Simple Java stand alone applications: Operators. Instructions. Variables. Arrays of variables.

Objectives:

- implementing simple Java applications
- implementing Java applications that work with operators, instructions, variables, arrays of variables

Java elementary types

For the development of simple stand-alone applications, the Java programming language defines the following simple (elementary) types of data: `byte, short, int, long, char, float, double` and `boolean`.

Java wrapper classes

Java provides the programmer with the so-called ***wrapper classes***. Each type of elementary data is associated with an encapsulating class with the same name as that of the type it represents.

The wrapper classes are located in the package `java.lang`.

The wrapper classes and the associated elementary types are mentioned below:

- The `Void` class is associated with the `void` type
- The `Byte` class is associated with the byte elementary type.
- The `Short` class is associated with the elementary `short` type.
- The `Integer` class is associated with the elementary `type int`.
- The `Long` class is associated with the elementary type `long`.
- The `Float` class is associated with the elementary `float` type.
- The `Double` class is associated with the `double` elementary type .
- The `Boolean` class is associated with the elementary `boolean` type.

An instance of an encapsulating class is a non-modifiable object and stores only one equivalent primitive value.

All classes mentioned above are marked with `final` and `public` access specifiers; they provide useful data processing methods.

Under certain circumstances the Java environment hides the use of wrapper classes by using ***autoboxing*** and ***unboxing*** mechanisms.

Java operators

The Java programming language has operators very similar to those in the C/C++ language, but there are a number of differences that distinguish them.

As differences we have among others:

`>>>` unsigned right shifting

Java variables

The Java programming language defines the following types of variables.

- **elementary variables**: are variables that have as type an elementary data type
- **instance variables**: are object or instance type variables of a Java class
- **class-type variables**: are variables declared with the *static* keyword and belong to the class in which they were declared and not to a particular instance
- **local variables**: are the variables declared within a code sequence (class, interface, method or any block of data delimited by *{...}*)
- **parameter variables**: are variables that are sent as parameters to methods

Java instructions

As with other programming languages, programs written in Java contain instructions that are interpreted by the compiler by parsing the source files from top to bottom.

The Java instructions are of the following types:

- **conditional instructions**(`if-else, switch`)
- **repetitive instructions**(`do-while, while, for`)
- **instructions for directing the execution of the program** (`break, continue, return`)

The instructions are similar to the C/C++ language with some differences given by newly added facilities such as: `break` and `continue` with labels, the generalized for loop, etc.

Arrays of variables (one-dimensional, multidimensional)

Elements of the same type (primitive data or class instances), accessed by index.

Declaration: *tip_date nume_variabila[ ]; or*
tip_date [ ] nume_variabila;
Memory allocation: *nume_variabila = new tip_date[nr_elemente];*

.*length* property => the number of elements allocated in the array

Class `Arrays`

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Arrays.html
=> operations on array variables (searches, sorting, copying, etc.)

Class `String`

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.htmlâ
=> allows maneuvering character strings; all Java literal variables that contain a sequence of characters can be processed through this class.

Examples:

```
//declaring and initializing a String variable
String sir = new String("abc");
```

! *String* variables cannot be changed once they have been initialized.

An instance of the *String* class can be converted into an array of characters in *Unicode* representation, using the `toCharArray()` method, without the character '\0' at the end (as opposed to C/C++)

Java provides a mechanism that allows the *+* operator (with higher priority than the relational operators) to be used for the concatenation of String type variables. It is also allowed to concatenate *String* variables with variables of other types, including elementary ones (*byte*, *short*, *int*, etc.). This is possible due to the fact that any Java class is considered as being derived directly or indirectly from the *Object* class. This class can transform the contained data (*the toString* method) into *String* variables.

Example:

```
String s1 = new String("abc");
String s2 = s1 + "def";       //the array "def" is automatically converted to String
String s3 = s1 + 5;           //the initial string is concatenated with the string "5"
System.out.println(s1 + s2 + 100.23f);
//etc.
```

Examples:

Bit operators:

```
public class OpLogiciBiti {
  public static void main(String args[])  {
    int a= 27;    // 0x001B;
    int masca= 15; // 0x000f;
   System.out.println(" ~a= "+ Integer.toBinaryString(~a)+"=
          0x"+Integer.toHexString(~a));
   System.out.println(a+" & "+masca +"= "+ (a&masca)+" =
          "+Integer.toBinaryString(a&masca));
   System.out.println(a+" | "+masca +"= "+ (a|masca)+" =
          "+Integer.toBinaryString(a|masca));
   System.out.println(a+" ^ "+masca +"= "+ (a^masca)+" =
          "+Integer.toBinaryString(a^masca));
  }
}
```

```
<terminated> OpLogiciBiti [Java Application] C:\Program Files\Ad
 ~a= 11111111111111111111111111100100= 0xffffffe4
27 & 15= 11 = 1011
27 | 15= 31 = 11111
27 ^ 15= 20 = 10100
```

for( )* standard and *for_loop( )

```
import java.util.Scanner;

public class Test {
 static Scanner in=new Scanner(System.in);
        public static void main(String[] args) {
                int nota, nr;
                System.out.println("Introdu nr. note : ");
                nr=in.nextInt();
                int note[ ]=new int[nr];
                System.out.println("Introdu notele : ");
                for(int i=0;i<note.length;i++) {
                        nota=in.nextInt();
                        if(nota<=10 && nota >=1)note[i]=nota;
```

```
        else { System.out.println("Nota incorecta. Reia: "); i--;continue;}
                }
        System.out.println("Notele introduse sunt: ");
                for(int it: note)
                System.out.print(" " + it);
        }
}
```

***break* and *continue* with labels**

```
public class LabelBreakContinue {
        public static void main(String[] args) {
int tabi[ ][ ] = {{1, 5, 3}, {11}, {7, 11, 9, 13}, {8,11}};
System.out.println("Valorile din matrice pana la prima divizibila cu 2:");
        et0: for(int i=0; i<tabi.length; i++){
                System.out.print("\nlinia " + (i+1) + ": ");
                for(int j=0; j<tabi[i].length; j++){
                        if(tabi[i][j] % 2 == 0)
                                continue et0;
                        System.out.print(tabi[i][j] + " ");
                }
        }

int mati[ ][ ]= {{1, 2, 13}, {5}, {8, 19, 6}};
        int i=0, j=0;
        boolean gasit=false;
        et1: for(i=0; i<mati.length; i++)
        for(j=0; j<mati[i].length; j++) {
                        if(mati[i][j]%3==0) {
                                gasit=true;
                                break et1;
                        }
                }
        if(gasit)
                System.out.println("\nPrima Valoarea  modulo 3 e: " + mati[i][j]+" pe pozitia, linia: "+(i+1)+",
coloana: "+(j+1));
        else
                System.out.println("\nNici un numar div. cu 3");
        } //main
}
```

Class *Arrays*

```
import java.util.Arrays;
public class JavaArraysExample {
   public static void main(String[ ] args) {
      // Base array for the example. It contains 9 elements.
      int[] baseArray = { 2, 4, 3, 7, 21, 9, 98, 76, 74 };
         // Sorts the specified range of the array into ascending order.
          System.out.printf("\nUnsorted baseArray:");
          for (int it: baseArray) System.out.printf(" %d ", it);
          System.out.println("\nUnsorted baseArray as string: "+ Arrays.toString(baseArray));
      Arrays.sort(baseArray, 0, baseArray.length);//fromIndex toIndex
      System.out.printf("\nSorted baseArray: %s\n\n", Arrays.toString(baseArray));
      // Assigns the specified int value to each element of the array
      int[] fillArray = new int[10];
       System.out.printf("fillArray (implicit): %s\n", Arrays.toString(fillArray));
       Arrays.fill(fillArray, 0, fillArray.length, 3);//array, fromIndex, toIndex, Value
       System.out.println("fillArray (after): "+ Arrays.toString(fillArray));

       String[ ] namesArray = new String[10];
       System.out.println("names Array (implicit): "+ Arrays.toString(namesArray));
       // Assigns the specified String value to each element of the array
       Arrays.fill(namesArray, 0, namesArray.length, "John");//array, fromIndex, toIndex, Value
       System.out.println("namesArray (after): "+ Arrays.toString(namesArray));
   }
}
```

Methods with variable number of parameters

```
public class Methods_var{

    void vaTest(int... v){

            System.out.print("Nr. arg= "+ v.length + "\nContinut: ");

            for(int x:v)

                    System.out.print(x+" ");

            System.out.println("\n");

    } //vaTest



    public static void main(String args[]){

            Methods_var ob1 = new Methods_var();

            ob1.vaTest(10);       // 1 arg

            ob1.vaTest(1,2,3);     // 3 arg
```

```
        ob1.vaTest();          // fara arg
        int tab[]= {100, 200, 300, 400, 500};
        ob1.vaTest(tab);
    }//main
}//class
```

Individual work

Complexity *

1. Write a Java application that reads an int value. If the int value is between 1-12, the corresponding string month will be displayed. If the value entered is a string and if it corresponds to a month of the year, display the numeric value of the month.

2. Read a string from the standard input. Turn the string into a character array. Look for in this array a character specified in the program. Display the number of occurrences.

3. Starting from the previous problem, copy the first 3 characters of the array to another array and display the obtained result(use the arraycopy () method from the System class).

4. Write a Java application which defines an integer value and displays it as a binary, octal and hexadecimal string. Write various bases convertion methods.

5. Implement the already known sorting algorithms (bubble sort, insertion sort, quick sort, etc.) and apply them upon an array of integer variables read from the keyboard.

6. Write a Java program that generates 2 random values and performs some mathematical operations on them.

Complexity **

7. Read from the keyboard an integer value bigger than 16.777.216. Use bit masks for isolating each of the 4 bytes of the read value. Display the initial and the isolated values as decimal, binary and hexadecimal strings.

8. Read from the keyboard the elements of a matrix of integer values with m lines (m taken from the command line). For each line the number of elements will increase by 1 compared to the previous line, the first line having only one element.

Implement the methods that:
- display the matrix, line by line and column by column;
- eliminate from the matrix (turns into 0) the values that are outside the interval defined by 2 specified limits;
- display the existent neighbour values of an element identified by its indexes (sent as parameters);

9. Define an array of String variables that will be populated with all the playing cards from a complete package. A series of randomly picked cards will be extracted until the current card will have the clubs sign and a value greater than 8. At each step, the current card and the number of already extracted cards will be displayed.

Hint: Use a random numbers generator. The cards which were already used are eliminated.

Complexity *

10. Assume that there is a cryptographic algorithm which takes an input text ‘A’ composed of lower and upper case characters. Separately a character string ‘B’ is defined. Each character from B has an associated random integer value between 1 and 100. The algorithm checks if the letters from B are found in A and adds the associated numerical values. To the final sum value, the algorithm also adds the positions from string A where characters

from string B were found. If the final sum is larger than 100, the encryption was valid. Display a message with the result.
Example:
String A = ”aTmPpDsst”
String B =”ams”
Associated numerical values for string B: 11 33 7
Sum: (11+33+7+7)+(1+3+7+8)=77 -> INVALID ENCRYPTION

11. Implement the naive dots and crosses game (X-O) as an automated game. The application will randomly select at each step a position from the board at which to place the next symbol, alternating between X and O. The selected position cannot be an already filled square. The game ends when either there are no more empty squares on the board, or one of the symbols wins by completing a line, a column or a diagonal. Display on the screen each step of the algorithm as a matrix. Unfilled squares will be represented by the * character.
Example:

```
 X        *        *
 *        *        *
 *        *        *
--
 X        *        *
 *        0        *
 *        *        *
--
 X        *        *
 *        0        *
 X        *        *
--
 X        *        0
 *        0        *
 X        *        *

 X        *        0
 X        0        *
 X        *        *
```

Game over!
---
*Extend the application so that the dimensions of the board (number of rows and columns) can be changed.