and are controlled using appropriate setter-getter methods. Write the methods for displaying the matrix, for determining and returning the number of 9 adjacent elements cells that don't differ with more than 5% from a threshold value.

4. Develop a Java program which defines a class named *Person* that contains as private attributes: *name* (*String*), *latitude* and *longitude* (*float*). The class contains an explicit constructor without parameters that will initialize the latitude and longitude with *0* and the string with *null*. A *finalize()* method will be used as a destructor from C++ and will display a message. The mutator methods, *setName()*, *setLongitude()* and *setLatitude()* will be defined to modify the attributes of the class. Define appropriate accesor methods. Instantiate *n* objects, read the corresponding data from the keyboard and display the information related to all the objects.

5. Build a Java application that tests the new methods of the String class, introduced in Java11.

6. Define a one-dimensional array of *char* type, less than a value *n* introduced from the keyboard. Each value contains some randomly generated alpha-numerical characters. Generate a *String* object using the array. Display the initial content of the array and after that, process all the elements so that all numbers will be replaced with '*'. Display the result.

Complexity **

7. Write an application which defines a class named *Child*. Define the methods and member variables for this class which enble a Child object to store/do the following:

- the name of the child
- the child’s birthday
- the child can introduce him/herself by „saying”: Hello my name is ...
- the child can tell his/her age
- the child can add two numbers smaller than 10 and return the result like so: The sum of X and Y is equal to Z
- the child knows how to say Goodbye!
- the child can speak the alphabet both in direct and inverse order
- the child can color a chess board given its dimensions by using alternative colors (for the colors use the symbols 1 and 0)
- the child can play dots and crosses (X-0) by him/herself ☺ (use the application developed in the previous homework)

Remarks: the child’s name and birthday cannot be accessed from outside the class.
All the information about a *Child* will be filled-in using a Child object and its associated methods and variables.
The interaction with teh child will be done through an object which is instantiated in the main method.

8. Write a Java application which defines an authentication key with the format: XXXXX-XXXXX-XXXXX-XXXXX, where X is a character which can be either a digit or a letter. The application should verify if this key has exactly 4 groups of characters with 5 characters each, and separated by the symbol ‘-‘. Also, compute the number of digits and letters from the authentication key. The number of digits should be greater than the number of letters, and the number of letters cannot be 0.
   If any of the above conditions are not met, display the message: “Invalid authentication key!”

9. Define a class named *Student* that has as private attributes the *name*, *tel. number* and *average mark* (*constructors*, *setters, getters*).
   In the *main* method create an uni-dimensional array of *Student* objects with the dimension specified by the user. The data corresponding to each object is read from the keyboard, respecting the format *name^^^tel. number^^^average mark*. If the data specific to the telephone number is not valid (10 numeric characters, with or without special characters like blank, - or .), the user is asked to re-enter the entire array of characters.
   Display the students ordered by name and by average mark.

10. mplement a class named *Circle* with the private attributes *color* (*int*), *radius* (*float*), *planar position* (2 *integer*).
    Define the specific constructors and setter getter methods. The color is divided in 4 bytes, each of them representing the transparency and the quantities of R, G and B.
    In the *main* method (included in another class), read from the keyboard the data for *n Circle* objects.
    Implement the methods that receive as parameter the array of objects and display:
    - the circles whose center is included in one of the 4 quadrants
    - the circles that are included entirely in one of the 4 quadrants
    - the circles that have the centers on the same horizontal or vertical line
    - the circles that have the R, G or B quantities in a certain specified interval