# LABORATOR 1-Introducere în Java

### Limbajul Java - compilat şi interpretat

Compilare: Un compilator traduce codul scris într-un limbaj de programare în alt limbaj de programare (de obicei, se generează cod maşină din cod sursă).

Interpretare: Un interpretor execută codul sursă pas cu pas, neavând loc neapărat o traducere prealabilă.

Limbajul Java - Compilare şi interpretare: Codul scris în Java (fişiere .java) este compilat şi transformat în bytecode (fişiere .class). Acest bytecode este interpretat şi executat de maşina virtuală Java.

Bytecode-ul generat de către compilatorul Java este independent de platformă. Maşina virtuală este cea care depinde de platforma pe care rulează. De aici rezultă şi portabilitatea aplicaţiilor scrise în limbajul Java - este necesară doar instalarea maşinii virtuale care ştie să interpreteze şi să execute bytecode-ul pe platforma dorită.

### Crearea unei aplicaţii de tip Hello World

Înainte de a scrie prima voastră aplicaţie în Java, trebuie să ţineţi cont de următoarele aspecte:

1. Orice clasă publică trebuie scrisă în fişierul său propriu, care îi poartă numele. Nu este permisă declararea mai multor clase publice în acelaşi fişier.
2. Execuţia programului porneşte de la o metodă "main" care trebuie să aibă următoarea signatură:

public static void main(String args[]);

1. Argumentele primite în linia de comandă sunt preluate în parametrul args, începând cu poziţia 0, nu 1, precum în C/C++.

Astfel, am putea crea următoarea clasă HelloWorld în fişierul "HelloWorld.java":

// HelloWorld.java   
// ---------------   
  
public class HelloWorld {   
   public static void main(String args[]) {   
      if(args.length > 0)   
         System.out.println("Hello, " + args[0] + "!");   
      else   
         System.out.println("Hello, world!");   
   }   
}

### Compilare în linia de comandă

javac HelloWorld.java

Fişierul va fi transformat în "HelloWorld.class".

### Execuţie în linia de comandă

java HelloWorld   
java HelloWorld Vasile

Bytecode-ul din fişierul "HelloWorld.class" va fi interpretat şi executat de către maşina virtuală. Rezultatele celor două execuţii vor fi "Hello, world!", respectiv "Hello, Vasile!".

### Tips & Tricks - Laboratorul 1

1. Folosiţi cu încredere [documentaţia](http://docs.oracle.com/javase/7/docs/api/) pentru a găsi clasele de care aveţi nevoie şi ce metode pun la dispoziţie acestea.
2. Inspectaţi clasa [System](http://docs.oracle.com/javase/7/docs/api/java/lang/System.html). Observaţi câmpurile statice err, in şi out. Scrierea în err şi în out se face folosind metodele puse la dispoziţie de clasa [PrintStream](http://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html) (uzual, print() şi println()).
3. Pentru a găsi o modalitate de a transforma un şir de caractere în întreg, inspectaţi metodele statice ale clasei [Integer](http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html).
4. O idee pentru numărarea zecimalelor v-ar putea veni dacă aruncaţi o privire asupra metodei [toString()](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html#toString%28%29) din clasa Double. Soluții ceva mai frumoase ar fi cele care folosesc [DecimalFormat](http://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html) sau [BigDecimal](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html).
5. Atunci când faceţi operaţii cu şiruri de caractere (de cele mai multe ori, concatenarea), pentru fiecare astfel de operaţie se creează un obiect nou şi se realocă memorie. Din acest motiv, instanţele clasei String intră în categoria de obiecte "immutable". Pentru ca operaţiile să se realizeze asupra aceleiaşi instanţe, folosiţi clasa [StringBuilder](http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html).
6. Există o metodă cu acelaşi nume atât în clasa String, cât şi în clasa StringBuilder, care vă permite să accesaţi caracterul de la o anumită poziţie.
7. Pentru colecţii, una din clasele pe care le puteţi folosi este [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html). Pentru sortarea unei liste, găsiţi o metodă folositoare în clasa [Collections](http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html).

## Antrenament examen

1. Care dintre următoarele atribuiri sunt greșite din punct de vedere sintactic (vor provoca eroare la compilare) ?
   1. **byte a = 128;**
   2. short b = 128 \* 129;
   3. long c = 0xC;
   4. double d = (float) 10E2;

Compilatorul își dă seama că 128 este prea mare pentru a fi reținut într-un byte. Dar nu face calculul efectiv 128 \* 129 pentru verifica dacă poate fi reținut într-un short.

1. Care dintre următoarele instrucțiuni de import sunt greșite din punct de vedere sintactic ?
   1. import java.lang.Object;
   2. import java.\*;
   3. **import package java.awt;**
   4. import static java.sql.Types.\*;
2. Este corectă secvența de mai jos? Dacă da, care va fi rezultatul afișat? Motivați răspunsul.

String s1 = Integer.parseInt(1);

String s2 = Integer.parseInt(2);

System.out.println((s1+s2) == new String("12"));

Secvența este greșită și va produce eroare la compilare deoarece metoda parseInt() trebuie să primească un String și returnează un int - invers față de cum este scris.

1. Declarați și instanțiați o matrice M, cu trei linii și două coloane, cu elementele M(i, j)=10\*i+j, într-o singură instrucțiune, fără a folosi structuri repetitive.

int M[][] = {{0, 1}, {10, 11}, {20, 21}};

sau

int M[][] = new int[][] {{0, 1}, {10, 11}, {20, 21}};

1. Care dintre următoarele variante nu este corectă?
   1. Float a[] = new Float[10];
   2. **Byte b[10] = new Byte[];**
   3. Integer c[] = {10, 20};
   4. **String d[] = {'x', 'y', 'z'};**
2. Ce va afișa secvența de mai jos?

int a,b;

for (a = 10, b = 1; a-- > ++b ; ) ;

System.out.println((a--) + "," + (++b));

* 1. 4,5
  2. 5,6
  3. **5,7**
  4. 6,7

1. Care dintre următoarele variante nu este corectă?
   1. **char []a = {"1", "2"};**
   2. **double b[4] = new double[];**
   3. int c[][] = {{1, 2}, {3, 4}};
2. Ce va afișa la execuție programul alăturat?

public class Asignare {

public static void main(String args[]) {

int a = 3;

int b = (a = 2) \* a;

int c = b \* (b = 5);

System.out.println("a = " + a + ", b = " + b + ", c = " + c);

}

}

a = 2, b = 5, c = 20

1. Ce se poate spune despre subprogramul Java alăturat?

int i = 1, suma = 0;

for ( ; ; ) {

suma += i++;

if (i > 5) break;

}

System.out.print(suma);

* 1. Eroare la compilare: lipsesc părțile componente ale lui for
  2. Subprogramul se compilează și la execuție afișează 0
  3. Subprogramul se compilează și la execuție afișează 10
  4. **Subprogramul se compilează și la execuție afișează 15**

1. Care vor fi valorile variabilelor a, b și c după execuția secvenței de mai jos?

int a = 0, b = 1, c = (++a >= b-- ? a ++ : ++b);

* 1. a=2, b=1, c=1
  2. a=1, b=0, c=1
  3. a=2, b=1, c=0
  4. **a=2, b=0, c=1**

# LABORATOR 2: Obiecte și clase

### Formatarea codului

Începând cu acest laborator, trebuie să respectaţi [convenţiile de scriere a codului pentru limbajul Java](http://www.oracle.com/technetwork/java/codeconvtoc-136057.html). Altfel, nerespectarea lor va atrage o penalizare de 0.5 la punctajul temei.

Code conventions are important to programmers for a number of reasons:

* 80% of the lifetime cost of a piece of software goes to maintenance.
* Hardly any software is maintained for its whole life by the original author.
* Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
* If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

### Generare javadoc

Începând cu acest laborator, trebuie să documentaţi codul scris pentru cel puţin una dintre problemele aferente fiecărui laborator. Nerespectarea acestei cerinţe va atrage o penalizare de 0.5 la punctajul temei.

În acest scop, trebuie să adăugaţi comentarii de forma /\*\* \*/ deasupra claselor, metodelor, câmpurilor. Consultaţi [regulile de scriere a acestor comentarii](http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html). Utilitarul javadoc le va putea interpreta şi transforma în fişiere .html cu aceeaşi formă ca şi [documentaţia limbajului Java](http://docs.oracle.com/javase/7/docs/api/).

Generarea documentaţiei cu utilitarul javadoc:

* NetBeans: Click dreapta proiect -> Generate Javadoc
* Eclipse: Project -> Generate Javadoc
* IntelliJ IDEA: Tools -> Generate JavaDoc

### Metodele toString(), equals(), hashCode()

Rădăcina [ierarhiei de clase](http://docs.oracle.com/javase/7/docs/api/overview-tree.html) din limbajul Java este clasa Object. Printre metodele acestei clase se numără şi toString(), equals(), hashCode(). Printre cerinţele laboratorului se numără şi suprascrierea acestor metode în clasele create de voi. Consultaţi documentaţia pentru a înţelege scopul lor.

Descrierea metodelor:

* [toString()](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#toString%28%29)
* [equals()](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals%28java.lang.Object%29)
* [hashCode()](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29)

În Java, operatorul "==" poate fi folosit corect doar la compararea tipurilor primitive. Atunci când este folosit pentru obiecte, el realizează doar compararea referinţelor, nu a conţinutului obiectului! Compararea corectă a două obiecte se face doar cu ajutorul metodei equals().

Analizaţi şi discutaţi output-ul următoarei secvenţe de cod:

String s1 = "abcd";   
String s2 = "abcd";   
  
String abc = "abc";   
String s3 = abc + "d";   
  
System.out.println(s1 == s2);   
System.out.println(s1 == s3);   
System.out.println(s1.equals(s2));   
System.out.println(s1.equals(s3));   
  
int int1 = 10;   
int int2 = 10;   
  
System.out.println(int1 == int2);   
  
Integer i1 = new Integer(10);   
Integer i2 = new Integer(10);   
  
System.out.println(i1 == i2);   
System.out.println(i1.equals(i2));

Vezi și [Save Memory by Using String Intern in Java](https://blog.codecentric.de/en/2012/03/save-memory-by-using-string-intern-in-java/).

### Tips & Tricks - Laborator 2

1. În limbajul Java nu există posibilitatea de a supraîncărca operatori. Creaţi metode proprii pentru realizarea operaţiilor cerute.
2. Folosiţi cât mai mult facilitatea de generare de cod pe care o pune la dispoziţie IDE-ul folosit, în special pentru constructori, getter-i şi setter-i, toString(), equals() şi hashCode().
3. Consultaţi câteva [sugestii](http://en.wikipedia.org/wiki/Sparse_matrix#Storing_a_sparse_matrix) pentru reprezentarea matricilor rare. O implementare frumoasă şi eficientă va primi bonus generos :)
4. Gândiţi-vă la şabloane de proiectare şi algoritmi pentru grafuri care v-ar putea uşura munca la cea de-a doua problemă. Puteţi primi bonus şi pentru folosirea acestora.

## Antrenament examen

1. Care afirmații legate de metoda hashCode() a clasei Object sunt false?
   1. Trebuie să returneze un număr întreg;
   2. **Trebuie să returneze numere diferite pentru obiecte diferite;**
   3. **Nu poate fi supradefinită;**
   4. **Nu poate fi supraîncărcată.**
2. Care afirmații legate de metoda equals() a clasei Object sunt false?
   1. **Trebuie obligatoriu corelată cu metoda hashCode();**
   2. **Trebuie obligatoriu supradefinită;**
   3. Metoda acceptă orice obiect ca argument;
   4. Metoda poate fi aplicată pentru tablouri cu valori primitive.
3. Ce se va întâmpla la încercarea de a compila fișierul Test.java, descris mai jos?

class A extends Object {}

class B extends A {

private class C extends B {}

}

* + 1. Nu va fi compilat deoarece... (completați)
    2. **Va fi compilat și vor fi create unitățile de compilare... (completați)**

A.class, B.class, B$C.class

1. Ce se va întâmpla la încercarea de a compila fișierul Test.java, descris mai jos?

public class A { static int A = 1; }

class B extends A {

B() { A = 2; }

}

Se va compila

1. Explicați pe scurt cum poate fi implementat șablonul de proiectare Immutable Object și dați un exemplu din API-ul standard Java.

Câmpuri private, inițializate doar din constructor. Se expun doar getter-i, nu și setter-i. Dacă se implementează operații, aceastea ar trebui să creeze o nouă instanță și să o returneze, nu să modifice instanța curentă.   
Exemplu din API-ul standard Java: String.

1. Ce puteți spune despre programul de mai jos?

class A {

A(int x) { System.out.println("apel constructor A"); } }

class B extends A {

B() { System.out.println("apel constructor B"); } }

public class C {

public static void main(String args[]) { B b = new B(); } }

**a.Nu va fi compilat;**

b.Va afișa "apel constructor B";

c.Va afișa "apel constructor A";

d.Va afișa "apel constructor B" și apoi "apel constructor A";

1. Ce semnificație are cuvântul cheie super folosit ca metodă?

a.Apelează constructorul corespunzător ca număr de argumente al clasei;

**b.Apelează constructorul corespunzător ca număr de argumente al superclasei;**

c.Face referire la variabilele de instanță ale superclasei;

d.Cuvâtntul cheie super nu poate fi folosit ca metodă.

1. Ce puteți spune despre programul alăturat?

class C1 {

static String f() { return "Mesajul Unu din C1"; }

String g() { return "Mesajul Doi din C1"; }

}

class C2 extends C1 {

static String f() { return "Mesajul Unu din C2"; }

String g() { return "Mesajul Doi din C2"; }

}

public class Test {

public static void main(String args[]) {

C1 obiect = new C2();

System.out.println(obiect.f() + ", " + obiect.g());

}

}

Deoarece in cazul de fata ambele metode f sunt statice si decic and cream obiect de tipul C2 va ramane cu apelul metodei f de pe C1

a.programul este corect și va afișa la execuție: Mesajul Unu din C1, Mesajul Doi din C1

**b.programul este corect și va afișa la execuție: Mesajul Unu din C1, Mesajul Doi din C2**

c.programul este corect și va afișa la execuție: Mesajul Unu din C2, Mesajul Doi din C1

d.programul este corect și va afișa la execuție: Mesajul Unu din C2, Mesajul Doi din C2

e.va apărea eroare la compilare deoarece în clasa Test variabila obiect nu aparține clasei C2

1. Ce puteți spune despre prgramul alăturat?

class C1 {

int x = 1;

void f(int x) { this.x = x; }

int getX\_C1() { return x; }

}

class C2 extends C1 {

float x = 5.0f;

int f(int x) { super.f((int)x); }

float getX\_C2() { return x; }

}

public class SuprascriereSiAscundere {

public static void main(String args[]) {

C2 obiect = new C2();

obiect.f(4);

System.out.print(obiect.getX\_C2() + " ");

System.out.println(obiect.getX\_C1());

}

}

* + 1. programul este corect și va afișa la execuție: 5 4
    2. programul este corect și va afișa la execuție: 5.0 4
    3. programul este corect și va afișa la execuție: 4.0 4
    4. va apărea eroare la compilare deoarece în clasa C2 s-a suprascris greșit atributul x din clasa C1
    5. **va apărea eroare la compilare deoarece metoda suprascrisă f() din clasa C2 întoarce un tip diferit de void**

1. Ce va afișa la execuție aplicația CtorDemo?

class A{

int a=1;

A() { a=2; }

}

class B extends A {

int b=3;

B() { b=a; }

}

public class CtorDemo {

public static void main(String args[]) {

B bobj = new B();

System.out.println(bobj.b);

}

}

a.1

**b.2**

c.3

d.Nu se va afișa nimic deoarece codul este incorect

# LABORATOR 3: Excepţii. Fluxuri (intrări/ieşiri)

## Excepţii

Generarea unei excepţii:

throw new Exception("mesaj");

Declararea că o metodă poate genera o excepţie:

public void metoda() throws IOException {   
    // Cod care ar putea genera IOException   
}

Tratarea unei excepţii:

try {   
    // Cod care ar putea genera excepţii   
} catch (Exception ex) {   
    // Cod pentru tratarea excepţiei   
} finally {   
    // Cod care se execută indiferent dacă apar excepţii sau nu   
}

Rădăcina excepţiilor şi erorilor din Java este clasa Throwable. Doar instanţele acestei clase sau ale subclaselor sale pot fi generate de maşina virtuală Java sau prin interemdiul instrucţiunii throw. De asemenea, doar această clasă sau o subclasă a sa poate fi tipul unui argument din clauza catch.

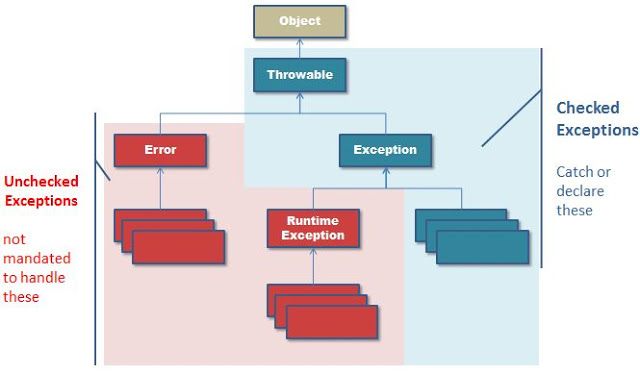


Figura 1. Ierarhia excepţiilor

Checked vs. unchecked:

* Checked exceptions - Sunt detectate la compilare, moştenesc clasele Throwable sau Exception, cu excepţia clasei RuntimeException. Trebuie obligatoriu tratate (try-catch) sau declarate ca putând fi generate de metoda respectivă (throws).   
  **Exemple**: FileNotFoundException, MalformedURLException, ClassNotFoundException.
* Unchecked exceptions - Sunt detectate la execuţie, moştenesc clasele Error sau RuntimeException. Tratarea lor este posibilă, dar nu este obligatorie.   
  **Exemple**: NullPointerException, ArrayIndexOutOfBoundsException; OutOfMemoryError, StackOverflowError.

Eroare vs. excepţie:

* Excepţie - Surprinde apariţia unei situaţii în care aplicaţia nu şi-ar mai putea continua execuţia în modul dorit, însă îşi poate reveni din această situaţie. Spre exemplu, dorim să scriem într-un fişier, dar acesta nu există. Nu este un motiv rezonabil pentru ca aplicaţia să fie oprită. O aplicaţie bine construită ar trebui să îşi poată reveni şi continua execuţia în astfel de situaţii, deci cazurile de excepţie ar trebui tratate.
* Eroare - Surprinde apariţia unei situaţii în care aplicaţia nu îşi mai poate continua execuţia, iar revenirea din această situaţie nu este posibilă şi nici indicată. Spre exemplu, în cazul unei erori de tipul OutOfMemoryError, aplicaţia ar trebui să fie oprită. Din acest motiv, erorile nu trebuie tratate.

## Fluxuri

În vârful ierarhiei pentru operaţii de I/O stau patru clase abstracte:

* InputStream, OutputStream - Pentru lucrul cu fluxuri de octeţi (8 biţi).
* Reader, Writer - Pentru lucrul cu fluxuri de caractere (16 biţi);

Toate clasele existente pentru lucrul cu fluxuri, fie că e vorba de fluxuri primitive sau de procesare, moştenesc una din aceste patru clase abstracte. Pentru mai multe detalii, vezi [cursul](http://thor.info.uaic.ro/%7Eacf/java/slides/fluxuri_slide.pdf) sau [documentaţia](http://docs.oracle.com/javase/7/docs/api/).

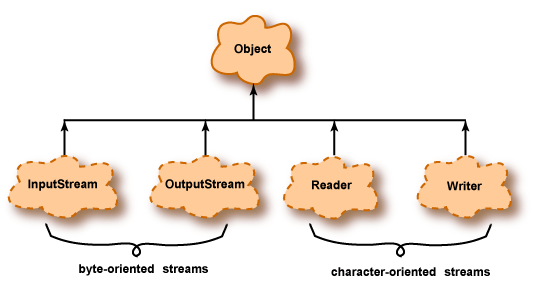


Figura 2. Vârful ierarhiei I/O

Funcţionalitatea de bază pentru un flux de intrare sau ieşire este citirea, respectiv scrierea unor date. Metodele abstracte de citire sau scriere sunt menite să lucreze la cel mai de jos nivel cu un flux nestructurat.

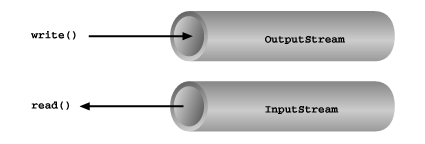


Figura 3. Funcţionalitatea de bază pentru fluxuri input/output

Implementările concrete ale celor patru clase amintite mai sus servesc pentru diverse scopuri: citire din fişier, citire din memorie, citire buffer-izată, conversii, serializare etc. Din nou, pentru mai multe detalii, vezi [cursul](http://thor.info.uaic.ro/%7Eacf/java/slides/fluxuri_slide.pdf) sau [documentaţia](http://docs.oracle.com/javase/7/docs/api/).

Diferenţierea între fluxuri primitive şi fluxuri de procesare se face prin intermediul constructorului. Cele primitive primesc ca parametru un dispozitiv extern (e.g. un nume de fişier). Cele de procesare primesc ca parametru un alt flux, de acelaşi tip.

De cele mai multe ori, funcţionalitatea pe care o dorim necesită combinarea mai multor clase dintre cele care implementează InputStream/OutputStream sau Reader/Writer. Acest lucru este posibil datorită şablonului de proiectare Decorator, folosit in implementarea acestor clase (i.e. faptul că un flux poate fi construit pornind de la alt flux de acelaşi tip este un indicator al folosirii acestui şablon de proiectare).

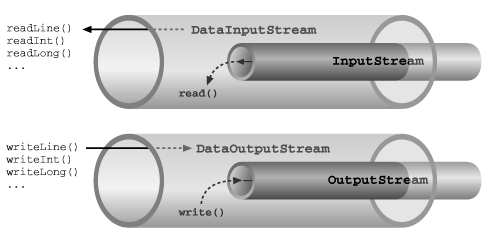


Figura 4. Fluxuri input/output decorate

Exemplul 1. Dorim să citim un fişier linie cu linie în mod buffer-izat. Avem nevoie de:

* O instanţă FileReader - flux primar care permite citirea dintr-un fişier;
* O instanţă BufferedReader - flux de procesare care permite citirea buffer-izată dintr-un Reader.

FileReader fr = new FileReader("fisier.txt");   
BufferedReader br = new BufferedReader(fr);   
  
String line = br.readLine();   
while(line != null) {   
   System.out.println(line);   
   line = br.readLine();   
}   
br.close();

Exemplul 2. Dorim să scriem într-un fişier diverse tipuri primitive în mod buffer-izat.

* O instanţă FileOutputStream - flux primar care permite scrierea într-un fişier;
* O instanţă BufferedOutputStream - flux de procesare care permite scrierea buffer-izată dintr-un OutputStream.
* O instanţă DataOutputStream - flux de procesare care permite scrierea tipurilor primitive într-un OutputStream;

FileOutputStream fos = new FileOutputStream("out");   
BufferedOutputStream bos = new BufferedOutputStream(fos);   
DataOutputStream dos = new DataOutputStream(bos);   
  
dos.writeBoolean(true);   
dos.writeDouble(3.1416);   
dos.writeInt(1234);   
dos.writeLong(123456789);   
dos.close();

Notă: Aceste exemple prezintă doar nişte variante de rezolvare. Puteţi găsi şi alte metode pentru a realiza aceleaşi funcţionalităţi.

### Tips & Tricks - laborator 3

1. Pentru crearea rapoartelor, recomand [Freemarker](http://freemarker.sourceforge.net/) sau [JasperReports | iReport](http://www.jaspersoft.com/JasperSoft_iReport.html). Pentru iReport, există şi un [plug-in pentru NetBeans.](http://plugins.netbeans.org/plugin/4425/ireport)
2. Pentru mai multe detalii despre lucrul cu date calendaristice, consultaţi lecţia [Dates and Times](http://docs.oracle.com/javase/tutorial/i18n/format/dateintro.html) din tutorialul Java.
3. Clasa [File](http://docs.oracle.com/javase/7/docs/api/java/io/File.html) poate reprezenta atât un fişier, cât şi un director. Dintre metodele acestei clase, v-ar putea fi utile isDirectory(), isFile(), list(). Uitaţi-vă şi la metodele statice din clasa [Files](http://docs.oracle.com/javase/7/docs/api/java/nio/file/Files.html). Pentru mai multe detalii despre lucrul cu fişiere, consultaţi lecţia [File I/O (Featuring NIO.2)](http://docs.oracle.com/javase/tutorial/essential/io/fileio.html), în special secţiunea [Listing a Directory's Contents](http://docs.oracle.com/javase/tutorial/essential/io/dirs.html#listdir).
4. Pentru a vă uşura munca la interpetarea comenzilor din shell, folosiţi-vă de [expresii regulate](http://docs.oracle.com/javase/tutorial/essential/regex/).

## Antrenament examen

1. Care dintre următoarele variante nu este corectă?
   1. **Writer a = new Writer("file.txt");**
   2. Writer b = new PrintWriter(new FileWriter("file.txt"));
   3. Writer c = new FileWriter("file.txt");
   4. **BufferedWriter d = new BufferedWriter("file.txt");**
2. Ce puteți spune despre subprogramul alăturat?

double x = 0;

if (Double.isInfinite(2 / x))

System.out.println("infinit");

else

System.out.println("2 / 0");

* 1. Eroare la compilare datorită împărțirii la zero
  2. Eroare la execuție datorită împărțirii la zero (se aruncă o excepție ArithmeticException)
  3. **Programul este corect și va afișa: infinit**
  4. Programul este corect și va afișa: NaN

1. Presupunem că avem un fragment de cod Java dintr-o aplicație care are drept de scriere în directorul de lucru curent și că în directorul curent nu există fișierul numit "fisier.txt". Care va fi rezultatul compilării și execuției codului alăturat?

try {

RandomAccessFile f1 = new RandomAccessFile("fisier.txt", "rw");

BufferedOutputStream f2 = new BufferedOutputStream(f1);

DataOutputStream f3 = new DataOutputStream(f2);

f3.writeDouble(Math.PI);

f3.close();

f2.close();

f1.close();

} catch(IOException e) { }

* 1. **Codul Java de mai sus nu se poate compila;**
  2. Codul Java de mai sus se poate compila, dar aruncă o excepție la linia 3;
  3. Codul Java de mai sus se poate compila și executa dar nu are nici un efect pe sistemul de fișiere local;
  4. Codul Java de mai sus se poate compila și executa, creându-se fișierul "fișier.txt" în directorul curent.

1. Ce se va întâmpla la execuția secvenței alăturate?

int test = 0, infinit = 0;

try {

infinit = 1/0;

test ++;

}

catch(Exception e) { test ++; }

finally { test ++; }

System.out.println("test=" + test + " infinit=" + infinit);

* 1. Se va afișa: test=2 infinit=NaN
  2. **Se va afișa: test=2 infinit=0**
  3. Se va afișa: test=1 infinit=0
  4. Nu se va afișa nimic, programul se va opri cu excepție

1. Ce se va afișa la execuția următoarei secvențe de cod:

RuntimeException a = new RuntimeException();

System.out.println(a.getClass().getSuperclass().getSuperclass());

* 1. class java.lang.Exception
  2. class java.lang.Error
  3. **class java.lang.Throwable**
  4. class java.lang.Object

1. Fie codul de mai jos. Care este rezultatul execuției lui?

File f1 = new File("numeDirector");

File f2 = new File(f1, "numeFisier");

* 1. eroare la compilare la linia 2, deoarece avem un apel recursiv nepermis
  2. programul este corect și se creează în directorul curent un director cu numele numeDirector, iar în acest director se crează fișierul cu numele numeFisier
  3. programul este corect și se creează în directorul curent un director cu numele numeDirector și fișierul cu numele numeFisier.
  4. programul este corect și se creează în directorul curent doar fișierul cu numele numeFisier
  5. **programul este corect, dar nu se creează nici un director și nici un fișier**

1. Care dintre afirmațiile de mai jos sunt adevărate?
   1. când se creează o instanță a lui File și nu se folosește semantica de denumire a fișierelor de pe mașina locală, atunci constructorul va arunca o excepție IOException
   2. când se creează o instanță a lui File și fișierul respectiv nu există pe sistemul de fișiere local, atunci se creează un astfel de fișier
   3. când o instanță a lui File este eliberată de garbage collector, atunci fișierul corespunzător de pe sistemul local este șters
   4. **nici una dintre afirmațiile de mai sus nu este adevărată**
2. Ce se va întâmpla la execuția următoarei secvențe?

float a=0;

int b=0, test=0;

try {

a=1/a; test ++;

b=1/b; test ++;

}

catch(Exception e) { test ++; }

finally { test ++; }

System.out.println("test=" + test + " a=" + a + " b=" + b);

* 1. Nu se va afișa nimic, prograul se va opri cu excepție
  2. Se va afișa: test=2 a=Infinity b=Infinity
  3. **Se va afișa: test=3 a=Infinity b=0**
  4. Se va afișa: test=4 a=0 b=0

1. Care dintre următoarele variante nu este corectă?
   1. FileWriter a = new FileWriter("file.txt");
   2. Writer b = new PrintWriter(new FileWriter("file.txt"));
   3. PrintWriter c = new PrintWriter("file.txt");
   4. **BufferedWriter d = new BufferedWriter("file.txt");**
2. Care este superclasa tutor claselor ce descriu excepții sau erori în Java?
   1. **Throwable**
   2. Exception
   3. RuntimeException
   4. Error

# LABORATOR 4: Interfețe. Pachete. Serializarea obiectelor. Colecții

## Discuții referitoare la curs

### Interfețe

* Care sunt modificatorii impliciți pentru atribute și metode când nu sunt specificați explicit de programator?
* Care sunt relațiile care se pot stabili între mai multe interfețe? Dar între clase și interfețe?
* Cum se poate implementa o metodă de tip callback folosind interfețe?
* Ce sunt clasele anonime? Unde și cum le putem instanția? Cum sunt compilate aceste clase?
* Care este diferența între interfața Compareable și interfața Comparator?

### Pachete

* Cum evităm ambiguitățile când dorim să folosim clase cu același nume din pachete diferite?
* Ce este classpath și la ce folosește?

### Serializarea obiectelor

* Ce fluxuri sunt folosite la serializare?
* Cum funcționează mecanismul implicit de serializare? Cum trebuie să fie toate atributele unei clase pentru a putea fi serializată cu succes? Ce se întâmplă cu un obiect serializat la deserializare când i-am modificat clasa?
* Cum funcționează modificatorul transient? Ce se întâmplă cu membrii privați ai unei clase la serializare?
* Cum putem să ne implementăm propriul mecanism de serializare?

### Colecții

* Cum se definesc tipurile și metodele generice? Cum se folosesc? Cum se pot delimita?
* Ce legătură există între MyType<A> și MyType<B>, dacă A extinde B și MyType este un tip generic?
* Cum arată ierarhia colecțiilor în Java? Numiți câteva implementări concrete.
* Cum pot fi parcurse colecțiile? Cum funcționează java.util.Collections.sort()?

### Tips & Tricks - laborator 4

1. Doar cea de-a doua problemă cere serializarea binară a obiectelor. În prima problemă se va folosi în mod obligatoriu formatul [TGF (Trivial Graph Format)](http://en.wikipedia.org/wiki/Trivial_Graph_Format).
2. Trimiterea ca argument a unei metode se va face precum în exemplul din [curs](http://thor.info.uaic.ro/%7Eacf/java/slides/interfete_slide.pdf) (slide 10). Orice altă abordare nu va fi punctată.
3. Pentru prima problemă puteţi primi bonus atât pentru elemente avansate de algoritmica grafurilor, cât şi pentru folosirea unei biblioteci de desenare a grafurilor ([jgraph](http://www.jgraph.com/), [jung](http://jung.sourceforge.net/), [prefuse](http://prefuse.org/) etc.).
4. La a doua problemă nu uiţaţi de fişierul de configurare care specifică caracterele folosite pentru reprezentarea tablei de joc si a pieselor. Acesta poate avea orice formă doriţi voi. Puteţi încerca să folosiţi un [fişier .properties](http://www.mkyong.com/java/java-properties-file-examples/).
5. Jocul [Atari Go](http://senseis.xmp.net/?AtariGo) funcţionează după aceleaşi [reguli](http://en.wikipedia.org/wiki/Rules_of_Go) ca şi jocul de Go. Diferenţa (şi simplitatea) constă în faptul că jocul de Atari Go este câştigat de cel care realizează prima captură.

## Antrenament examen

1. O aplicaţie de gestiune a studenţilor unei universităţi conţine clasa Student şi câte o structură de date List<Student> studenti; cu studenţii fiecărei facultăţi. Care din invocările a) Arrays.sort(studenti) sau **b) Collections.sort(studenti)** este corectă și în ce condiții?

Clasa Student trebuie să implementeze interfața Comparable și metoda compareTo().

1. În cadrul unei aplicații doriți să utilizați o clasă pentru lucru cu matrici rare cu numele org.math.cool.SparseMatrix, clasă aflată în arhiva coolmath.jar. Care sunt pașii pe care trebuie să îi faceți pentru a realiza acest lucru?

Se adaugă arhiva în classpath. Pot fi menționați și pașii specifici IDE-ului pe care il folosiți.

1. O aplicație se conectează la un serviciu Web de unde citește valorile cursurilor valutare la o anumită dată. Declarați și instanțiați, folosind tipuri generice, o colecție care să poată memora informațiile citite de la respectivul serviciu Web.

HashMap<String, Double> map = new HashMap<String, Double>();

1. Ce puteți spune despre programul alăturat?

interface I1 {

int j = 3;

int [] i = { j + 1, j + 2};

}

interface I2 {

int j = 5;

int [] i = { j + 2, j + 4};

}

public class Test implements I1, I2 {

public static void main(String[] args) {

System.out.print(I1.j + " ");

System.out.print(I2.i[1] + " ");

System.out.print(I1.i[0]);

}

}

* 1. **programul este corect și va afișa la execuție: 3 9 4**
  2. va apărea eroare la compilare deoarece interfețele I1 și I2 conțin declarațiile acelorași atribute
  3. va apărea eroare la compilare deoarece în interfețele I1 și I2 atributul i conține referință la atributul j
  4. va apărea eroare la compilare deoarece interfețele I1 și I2 conțin declarațiile unui atribut de tip tablou
  5. va apărea eroare la compilare deoarece atributele interfețelor I1 și I2 nu se pot accesa dintr-o metodă statică

1. În ce condiții putem apela o metodă de sortare polimorfică unei colecții formate din obiecte de același tip?
   1. **Metoda de sortare primește un argument de tip Comparator;**
   2. Este suficientă supradefinirea metodei hashCode() în clasa care descrie obiectele;
   3. **Clasa care descrie obiectele implementează interfața Comparable și metoda compareTo;**
   4. Apelul este posibil în orice situație, fiind apelată metoda implicită equals() a clasei Object.
2. Să considerăm declarațiile alăturate? Care dintre următoarele variante nu este corectă?

class A{

static int x = 37;

transient int y = 47;

static void f() {

A aref = this;

int i = x;

int j = y;

g();

}

void g() { }

}

* 1. **Declarația lui aref**
  2. Declarația lui i
  3. **Declarația lui j**
  4. **Apelul metodei g()**

1. Să considerăm declarațiile alăturate de clase și interfețe. Care dintre acestea nu este corectă?

interface Test { }

abstract class AbstractImpl implements Test { }

class TestImpl extends AbstractImpl { }

* 1. **Test t = new Test();**
  2. **TestImpl t = new AbstractImpl();**
  3. **AbstractImpl t = new Test();**
  4. AbstractImpl t = new TestImpl();

1. Ce puteți spune despre aplicația TestImpl?

public class TestImpl {

public static void main(String[] args) {

HelloImpl h1 = new HelloImpl(), h2 = h1;

h1.message = "Salut";

System.out.println(h2.sayHello());

}

}

interface Hello {

String sayHello();

}

class HelloImpl implements Hello {

static String message = "Hello";

String sayHello() {

return message;

}

}

Nu va fi compilate deoarece in HelloImpl la implementarea metodei sayHello trebuia sa fie in fata modificatorul de acces public deoarece cand declare o interfata toate metodele ei sunt declarate FARA DREPT DE MODIFICARE ca fiind public

* 1. Va afișa la execuție mesajul "Hello";
  2. Va afișa la execuție mesajul "Salut";
  3. Se va opri cu o excepție de tipul NullPointerException;
  4. **Nu va fi compilată.**

1. Ce înțelegeți prin serializare?
   1. **Scrierea unui obiect într-un fișier de unde să poată fi refăcut ulterior;**
   2. Transformarea unui obiect într-un flux de octeți;
   3. Adăugarea unui număr serial unui obiect pentru a putea fi deosebit de alte obiecte ale aceleiași clase, dar din alte versiuni;
   4. Transmiterea obiectelor prin rețea.
2. Care dintre afirmațiile următoare referitoare la serializarea obiectelor nu este adevărată?
   1. Pentru a fi serializabilă o clasă trebuie obligatoriu să implementeze una din interfețele Serializable sau Externalizable;
   2. Obiectele serializate pot fi scrise atat in fișiere, cât și în memorie;
   3. **Variabilele declarate cu modificatorii transient sau private nu participă la serializare.**
   4. Mecanismul implicit de serializare a obiectelor unei clase poate fi modificat;
   5. Clasele ObjectInputStream și ObjectOutputStream permit și serializarea tipurilor primitive de date.

# LABORATOR 5: Interfaţa grafică cu utilizatorul

## Anonymity CheckerSwing

Swing este GUI widget toolkit-ul primar din Java, făcând parte din JFC (Java Foundation Classes)- un API pentru realizarea de interfeţe grafice cu utilizatorul. Swing este construit peste AWT (Abstract Window Toolkit), iar în cadrul său se regăsesc:

* [componente top level](http://docs.oracle.com/javase/tutorial/uiswing/components/toplevel.html): JFrame, JDialog, JWindow (derivate din java.awt.Window), JApplet (derivat din java.awt.Panel);
* alte componente, derivate din [clasa JComponent](http://docs.oracle.com/javase/tutorial/uiswing/components/jcomponent.html). Pentru mai multe informaţii despre aceastea, consultaţi [tutorialul Java](http://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html).

Figura 5. Ierarhia de clase din AWT şi Swing

Pentru aranjarea componentelor în cadrul unui container se folosesc [gestionare de poziţionare](http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html). Pentru a le înţelege modul de utilizare, consultaţi capitolele dedicate din cadrul tutorialului. Printre cele mai utilizate sunt [BorderLayout](http://docs.oracle.com/javase/tutorial/uiswing/layout/border.html) şi [GridLayout](http://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html).

Acţiunile efectuate de utilizator asupra componentelor interfeţei grafice generează evenimente. Pentru a trata aceste evenimente, trebuie să înregistrăm un listener pentru acea componentă (recunoaşteţi şablonul de proiectare folosit?). Pentru explicaţii detaliate, consultaţi capitolul [Writing Event Listeners](http://docs.oracle.com/javase/tutorial/uiswing/events/index.html) din tutorialul Java.

## Exerciţiu

Identificaţi componentele şi gestionarii de poziţie utilizaţi în interfaţa grafică de mai jos.

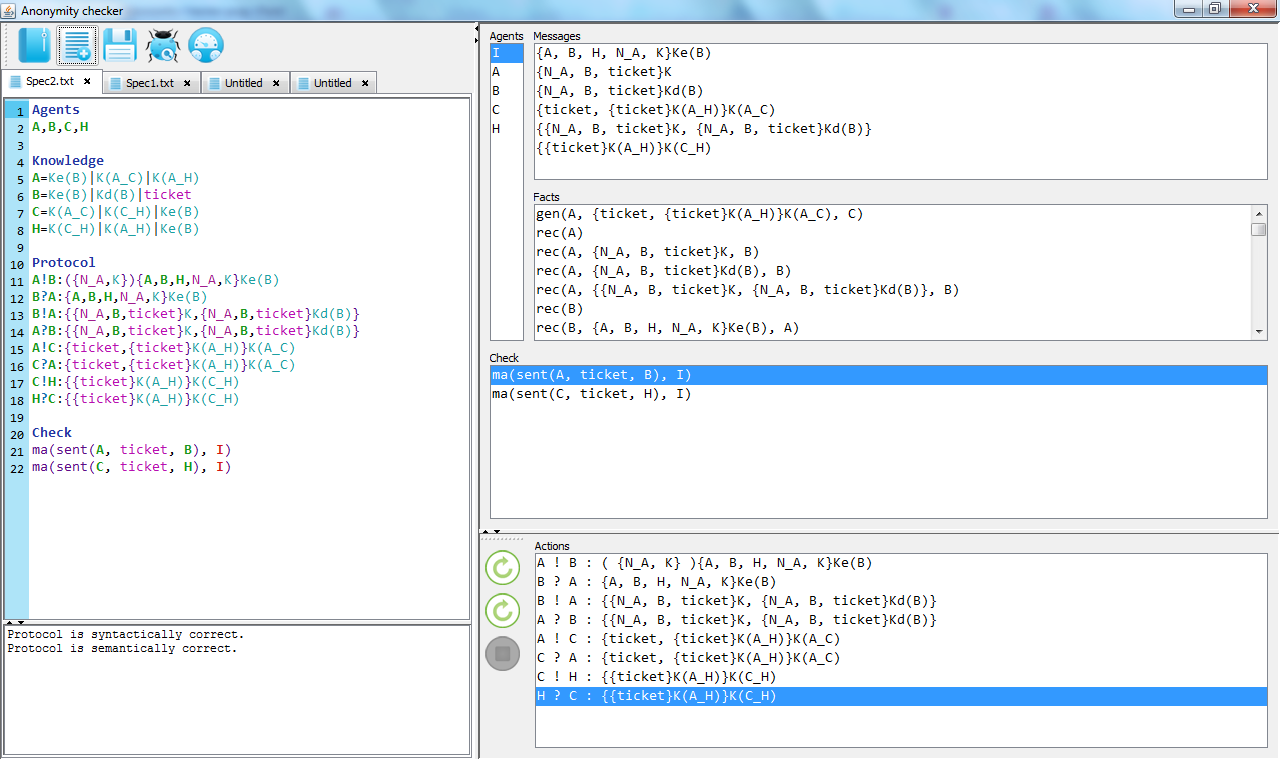
[](http://students.info.uaic.ro/~cristina.serban/pav2014/files/Fig6.png)

Figura 6. Anonymity Checker - Cristina Şerban, 2012. Meniul plasat deasupra

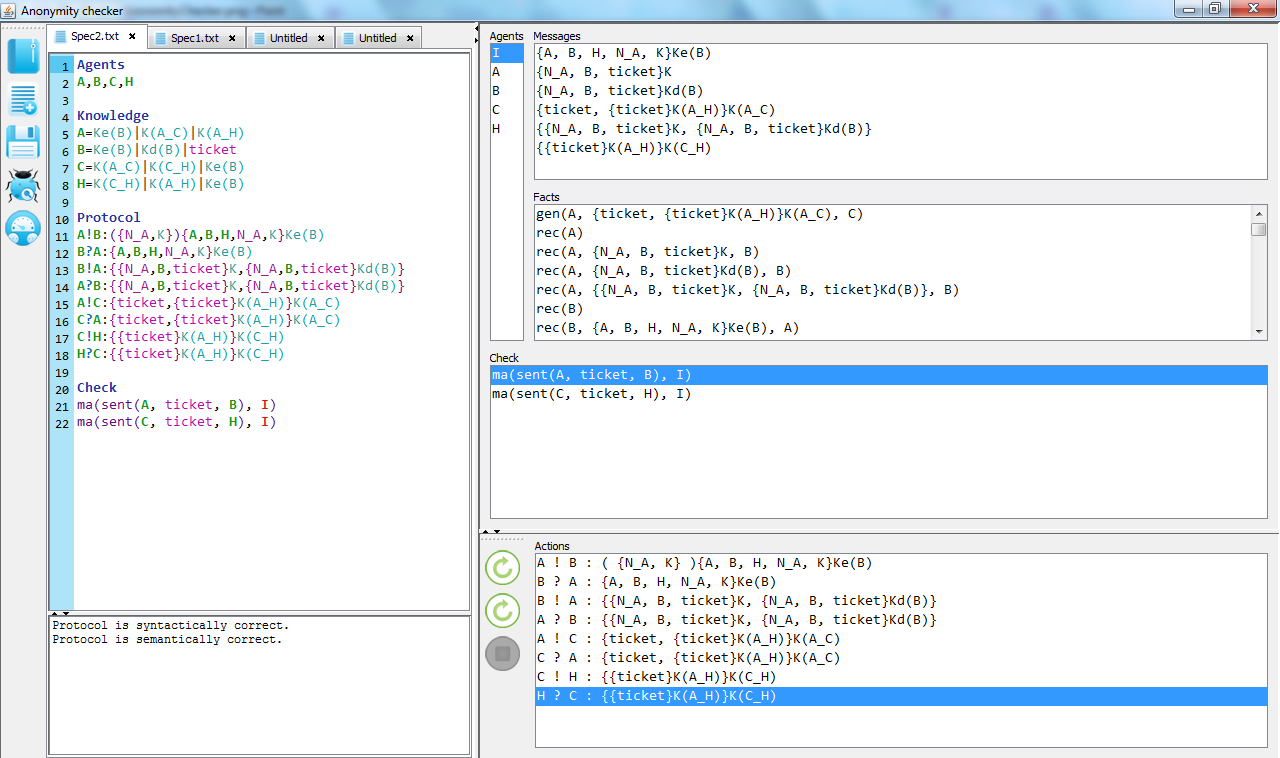
[](http://students.info.uaic.ro/~cristina.serban/pav2014/files/Fig7.png)

Figura 7. Anonymity Checker - Cristina Şerban, 2012. Meniul plasat in partea stângă

### Tips & Tricks - laborator 4

1. Pentru toate temele care cer interfaţă grafică puteți folosi (în loc de Swing):
   1. [SWT](http://www.eclipse.org/swt/) - Principiile de funcţionare sunt asemănătoare. Învăţaţi conceptele din spate şi folosiţi-vă cât mai puţin de mediul grafic pus la dispoziţie de Eclipse - codul generat, în general, are tendinţa să fie haotic.
   2. [JavaFX](http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html) - Deși orientat mult spre grafică, oferă și o suită de controale pentru GUI.
2. Învăţaţi să creaţi elementele de interfaţă din cod, nu cu ajutorul mediului grafic de editare. NetBeans oferă un astfel de mediu grafic, construit cumva peste Swing, dar codul generat în spate este foarte greoi de parcurs, iar comportamentul nu este întotdeauna cel dorit. Învățați să creați componente, să le adăugați în container-e, să adăugați listner-e, să implementați interfețe ce descriu modele.
3. Pentru partea de criptare a răspunsurilor s-ar putea să vă ajute [Java Cryptography Architecture (JCA)](http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html).
4. Pentru folosirea documentelor în format XML consultaţi [Java API for XML Processing (JAXP)](http://docs.oracle.com/javase/tutorial/jaxp/index.html).
5. Se mai poate primi bonus pentru GUI deosebit din punct de vedere al structurii, separare bine făcută între model și view, utilizarea (corectă și nu forțată) a șabloanelor de proiectare (design patterns).

## Antrenament examen

1. Care va fi valoarea variabilei s dupa execuția blocului de instrucțiuni alăturat?

int s = 0;

try {

java.awt.Point p[] = new java.awt.Point[10];

p[0].setLocation(-1, -1);

s += 1;

} catch(IllegalArgumentException e) { s += 2; }

catch(NullPointerException e) { s += 3; }

finally{ s += 4; }

* 1. 0
  2. 1
  3. 5
  4. 6
  5. **7**

1. Ce se va întâmpla la compilarea fișierului Test.java ce conține clasele alăturate?

class A {

double x = 0;

A(double x) { this. x = x; }

A(float x) { this. x = x; }

}

class B extends A {

double y = 0;

B(double x) {

y = 0;

super(x);

}

}

Deoarece la mostenire cand se apeleaza super() acesta tre sa fie pe prima linie din constructor

* 1. Compilarea va fi cu succes.
  2. Compilarea va fi cu succes, dar la execuție instanțierile obiectelor din clasa A pot provoca excepții.
  3. Clasa B nu va fi compilată deoarece nu supradefinește ambii constructori.
  4. **Clasa B nu va fi compilată deoarece constructorul este definit greșit.**
  5. Clasa A nu va fi compilată deoarece al doilea constructor este greșit.

1. Care dintre afirmațiile următoare referitoare la colecții nu este adevărată?
   1. Clasa TreeMap menține sortate elementele sale.
   2. Singura modalitate de a parcurge o mulțime de tip HashSet, fără a folosi alte structuri de date, este prin intermediul iteratorilor.
   3. Clasa Collection este superclasa tuturor claselor ce descriu colecții.
   4. Într-un obiect de tip HashMap, cheia și valoarea sunt de tip Object.
   5. **Orice colecție de elemente poate fi sortată folosind metoda sort() din clasa Collections.**
2. Scrieți metoda contains(Point p, Point p1, Point p2) care să testeze dacă un punct p se găsește pe dreapta determinată de punctele p1 și p2.

public static boolean contains(Point p, Point p1, Point p2) {   
   return p.getY() - p1.getY() ==   
      ((p.getX() - p1.getX()) \* (p2.getY() - p1.getY())) / (p2.getX() - p1.getX());   
}

1. În urma compilării fișierului Test.java a rezultat, pe lângă fișierul Test.class, și fișierul Test$1.class. Care este explicația acestui fapt?
   1. În fișierul Test.java mai există o clasă cu numele Test$1.
   2. Clasa Test conține o subclasă cu numele Test$1.
   3. Clasa Test conține o clasă internă statică cu numele 1.
   4. **Clasa Test conține o clasă anonimă.**
   5. Test$1.class este un fișier temporar creat de către compilator.
2. Care dintre următoarele variante nu este corectă?
   1. **Map a = new Map();**
   2. Map b = new HashMap();
   3. **SortedMap d = new HashMap();**
   4. SortedMap c = new TreeMap();
3. Ce se va întâmpla la compilarea fișierului Stiva.java ce conține clasele alăturate?

class ExceptieStiva extends Exception {}

public class Stiva {

Object v[] = new Object[100];

int top = 0;

public void push(Object x) { v[top] = x; top++; }

public void pop() { if(top == 0) throw new ExceptieStiva(); top--; }

public Object top() { return v[top - 1]; }

}

* 1. Compilarea va fi cu succes și vor rezulta fișierele ExceptieStiva.class și Stiva.class.
  2. Compilarea va fi cu succes, dar la execuție metodele push() și top() pot fi sursele unor excepții netratate.
  3. Clasa ExceptieStiva nu va fi compilată deoarece nu supradefinește constructorul clasei Exception.
  4. Clasa Stiva nu va fi compilată deoarece metodele push() și top() nu tratează excepțiile ce pot apărea.
  5. **Clasa Stiva nu va fi compilată deoarece metoda pop() este incorect definită.**

1. În fișierul A.java sunt următoarele declarații de clase și interfețe:

interface X {}

interface Y {}

public class A implements X {}

public class B extends A implements X, Y {}

Care este motivul pentru care secvența de mai sus nu va fi compilată?

* 1. În Java nu există moștenire multiplă.
  2. Clasa B face o dublă implementare interfețelor.
  3. **În fișier există două clase publice.**
  4. Pentru a putea fi implementate, interfețele trebuie să fie publice.

1. Care va fi valoarea variabilei tip după execuția secvenței alăturate?

int c = (int) 'b';

String tip = "nimic";

switch(c) {

case 'a'-'z': tip = "litera";

case '0'-'9': tip = "cifra"; break;

default: tip = "altceva";

}

Deoarece caracterul b va fi convertit la reprezentarea sa ASCII din spate iar aceasta este un numar de 2 cifre

* 1. "litera"
  2. "cifra"
  3. **"altceva"**
  4. "nimic"

1. Considerându-se declarația "public private int h;" să se precizeze care afirmație este adevărată:
   1. variabila h va fi accesată în mod public deoarece se ia în considerare primul modificator de acces.
   2. variabila h va fi accesată în mod private deoarece se ia în considerare ultimul modificator de acces.
   3. **va fi eroare la compilare deoarece o variabilă nu poate fi în acelaşi timp accesată public şi private.**

# LABORATOR 6: Desenarea (Java 2D). JavaFX

## Java 2D

Majoritatea aspectelor legate de Java 2D API v-au fost explicate la [curs](http://thor.info.uaic.ro/%7Eacf/java/slides/desenare_slide.pdf). Pentru ceva mai multe detalii, consultaţi [tutorialul referitor la Java 2D API](http://docs.oracle.com/javase/tutorial/2d/). Câteva aspecte de interes legate de desenare sunt surprinse în capitolele [Drawing Geometric Primitives](http://docs.oracle.com/javase/tutorial/2d/geometry/primitives.html), [Stroking and Filling Graphics Primitives](http://docs.oracle.com/javase/tutorial/2d/geometry/strokeandfill.html) şi [Working with Images](http://docs.oracle.com/javase/tutorial/2d/images/index.html).

Ca suport pentru desenare, veţi utiliza clasa [Canvas](http://docs.oracle.com/javase/7/docs/api/java/awt/Canvas.html) (de obicei, o clasă derivată din aceasta). Desenarea efectivă se va face în metoda paint(), pe care o suprascrieţi. Pentru a folosi Java 2D, faceţi cast de la obiectul de tip Graphics primit ca parametru la un obiect de tip Graphics2D. Acest obiect reprezintă contextul de desenare.

public void paint (Graphics g) {   
   Graphics2D g2 = (Graphics2D) g;   
   // ...   
}

### Flickering. Double buffering

Metoda update() reactualizează suprafaţa de desenare şi este apletă în urma apelului metodei repaint(). Problema care apare este că în metoda update(), suprafaţa de desenare este ştearsă şi apoi redesenată, cauzând apariţia unui efect de "flickering". Soluţia pentru acestă problemă este prezentată în [curs](http://thor.info.uaic.ro/%7Eacf/java/slides/desenare_slide.pdf), la pagina 19. La aceeşi pagină amintită mai sus este dat şi un exemplu de double buffering.

### Antialiasing

"In computer graphics, antialiasing is a software technique for diminishing jaggies - stairstep-like lines that should be smooth. Jaggies occur because the output device, the monitor or printer, doesn't have a high enough resolution to represent a smooth line. Antialiasing reduces the prominence of jaggies by surrounding the stairsteps with intermediate shades of gray (for gray-scaling devices) or color (for color devices). Although this reduces the jagged appearance of the lines, it also makes them fuzzier" (sursa: [Webopedia](http://www.webopedia.com/TERM/A/antialiasing.html)).

public void paint (Graphics g) {   
   Graphics2D g2 = (Graphics2D) g;   
   g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,   
      RenderingHints.VALUE\_ANTIALIAS\_ON);   
   // ...   
}

## JavaFX

"The JavaFX platform is the evolution of the Java client platform designed to enable application developers to easily create and deploy rich internet applications (RIAs) that behave consistently across multiple platforms. Built on Java technology, the JavaFX platform provides a rich set of graphics and media API with high-performance hardware-accelerated graphics and media engines that simplify development of data-driven enterprise client applications."

Pentru mai multe informaţii şi tutoriale despre cum să utilizaţi JavaFX, consultaţi [documentaţia](http://docs.oracle.com/javafx/index.html). Pentru ceea ce aveţi nevoie în cadrul laboratorului, puteţi începe cu [Working with Canvas](http://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm), [Applying Transformations in JavaFX](http://docs.oracle.com/javafx/2/transformations/jfxpub-transformations.htm) şi [Creating Transitions and Timeline Animation in JavaFX](http://docs.oracle.com/javafx/2/animations/jfxpub-animations.htm).

## Tips & Tricks - laborator 6

1. Pentru a urmări acţiunile mouse-ului, utilizaţi interfaţa [MouseInputListener](http://docs.oracle.com/javase/7/docs/api/javax/swing/event/MouseInputListener.html).
2. Atunci când trasaţi o figură, afişaţi toate formele intermediare pe care aceasta le ia atunci când se mişcă mouse-ul, fără a lăsa rămăşiţe de la desenări anterioare. Acest proces începe în momentul când butonul mouse-ului este apăsat şi se termină atunci când acesta este eliberat.
3. Eliminaţi efectul de "flickering", aşa cum este prezentat în curs.
4. Folosiţi tehnica de double buffering, aşa cum este descrisă în curs.
5. Pentru a înţelege mai uşor diferenţa între modul de desenare "imediat" şi cel "scenă", faceţi analogia cu programe pe care le cunoasteti deja, spre exemplu Paint şi Photoshop.
   * Paint - Toate elementele deja desenate devin o singură imagine indivizibilă. Există control doar asupra celei mai recente figuri desenate.
   * Photoshop - componentele sunt separate unele de altele, eventual alcătuite din subcomponente, rezultând o scenă arborescentă.
6. Pentru a putea comuta uşor între cele două moduri de desenare este necesar un nivel de abstractizare. Gândiţi-vă la noţiuni discutate în laboratoarele anterioare care v-ar putea ajuta.

## Antrenament examen

1. Ce se va întâmpla la compilarea următorului program:

public class SubiectLicentaUnu {

int i = 3, j = 4;

public static void main(String [] arg) {

System.out.println(i ++ + ++ j);

}

}

* 1. va apărea eroare la compilare deoarece parametrul metodei main() trebuie să fie String args[]
  2. **va apărea eroare la compilare deoarece dun funcția main() nu putem accesa variabilele nestatice i și j**
  3. va apărea eroare la compilare deoarece nu s-au inserat paranteze între operatorii ++, adică (i++) și (++j)
  4. se va afișa 8
  5. se va afișa 9

1. Ce se va afișa în cadrul următorului program (liniile sunt numerotate):

1. public class SubiectLicentaDoi {

2. static int x = 10;

3. static { x += 5; }

4. public static void main(String args[]) {

5. System.out.println("x = " + x);

6. }

7. static { x /= 5; }

8. }

Deoarece fiecare modificare a variabilei x se realizeaza din context static

* 1. liniile 3 și 7 nu se vor compila deoarece lipsesc numele metodelor și tipurile returnate
  2. linia 7 nu se va compila deoarece poate exista doar un inițializator static
  3. codul se compilează și execuția produce ieșirea x = 15
  4. codul se compilează și execuția produce ieșirea x = 10
  5. **codul se compilează și execuția produce ieșirea x = 3**

1. Ce se afișează în urma execuției următorului program?

public class SubiectLicentaTrei {

public static void main(String args[]) {

double d = 12.34;

ScadeUnitate su = new ScadeUnitate();

su.scadeUnitate(d);

System.out.println(d);

}

}

class ScadeUnitate {

public void scadeUnitate(double d) {

d = d - 1.0;

}

}

Deoarece metoda scadeUnitate desi scade 1.0 din valoarea lui d se face practic doar pe instanta si atat

* 1. 0.0
  2. -1.0
  3. **12.34**
  4. 11.34
  5. eroare la compilare deoarece lipsește constructorul clasei ScadeUnitate

1. Ce rezultat obținem la execuția următorului cod Java:

int x = 1;

String[] sir = {"Test", " Licenta", " JAVA"};

sir[--x] += ".";

for (int i = 0; i < sir.length; i++)

System.out.print(sir[i]);

* 1. șirul: Test.
  2. șirul: Test Licenta. JAVA
  3. șirul: Tes
  4. **șirul: Test. Licenta JAVA**
  5. o excepție de tip ArrayIndexOutOfBoundsException

1. Considerându-se secvența de cod alăturată, să se precizeze care afirmație este adevărată:

i) float a = 2;

ii) float b = 1.3;

* 1. instrucțiunile i) și ii) sunt corecte
  2. **instrucțiunea i) este corectă, instrucțiunea ii) este incorectă**
  3. instrucțiunea i) este incorectă, instrucțiunea ii) corectă
  4. instrucțiunile i) și ii) sunt incorecte

1. Ce puteți spune despre următorul subprogram Java?

String s1 = "anul" + 200 + 2,

s2 = 200 + 2 + "anul";

System.out.println("s1 = " + s1 + ", s2 = " + s2);

Deoarece operatorul + ia de la stanga la dreapta si prima data incearca sa fie operatorul de adunare si apoi cel de concatenare

* 1. s1 = anul202, s2 = 202anul
  2. **s1 = anul2002, s2 = 202anul**
  3. s1 = anul202, s2 = 2002anul
  4. s1 = anul2002, s2 = 2002anul
  5. Eroare la execuție: este necesară o conversie explicită!

1. Ce puteți spune despre programul alăturat? (liniile sunt numerotate)

1. public class TestTablouri {

2. public static void main(String[]args) {

3. int[] a = {2, 6};

4. Object b = a;

5. System.out.print(b[0] + " ");

6. byte[] c = new byte[2];

7. c = a;

8. System.out.print(c[1] + " ");

9. }

10. }

* 1. programul este corect și va afișa la execuție: 2 6
  2. va apărea eroare de compilare doar la linia 4 deoarece b necesită o conversie explicită
  3. va apărea eroare de compilare doar la linia 5 deoarece b[0] nu poate fi accesat
  4. **vor apărea erori de compilare la linia 5 deoarece b[0] nu poate fi accesat și la linia 7 deoarece tipul componentelor tablourilor a și c nu sunt compatibile**
  5. va apărea eroare la execuție din cauza liniei 5, aruncându-se excepția ClassCastException

1. Ce se poate spune despre programul alăturat?

public class Apel {

static int f(short a) { return a \* 2; }

public static void main(String[] args) {

System.out.println(f(3));

}

}

* 1. **Eroare la compilare: nu există o definiție a metodei f() cu parametrul int**
  2. Eroare la compilare: metoda f() nu întoarce o valoare de tip short
  3. Eroare la compilare: metoda statică f() trebuie apelată astfel: Apel.f(3)
  4. Programul se compilează și la execuție afișează 6

1. Ce se va afișa în urma execuției secvenței alăturate?

String s1 = new String("hello"),

s2 = "Hello".toLowerCase();

System.out.println(s1 == s2);

* 1. Secvența nu va fi compilată deoarece variabila s2 nu este instanțiată
  2. Secvența nu va fi compilată deoarece șirurile nu pot fi comparate cu ==
  3. **false, deoarece s1 și s2 au adrese de memorie diferite**
  4. true, deoarece toLowerCase convertește un șir la litere mici
  5. true, deoarece comparația șirurilor nu ține cont de litere mici sau mari

1. Care dintre următoarele afirmații referitoare la cele 4 linii numerotate ale metodei main este corectă?

public class Test {

public static String sayHello() {

return "Hello";

}

public static void main(String[] args) {

Test t = new Test();

String a = Test.sayHello(); // 1

t.sayHello().toUpperCase(); // 2

System.out.println(sayHello()); // 3

args[0] = t.sayHello(); // 4

}

}

* 1. Totul este corect, clasa va fi compilată
  2. Linia 1 este greșită, deoarece metodele de clasă nu pot fi apelate astfel
  3. Linia 2 este greșită, deoarece metoda sayHello returnează un șir de caractere
  4. Linia 3 este greșită, deoarece metoda sayHello este statică
  5. **Linia 4 este greșită, deoarece este posibil ca vectorul args să nu aibă elemente**

**LABORATOR 7: Fire de execuţie**

## Crearea şi pornirea firelor de execuţie

public class MyThread extends Thread {   
   public void run() {   
      //...   
   }   
}

new MyThread().start();

public class MyRunnable implements Runnable {   
   public void run() {   
      //...   
   }   
}

new Thread(new MyRunnable()).start();

## Ciclul de viaţă al unui thread

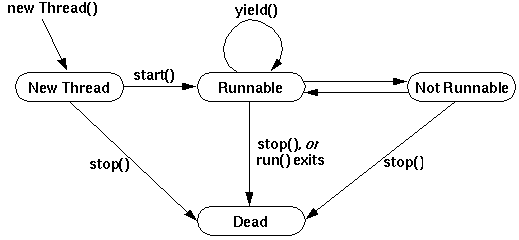


Figura 7. Ciclul de viaţă al unui thread

## Sincronizare

Un segment de cod ce gestionează o resursă comună mai multor de fire de execuţie separate şi concurente se numeşte secţiune critică. În Java o secţiune critică poate fi un bloc de instrucţiuni sau o metodă. Controlul accesului într-o secţiune critică se face prin cuvântul cheie synchronized.

public synchronized void metoda() { //... }

public void metoda() {   
   //...   
   synchronized {   
       //...   
   }   
   //...   
}

### Tips & Tricks - laborator 7

1. Un exemplu interesat de vizualizare grafică sugestivă a execuţiei mai multor algoritmi de sortare este <http://www.sorting-algorithms.com/>. Pentru bonusul de la prima problemă, realizaţi ceva asemănător. E de ajuns să fie reprezentaţi doar trei algoritmi, iar vectorul de sortat trebuie să fie acelaşi pentru toţi algoritmii şi poate fi generat aleator.
2. La a doua problemă, numărul de iepuraşi care vor concura va fi parametrizat şi ales de utilizator înaintea începerii concursului. Cursa se va termina după ce toţi iepuraşii au ajuns la sfârşit. În momentul când un iepuraş ajunge la finalul cursei, i se va afişa locul pe care a terminat.
3. Orice ar fi, nu folosiţi metoda stop() pentru oprirea unui fir de execuţie.
4. Folosiţi cuvântul cheie synchronized atunci când un câmp sau o metodă sunt accesate din mai multe fire de execuţie.
5. Atenţie la priorităţile pe care le acordaţi firelor de execuţie. Dacă este necesar, folosiţi metoda yield().

# LABORATOR 8:Programarea în rețea

## Noţiuni generale

Programarea în retea implică trimiterea de mesaje şi date între aplicaţii ce rulează pe calculatoare aflate într-o reţea locală sau conectate la Internet. Pachetul care oferă suport pentru scrierea aplicaţiilor de retea este java.net. Clasele din acest pachet oferă o modalitate facilă de programare în reţea, fără a fi nevoie de cunoştinţe prealabile referitoare la comunicarea efectivă între calculatoare. Cu toate acestea, sunt necesare câteva noţiuni fundamentale referitoare la reţele, cum ar fi protocol, adresa IP, port, socket.

Protocol - convenţie de reprezentare a datelor folosită în comunicarea între două calculatoare.

Adresă IP - Orice calculator gazdă conectat la Internet este identificat în mod unic de adresa sa IP (IP este acronimul de la Internet Protocol). Aceasta este un număr reprezentat pe 32 de biţi, uzual sub forma a 4 octeţi, cum ar fi, de exemplu, 193.226.26.231, şi este numit adresă IP numerică. Corespunzatoare unei adrese numerice există şi o adresă IP simbolică, cum ar fi fenrir.infoiasi.ro.

Port - un număr de 16 biţi care identifică în mod unic procesele care rulează pe o anumită maşină.

Socket - o abstracţiune software folosită pentru a reprezenta fiecare din cele două "capete" ale unei conexiuni între două procese ce rulează într-o retea. Socket-urile sunt de două tipuri:

* TCP, implementate de clasele Socket şi ServerSocket
* UDP, implementate de clasa DatagramSocket

Găsiţi în curs exemple de cod:

* [Comunicarea prin conexiuni](http://thor.info.uaic.ro/%7Eacf/java/curs/8/prog_retea.html#conexiuni)
* [Comunicarea prin datagrame](http://thor.info.uaic.ro/%7Eacf/java/curs/8/prog_retea.html#datagrame)

### Tips & Tricks - laborator 8

1. Pentru prima problemă, serverul va putea gestiona mai multe partide simultan, în fire de execuţie diferite. Consideraţi ca unitate de lucru o pereche de clienţi care joacă împreună.
2. Aşa cum este menţionat şi în cerinţă, memoraţi starea tablei de joc la nivel de server. Atunci când vine rândul unui jucător, acesta va primi starea actualizată a tablei de joc.
3. Ambele probleme trebuie să aibă interfaţă grafică cu utilizatorul.
4. La a doua problemă este de dorit ca site-ul copiat să fie navigabil offline.

## Antrenament examen

1. Să considerăm declarația A a = new A(); unde clasa A este definită mai jos:

class A implements Runnable {

int counter = 0;

public void run() { while (true) counter++; }

}

Care dintre variantele de mai jos instanțiază și lanseaza un fir de execuție?

* 1. a.run();
  2. a.start();
  3. new Thread(a).run();
  4. **new Thread(a).start();**

1. Care din afirmațiile următoare referitoare la fire de execuție în Java sunt adevărate?
   1. **Orice fir de execuție este o instanță a clasei Thread.**
   2. Orice clasă care descrie un fir de execuție trebuie să implementeze obligatoriu interfața Runnable.
   3. Pentru a controla accesul mai multor fire de execuție la o resursă comună, trebuie să declarăm clasa ce descrie resursa respectivă folosind modificatorul synchronized .
   4. **Un fir de execuție poate să cedeze procesorul altor fire de execuție cu aceeași prioritate, chiar dacă nu și-a terminat încă activitatea** (vezi [Thread Priorities, yield() and Join](http://inheritingjava.blogspot.ro/2011/02/chapter-59-thread-priorities-yield-and.html)).
   5. Singura modalitate de a opri un fir de execuție este prin apelul metodei stop().
2. Ce se va afișa la execuția următorului program?

1. public class Exemplu {

2. public static void main(String args[]) {

3. A.B a = new A().new B();

4. a.f();

5. }

6.}

7. class A {

8. private int x = 4;

9. public class B {

10. private int y = 5;

11. public void f() {

12. System.out.println("x = " + x);

13. x = 3;

14. System.out.println("x = " + x + ", y = " + y);

15. }

16. }

17.}

* 1. Eroare de compilare la linia 3 deoarece construcția A.B nu este validă.
  2. Eroare de compilare la linia 4 deoarece metoda f() nu aparține clasei A.
  3. Eroare de compilare la linia 9 deoarece nu se poate declara o clasă în interiorul altei clase.
  4. Eroare de compilare la linia 12 deoarece nu se poate accesa data privată x.
  5. **Se va afișa:   
     x = 4   
     x = 3, y = 5**

1. Scrieți o metodă care primeste ca argumente o colecție de numere întregi v precum și un alt întreg a. Metoda va elimina din colecție toate valorile strict mai mici decât a și va returna numărul de eliminări efectuate. Metoda poate să accepte atât colecții de tip ArrayList, cât și LinkedList.

Se punctează signatura funcției (tip returnat, parametri - în special cel de tip List<Integer>), parcurgerea listei fără folosirea iteratorilor (de ce?), compararea elementelor, eliminarea elementelor și numărarea eliminărilor.

public static int elimina(List<Integer> v, int a) {   
   int count = 0;   
  
   for(int i = 0; i < v.size(); i++) {   
      if(v.get(i).compareTo(a) < 0) {   
         v.remove(i);   
         i--;   
         count++;   
      }   
   }   
  
   return count;   
}

1. Care dintre următoarele afirmații sunt adevărate relativ la codul de mai jos?

1. public class Exemplu extends Thread {

2. public void run() {

3. System.out.print("Ultima ");

4. yield();

5. System.out.print("intrebare");

6. }

7. public static void main (String args[]) {

8. Exemplu e = new Exemplu();

9. e.start();

10. }

11.}

* 1. Compilarea va eșua la linia 4 deoarece yield() se poate apela doar în cod sincronizat.
  2. Compilarea va eșua la linia 4 deoarece yield() se poate apela doar într-un bloc try/catch.
  3. Compilarea se va face cu succes. La execuție, nu se va afișa nimic.
  4. Compilarea se va face cu succes. La execuție, se va afișa Ultima.
  5. **Compilarea se va face cu succes. La execuție, se va afișa Ultima intrebare.**

**LABORATOR 9: Applet-uri. Internaţionalizare şi localizare**

## Applet-uri

Un **applet** reprezinta o suprafata de afisare (container) ce poate fi inclusa într-o pagina Web si gestionata printr-un program Java. Un astfel de program se mai numeste miniaplicatie sau, prin abuz de limbaj, applet.

Codul unui applet poate fi format din una sau mai multe clase. Una dintre acestea este principala si extinde clasa Applet (pentru Swing, vom folosi JApplet), fiind clasa ce trebuie specificata în documentul HTML ce descrie pagina de Web în care dorim sa includem appletul.

Diferenta fundamentala dintre un applet si o aplicatie consta în faptul ca, un applet nu poate fi executat independent, ci va fi **executat de browserul** în care este încarcata pagina Web ce contine appletul respectiv. O aplicatie independenta este executata prin apelul interpretorului java, având ca parametru numele clasei principale a aplicatiei, clasa principala fiind cea care contine metoda main. Ciclul de viata al unui applet este complet diferit, fiind dictat de evenimentele generate de catre browser la vizualizarea documentului HTML ce contine appletul.

Pachetul care ofera suport pentru creearea de appleturi este java.applet.

## Internaţionalizare

**Internaţionalizarea** - pregătirea unei aplicaţii astfel încât să poată fi personalizată în funcţie de o regiune geografică, politică sau culturală.

**Localizarea** - adaptarea unei aplicaţii internaţionalizate la o anumită regiune.

### Tips & Tricks - laborator 9

1. Puteți folosi biblioteci care implementează lucrul cu polinoame, dar trebuie să știți să explicați care este principiul de funcționare.
2. Consultați tutorialul Java pentru exemple de [comunicare între două applet-uri](http://docs.oracle.com/javase/tutorial/deployment/applet/iac.html).
3. Pentru a obține lista de instanțe de tip Locale disponibile, aruncați o privire asupra metodelor statice din clasa [Locale](http://docs.oracle.com/javase/7/docs/api/java/util/Locale.html).
4. Pentru a obține lista de instanțe de tip TimeZone disponibile, aruncați o privire asupra metodelor statice din clasa [TimeZone](http://docs.oracle.com/javase/7/docs/api/java/util/TimeZone.html).
5. Pentru a obţine denumirile de zile şi de luni pentru un anumit Locale, folosiţi clasa [DateFormatSymbols](http://docs.oracle.com/javase/7/docs/api/java/text/DateFormatSymbols.html).
6. Pentru a obţine denumirea monedei utilizate într-un anumit Locale, folosiţi clasa [Currency](http://docs.oracle.com/javase/7/docs/api/java/util/Currency.html).

**LABORATOR 10: Lucrul cu baze de date în Java**

## JDBC

Accesul la o bază de date se face prin intermediul unui driver specific tipului respectiv de bază de date. Acesta este responsabil cu accesul efectiv la datele stocate, fiind legătura între aplicație și baza de date.

JDBC (Java Database Connectivity) este o interfață standard SQL de acces la baze de date. JDBC este constituită dintr-un set de clase și interfețe scrise în Java, furnizând mecanisme standard pentru proiectanții aplicațiilor de baze de date. Pachetul care ofera suport pentru lucrul cu baze de date este java.sql.

În linii mari, JDBC face trei lucruri:

* stabilește o conexiune cu o bază de date;
* trimite secvențe SQL;
* prelucrează rezultatele.

O conexiune (sesiune) la o bază de date reprezintă un context prin care sunt trimise secvențe SQL și primite rezultate. Într-o aplicație pot exista mai multe conexiuni simultan la baze de date diferite sau la aceeași bază.

### Tips & Tricks - laborator 10

1. Urmăriți şi acest [tutorial](http://www.youtube.com/watch?v=Fq7ZsuQZM_w) despre crearea de rapoarte cu JasperReports|iReport prin intermediul unei conexiuni JDBC.
2. Pentru mai multe exemple și explicații, consultați [tutorialul Java despre JDBC](http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html).
3. Pentru extragerea de metadate, v-ar putea fi de folos următoarele materiale:
   * [Database Metadata with JDBC](http://www.codebeach.com/2008/12/database-metadata-with-jdbc.html);
   * [JDBC: DatabaseMetaData](http://tutorials.jenkov.com/jdbc/databasemetadata.html)
   * [Understanding JDBC Metadata;](http://eye-on-objects.com/c_brown.htm)
   * Documentația pentru [interfața DatabaseMetaData](http://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html);

**LABORATOR 11: Lucrul dinamic cu clase. Java Web Start**

### Tips & Tricks - laborator 11

1. Puteți folosi și altceva decât Swing, atâta vreme cât sunt implementate cerințele problemei (i.e. bară de componente, listare de proprietăți, schimbare valori proprietăți, salvare și restaurare în/din format XML).
2. Accesarea și modificarea proprietăților componentelor se va face folosind reflection.
3. Pentru scrierea și citirea fișierelor XML folosiți [XMLEncoder](http://docs.oracle.com/javase/7/docs/api/java/beans/XMLEncoder.html) și [XMLDecoder](http://docs.oracle.com/javase/7/docs/api/java/beans/XMLDecoder.html).
4. Consultați [The Reflection API](http://docs.oracle.com/javase/tutorial/reflect/), [Java Web Start](http://docs.oracle.com/javase/tutorial/deployment/webstart/), [JNLP API](http://docs.oracle.com/javase/tutorial/deployment/doingMoreWithRIA/jnlpAPI.html).

## Antrenament examen

1. Ce definește structura de mai jos? Descrieți o modalitate de utilizare a ei.

@Target(ElementType.FIELD)

public @interface NotNull { }

Structura definește o adnotare care poate fi folosită pe câmpuri. Dacă aveți astfel de întrebări, dați și exemple!   
  
@NotNull   
int x;

1. Implementați metoda drawShape care primește ca argument un String ce reprezintă numele unei clase ce implementează interfața java.awt.Shape și un obiect de tip java.awt.Graphics2D, funcționalitatea acesteia fiind desenarea respectivei forme geometrice (implicite), folosind metoda Graphics2D.draw().

Se punctează:

* + Signatura metodei: void drawShape(String c, Graphics2D g)
  + Folosirea mecanismului de reflection: z = Class.forName(c).newInstance();
  + Desenarea: g.draw((Shape)z);

1. Care dintre următoarele afirmații legate de applet-uri nu sunt corecte?
   * Un applet poate fi descărcat de pe un server Web sub formă de arhivă jar.
   * **Execuția unui applet începe întotdeauna cu metoda main().**
   * Applet-urile pot accesa fișiere de pe mașina client prin JNLP API.

[JNLP API](http://docs.oracle.com/javase/tutorial/deployment/doingMoreWithRIA/jnlpAPI.html), [Accessing the Client Using JNLP API](http://docs.oracle.com/javase/tutorial/deployment/doingMoreWithRIA/usingJNLPAPI.html)

* 1. Într-un applet pot fi folosite mai multe fire de execuție.

1. Fie a un obiect de tip java.awt.Applet ce implementează interfața Runnable. Care dintre următoarele variante va determina pornirea unui fir de execuție?
   * a.start();
   * new Thread(a).run();
   * **new Thread(a).start();**
   * Thread.start(a);
2. Care informații referitoare la applet-uri sunt corecte?
3. **Conțin obligatoriu o clasă ce extinde java.applet.Applet.**
4. **Pot fi formate din mai multe clase.**
5. **Clasele unui applet pot fi grupate în mai multe pachete.**
6. Folosesc cel puțin două fire de execuție.
7. Care din următoarele afirmații referitoare la applet-uri sunt adevărate?
8. **Trebuie obligatoriu să extindă clasa java.applet.Applet**
9. Trebuie obligatoriu să supradefinească metodele init(), start(), stop(), destroy().
10. **Clasele unui applet pot fi grupate în mai multe pachete.**
11. Un applet trebuie să definească cel puțin două fire de execuție.
12. Dacă applet-ul nu se găsește într-o arhivă .jar, clasele sale sunt transferate una câte una în conexiuni HTTP diferite.
13. Ce se poate spune despre programul alăturat?

public class Program {

static int x = 6;

public static void main(String[] args) {

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

int x = (x = 3) \* x;

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

}

}

* + Eroare la compilare: variabila x este declarată de două ori.
  + Programul se compilează și la execuție afișează x = 6, x = 3.
  + **Programul se compilează și la execuție afișează x = 6, x = 9.**
  + Programul se compilează și la execuție afișează x = 6, x = 18.

1. Care va fi valoarea variabilei counter după execuția blocului de instrucțiuni alăturat?

int counter = 0;

try {

Object obj[] = new Object[10]; counter ++;

obj[0] = new String("hello"); counter ++;

counter ++;

} catch (Exception e) { counter ++; }

finally { counter ++; }

1. 1
2. 2
3. 3
4. **4**
5. 5
6. Care vor fi erorile depistate la compilarea clasei alăturate?

class A {

private int x=0;

protected static int y=0;

public void f() { x=1; y=1; }

static void g() { x=2; y=2; }

}

1. **Variabila x nu poate fi apelată dintr-un context static**
2. Variabila x nu poate fi apelată dintr-un context ne-static
3. Variabila y nu poate fi apelată dintr-un context static
4. Variabila y nu poate fi apelată dintr-un context ne-static
5. Metoda g nu are modificator de acces

**LABORATOR 12: Introducere în Java ME**

### Tips & Tricks - laborator 12

1. Se punctează orice implementare (sau tentativă de implementare) care conține elementele cerute (ecran structurat, ecran grafic, persistență).
2. Tutoriale Netbeans: [Java ME MIDP Development Quick Start Guide](https://netbeans.org/kb/72/javame/quickstart.html), [Visual Mobile Designer Custom Components: Creating Login Screens](https://netbeans.org/kb/71/javame/loginscreen.html).
3. Tutoriale Eclipse: [Using the Oracle Java ME SDK Software with Eclipse](http://docs.oracle.com/javame/config/cldc/rel/3.3/win/gs/html/getstart_win32/setup_eclipseenv.htm), [Creating a new J2ME MIDlet Project](http://eclipseme.org/docs/createProject.html).
4. Tutorial IntelliJ: [IntelliJ IDEA 13.1 Web Help - J2ME](http://www.jetbrains.com/idea/webhelp/j2me.html).