Mediul Eclipse. Aplicații de bază Java

The Eclipse environment. Basic Java applications

Obiective

- Înţelegerea teoretică şi practică a elementelor de baza privind programarea in mediul Java.
- Scrierea de programe simple, după modelul programării obiectuale, folosind mediul integrat Eclipse
- Executia aplicatiilor din prompter de comanda
- Aplicatii simple cu operatii de intrare/iesire si aplett-uri minimale in Java.

Limbajul Java a apărut (1996, lansarea oficială a primei versiuni a limbajului Java) ca alternativa a mediului software C++ şi s-a dorit preluarea tuturor aspectelor pozitive manifestate de acesta şi eventual, eliminarea celor negative. Putem spune că limbajul Java l-a avut drept părinte pe C/C++, sintaxele Java şi C/C++ fiind foarte apropiate. Se poate considera, de asemenea, că pentru un programator familiarizat cu limbajul C/C++, acomodarea cu mediul Java nu ar trebui să prezinte nici un fel de problemă. Limbajul Java înglobează, pe lângă moştenirea esenţială primită de la C++, o multitudine de aspecte preluate dintr-o varietate de alte terţe limbaje de programare, dintre care amintim Eiffel, SmallTalk, Objective C, Cedar/Mesa, etc.

Platforma Java:

- este o platformă pur software;
- rulează pe un nivel imediat superior (se bazează pe platformele menţionate anterior);

Platforma Java conţine 3 componente:

- Maşina virtuala Java (*Java Virtual Machine*, JVM). In JVM este JIT (Just In Time) compiler folosit pentru a compila cod Java si a oferi facilitati de cache-ing in vederea imbunatatirii performantelor JVM
- Interfaţa de programare a aplicaţiilor (*Application Programming Interface,* API)
- Mediul de rulare Java (*Java Runtime Environment, JRE*)

Tipuri de aplicații Java

Aplicaţiile Java pot fi clasificate după cum urmează:

- ***aplicaţii de sine stătătoare*** (*stand-alone applications*). Caracteristica lor esenţiala este că, conţin metoda *public static void main(String[] args*) care constituie punctul de intrare în aplicaţie. Pot avea funcţionalităţi diverse şi pot rula din linia de comandă sau pot avea interfeţe grafice de interacţiune cu utilizatorul.
- ***applet-uri.*** Sunt aplicaţiile destinate rulării într-un browser de Web. Datorită dinamicii aplicațiilor web aplicațiile bazate pe applet-uri nu mai sunt încurajate a fi folosite ele fiind *eliminate din Java 11* putând fi executate pana la *Java 8*. În faza de testare pot fi executate şi cu utilitarul *appletviewer.exe*. Nu au metoda *public static void main(String[] args)* dar au alte funcţii caracteristice, printre care *init(), start(), stop(), destroy()*, etc. care pot fi suprascrise de programator pentru a defini comportamentul specific a fiecărei aplicaţii de acest tip. Applet-urile trebuie să fie însoţite de minim un fişier **.html* care să conţină liniile specifice de încărcare şi vizualizare a applet-ului.
- ***applet-aplicaţie*** *(appletcation)* sunt un tip de aplicaţie hibrid între primele două menţionate mai sus, care sunt de asemenea *eliminate* incepand cu *Java 11*. Conţin metodele specifice applet-urilor, dar au şi metoda public static void main(String[] args). Pot fi lansate în execuţie atât ca aplicaţii de sine stătătoare cât şi că applet-uri, modalitatea concretă de funcţionare fiind decisă în momentul rulării.
- ***servlet***-urile. Un servlet se aseamănă cu un applet, prin faptul că rulează în interiorul unei alte aplicaţii care

suportă mediul Java, şi respectă regulile impuse de aceasta. În cazul servlet-ului aplicaţia este un server de Web. Servlet-urile Java sunt folosite pentru construirea de aplicaţii Web interactive, fiind un înlocuitor al script-urilor CGI. Aplicatiile web au facilitati de *frontend* si de tip *backend* care urmaresc gestiunea interactiunii cu utilizatorul si cu bazele de date. Ele sunt acum dezvoltate folosind diferite framewok-uri dedicate mult mai performante bazate pe Java Script si alte medii adecvate, cum ar fi Angular, React, Ruby, Vue, Svelte, MySql, etc.)

- ***etc.***

JDK – Java Developers's Kit

Conține toate instrumentele necesare pentru crearea, depanarea și execuția programelor

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Evoluția: JDK 1.0 – 1.1.7, J2SE (JDK 1.2 – 1.4), generice in J2SE 5, Java SE6, Java SE7, Java SE8 (LTS), Java SE9, Java SE10, Java SE11 (LTS) …

Documentatia referitoare la ultima versiune Java SE17:

https://docs.oracle.com/en/java/javase/17/docs/api/index.html

Fișierele sursa: *.java

Fișierele bytecode: *.class

JDK - Structura de subdirectoare

bin => director de stocare a utilitarelor şi executabilelor Java. Aici rezidă fişierele *appletviewer.exe, jar.exe, java.exe, javac.exe, javadoc.exe, javah.exe, javap.exe, jdb.exe, serialver.exe, servertool.exe,* etc.

conf => director ce stochează fișiere care conțin opțiuni configurabile de utilizator

- modificarea permisiunilor de acces ale JDK

- configurarea algoritmilor de securitate

- setarea fișierelor Java Cryptography Extension Policy

include => director de stocare a unor fişiere de tip header C/C++, **.h*;

jmods => stochează module compilate, utilizate de jlink pentru a crea cod *runtime* personalizat.

legal => director de păstrare a fișierelor de licență și drepturi de autor pentru fiecare modul

- include notificările terților ca fișiere .md (marcaj).

lib => director de stocare a bibliotecilor Java (fişiere **.jar*, **.sym*, **.lib*, **.idl*).

Utilitare Java

Utilitarele Java, după cum am menţionat anterior, se află stocate în directorul ***bin*** aflat în locaţia de instalare a JDK. Pe versiunile JDK destinate sistemelor Microsoft, utilitarele Java sunt de regulă fişiere executabile **.exe*. Oricare dintre ele poate fi rulat din linia de comandă. Orice utilitar Java acceptă o serie de opţiuni asupra cărora utilizatorul este informat (de regulă) prin scrierea directă a numelui executabilului dorit direct în linia de comandă.

javac.exe = compilator

javac [opţiuni] [fişiere_sursa] [parametri_din_linia_de_comanda]

Exemplu: fișierul sursă Test.java

Variante de compilare:

javac Test.java

javac –nowarn Test.java invalidează mesajele de warning

*javac *.java* compilează toate fișierele cu extensia *java* din director

java.exe = interpretor

java [opţiuni]nume_clasa [lista_de_argumente]

java [opţiuni] -jar nume_arh.jar [lista_de_argumente]

Exemplu: după compilare → *Test.class*

rulare: *java Test*

Generare documentație /**...*/ ***javadoc*** *nume_fisier.**java***

Dezasamblare fișiere bytecode (.class): ***javap*** *nume_fisier.**class***

Depanator java: ***jdb*** *nume_de_ clasa*

Java IDE

IDE = Integrated Development Environment

- *Eclipse,* https://www.eclipse.org/
- ApacheNetBeans, https://netbeans.apache.org/download/index.html
- IntelliJ IDEA, https://www.jetbrains.com/idea/download/
- etc.

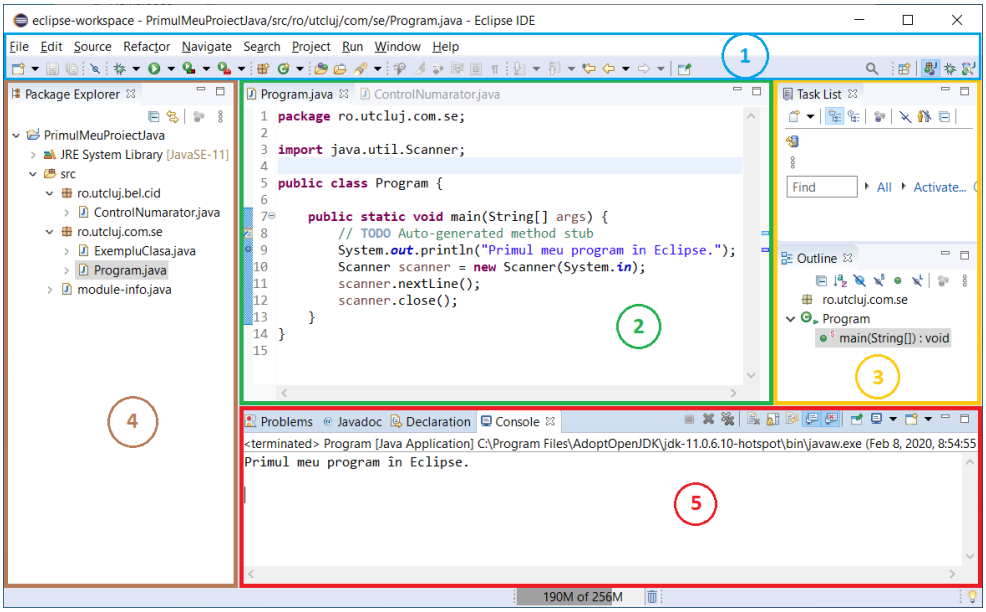
Mediul de dezvoltare integrată Eclipse

Perspectivele de lucru sunt moduri de afișare pentru diferite sarcini pe care un programator le poate face. Există 2 perspective de lucru frecvent utilizate:

1. Perspectiva Java, folosită pentru scrierea de cod și navigarea ierarhiei de fișiere din proiect, a ierarhiei de clase și pentru vizualizarea sarcinilor de lucru. Aceasta este perspectiva implicită în Eclipse pentru proiectele Java.
2. Perspectiva Java Debug, folosită pentru depanarea programelor Java și pentru execuția pas cu pas a acestora. Se deschide automat în momentul în care un program este pornit în mod depanare, și constă în afișarea unor ferestre pentru vizualizarea firelor de execuție, a variabilelor, a punctelor de oprire pentru depanare și a consolei. Este foarte important de reținut că în mediul de dezvoltare Eclipse consola este văzută sub forma unei ferestre din mediul de dezvoltare.

Utilizatorul poate schimba manual perspectiva de lucru din meniul Window și submeniul Perspective, dar de cele mai multe ori acest lucru nu e necesar.

În figura următoare este prezentată perspectiva de lucru Java, cu zonele principale etichetate în vederea explicării.



Zona 1 reprezintă bara de meniuri, localizată în jumătatea de sus, și bara de butoane. Bara de butoane are rolul de a aduce mai la îndemână funcțiile din meniu care sunt folosite cel mai frecvent. Partea din stânga a barei de butoane facilitează accesul la crearea de noi proiecte, clase, interfețe sau alte entități, salvarea documentelor, și este urmată de o zonă specială pentru lansarea programelor în depanare sau în execuție. În partea dreaptă avem acces facil pentru schimbarea perspectivei de lucru.

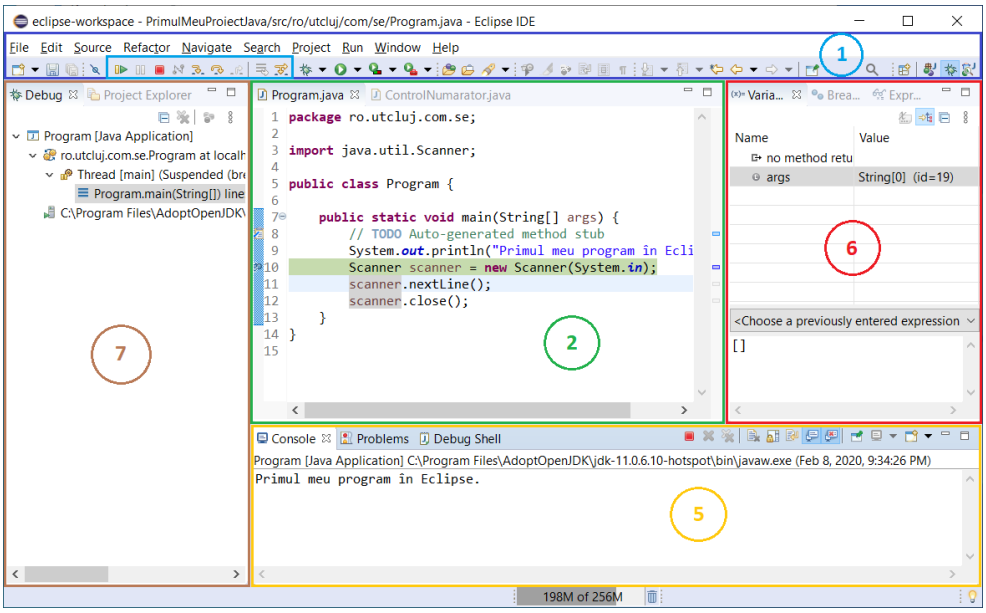
Zona 2 reprezintă editorul de programe.

Zona 3 reprezintă fereastra Outline și fereastra cu lista de sarcini. Fereastra Outline afișează elementele de program disponibile în editorul de programe.

Zona 4 reprezintă exploratorul de proiecte și de pachete. Aici se poate explora conținutul spațiului de lucru și al fiecărui proiect, respectiv pachet în parte.

Zona 5 conține ferestre pentru afișarea potențialelor probleme din program, a documentației generate cu javadoc, a liniei pe care o variabilă selectată în editorul de program a fost declarată și a consolei. Această zonă vine însoțită de o bară de butoane, care permite utilizatorului să facă operații cu consola, ștergerea textului, selectarea modului de afișare a textului, selectarea consolei în cazul în care mai multe programe au fost lansate în execuție simultan.

În continuare puteți observa perspectiva de lucru Java Debug, împreună cu blocurile principale etichetate pentru explicații.



Zona 1 este în mare parte la fel ca în perspectiva Java, cu o modificare notabilă, butoanele pentru crearea de elemente noi de cod au fost înlocuite cu butoane pentru control execuției pas cu pas a programului. Execuția pas cu pas este utilă deoarece ne permite să vizualizăm într-un anumit moment valorile pe care diferitele variabile utilizate le au și alte detalii relevante.

Zona 2 este aceeași ca în perspectiva Java, dar în această figură se observă linia 10 ca fiind marcată printr-un punct de oprire (eng. breakpoint) în bara din stânga liniei și evidențiată cu verde, acest lucru însemnând că execuția programului a fost oprită la acea linie pentru inspecție.

Zona 5 oferă accesul la fereastra consolă, fereastra cu lista de potențiale probleme și la fereastra pentru execuția de cod ad-hoc în contextul programului.

Zona 6 afișează o listă cu variabilele vizibile la linia la care a fost oprit programul pentru inspecție. Tot în zona 6 avem acum deschise ferestre pentru gestiunea punctelor de oprire pentru inspecție și pentru evaluarea de expresii în contextul programului.

Zona 7 afișează acum o fereastră cu lista programelor în curs de depanare, fiecare defalcate la nivel de fir de execuție. În momentul în care un program conține mai multe fire de execuție și este oprit pentru inspecție, utilizatorul poate naviga între ele pentru a inspecta starea corespunzătoare fiecăruia.

Tutoriale

https://helios.utcluj.ro/media/InstalareJDK_Eclipse.mp4

https://helios.utcluj.ro/media/SetareVariabileSistem.mp4

https://helios.utcluj.ro/media/Creare_Rulare_ProiectEclipse.mp4

https://helios.utcluj.ro/media/RulareLinieComanda.mp4

Exemple aplicatii simple Java

Aplicatie minimala Hello World

```
public class Test {

	public static void main(String args[])
		{
	System.out.println("\nHello Test");
		}

	}
```

Aplicatii I/O

a) I/O folosind BufferedReader

```
//import java.io.DataInputStream;
	import java.io.*;
	import java.lang.String;
	import java.io.IOException;

	public class LeapYear {
		 public static void main(String args[])
		 {
				// DataInputStream dis = new DataInputStream(System.in);
	     BufferedReader dis= new BufferedReader(new InputStreamReader(System.in));
		  String Syear=null;
		  System.out.print("Enter a year: ");
		  System.out.flush();
		  try {
		    Syear = dis.readLine();
		  } catch (IOException ioe)
```

```
                        {
                                System.out.println(ioe.toString());
                                System.exit(1);
                        }

                        long year = Long.parseLong(Syear);
                        System.out.println("year is " + year);
                        if ( ((year % 4 == 0) && (year % 100 != 0))  ||
                              (year % 400 == 0) )
                                System.out.println(year + " is a leap year!");
                        else
                                System.out.println(year + " is NOT a leap year.");
                }
        }
```

b) I/O folosind Scanner

```
//import java.io.DataInputStream;
        import java.util.Scanner;

//      import java.lang.String;
//      import java.io.IOException;

        public class LeapYear {
                private static Scanner dis;
                 public static void main(String args[])
                 {
                     //DataInputStream dis = new DataInputStream(System.in);
                      dis = new Scanner(System.in);//System.in is an InputStream

                        String Syear=null;
                        System.out.print("Enter a year: ");
                        System.out.flush();
                       // try {
                         // Syear = dis.readLine();
                            Syear = dis.next();
                        //} catch (IOException ioe)
                         // {
                        //       System.out.println(ioe.toString());
                        //        System.exit(1);
                        //  }

                        long year = Long.parseLong(Syear);
                        System.out.println("year is " + year);
                        if ( ((year % 4 == 0) && (year % 100 != 0))  ||
                              (year % 400 == 0) )
                                System.out.println(year + " is a leap year!");
                        else
                                System.out.println(year + " is NOT a leap year.");
                }
        }



//Leap year clean version
```

```
import java.util.Scanner;

public class LeapYear {
       private static Scanner dis;
        public static void main(String args[])
        {
                    dis = new Scanner(System.in);//System.in is an InputStream

                long year;
                System.out.print("Enter a year: ");
                System.out.flush();
                year = dis.nextLong();
                System.out.println("year is " + year);
                if ( ((year % 4 == 0) && (year % 100 != 0))  ||
                      (year % 400 == 0) )
                        System.out.println(year + " is a leap year!");
                else
                        System.out.println(year + " is NOT a leap year.");
        }
}
```

c) argumente din linia de comanda

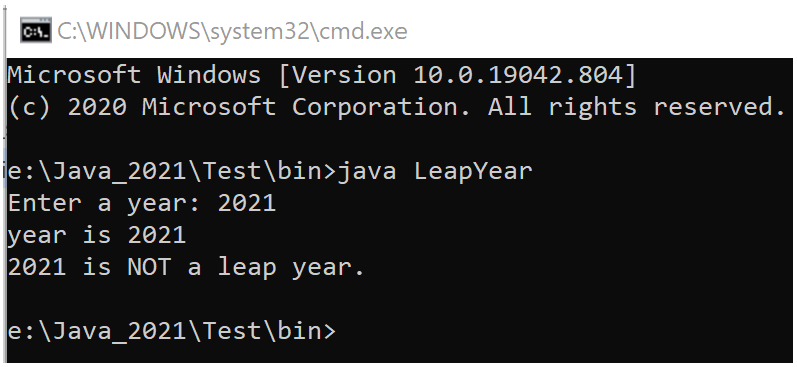
```
//import java.io.DataInputStream;
       import java.io.Console;

       //import java.lang.String;
    //import java.io.IOException;

       public class LeapYear {
              //private static Scanner dis;
               public static void main(String args[])
               {
                   //DataInputStream dis = new DataInputStream(System.in);
                    Console dis = System.console( );

                     String Syear=null;
                    System.out.print("Enter a year: ");
                    System.out.flush();
                    Syear = dis.readLine();

                    long year = Long.parseLong(Syear);
                    System.out.println("year is " + year);
                    if ( ((year % 4 == 0) && (year % 100 != 0))  ||
                          (year % 400 == 0) )
                            System.out.println(year + " is a leap year!");
                    else
                            System.out.println(year + " is NOT a leap year.");
               }
       }
```



d) I/O folosind DataInput.Stream (deprecated)

```
import java.io.DataInputStream;
import java.lang.String;
import java.io.IOException;

public class LeapYear {
       @SuppressWarnings("deprecation")
        public static void main(String args[])
        {
                DataInputStream dis = new DataInputStream(System.in);
                String Syear=null;
                System.out.print("Enter a year: ");
                System.out.flush();
                try {
                  Syear = dis.readLine();
                } catch (IOException ioe)
                  {
                        System.out.println(ioe.toString());
                        System.exit(1);
                  }

                long year = Long.parseLong(Syear);
                System.out.println("year is " + year);
                if ( ((year % 4 == 0) && (year % 100 != 0))  ||
                      (year % 400 == 0) )
                        System.out.println(year + " is a leap year!");
                else
                        System.out.println(year + " is NOT a leap year.");
        }
}
```

Operatii simple folosind biblioteca Math

```
import java.lang.Math;

public class MathM
{
       public static void main(String[] args)
       {
       double x,y,z;
       int i;
              i=10;
```

```
            y=Math.log(10.0);
            x=Math.pow(3.0,4.0);
            z=Math.random(); // random number from 0 to 1
            System.out.println("Value of i is " + i);
            System.out.println("Value of log(10) is " + y);
            System.out.println("Value of 3 at power 4 is " + x);
            System.out.println("A random number is " + z);
            System.out.println("Square root of 2 is " +   Math.sqrt(2));
      }
}
```

Aplicatie grafica

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;

//Clasa care deriva din JPanel pentru a putea modifica modul
//in care panoul este desenat pe ecran

class PanouExemplu extends JPanel {
    private static final long serialVersionUID = 1L;
    //metoda desenare mostenita din clasa de baza
    public void paintComponent(Graphics g) {
        int totalP=6;
        int xP1[]={110,50,200,20,170,110};
        int yP1[]={20,190,80,80,190,20};
        g.drawPolygon(xP1,yP1,totalP);
        int xP2[]={310,250,400,220,370,310};
        int yP2[]={20,190,80,80,190,20};
        g.setColor(Color.blue);
        g.fillPolygon(xP2,yP2,totalP);
      }
    }//PanouExemplu

public class JPanelTest{
      public static void main(String args[ ]) {
        JFrame frame = new JFrame("Exemplu desenare pe panou.");
        frame.add(new PanouExemplu());
        frame.setSize(440, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }//main
}
```

Probleme propuse

Complexitate *

1. Verificati functionalitatile mediului Eclipse considerand aplicatii stand-alone minimale cu operatii de I/E

2. Scrieti o aplicatie Java in care cititi de la tastatura o valoare intreaga si se afiseaza factorialul numarului citit.
3. Scrieti o aplicatie Java in care se citesc de la tastatura 2 valori intregi si se afiseaza cmmdc-ul valorilor.
4. Scrieti o aplicatie Java in care cititi de la tastatura un numar intreg si afisati toti divizorii numarului citi.
5. Scrieti o aplicatie Java in care cititi de la tastatuta mai multe numere intregi si verificati daca sunt numere prime. Numarul de valori citite este preluat de la tastatura sau din linia de comanda.

Complexity *

1. Verify the functionalities of the Eclipse environment considering minimum stand-alone applications with I / O operations
2. Write a Java application where you read from the keyboard an *int* value and the factorial of the number read, is displayed.
3. Write a Java application that reads from the keyboard 2 integer values and displays the cmmdc of the values.
4. Write a Java application where you read from the keyboard an integer and display all the dividers of the number you read.
5. 5. Write a Java application where you can read several integers from the keypad and check that they are prime numbers. The number of read values is taken from the KB or command line.