Advanced Programming Methods

Lecture 1

Grading

Laboratory:--50%

- Assignments at each lab (about 10 lab assigments) -- 35%
- First Practical Test Exam at LAB5 (1 hour, open book): -- 5%
- Second Practical Test Exam at LAB11 or LAB12 or LAB13 (2 hours, open book): --10%

Final exam: -- 50%

- Written Exam (about 1 hour, closed books): --20%
- Practical Exam (about 4 hours, one problem in Java, one problem in C#, open books): --30%

Seminar:

- your contribution will be taken into account for the rounding of the final mark

Rules

- you have to present each assignment at its deadline
- each week of the assignment delay means -1points from that assigment mark
- if you delay an assignment more than 4 weeks you will get the grade 1 for it
- if you will not come to a lab practical exam you will get 1 for that exam
- in order to get into the final exam your average grade for the lab activities (assigments + 2 lab practical tests) must be at least 5
- in order to pass the final exam you must have at least 5 at the final written exam and at least 5 at the final practical exam such that the final grade must be at least 5
- you can pass either both the final written exam and the final practical exam or nothing

Rules for the second exam (“restanta”)

- in order to pass the final second exam you must have at least 5 at the final written exam and at least 5 at the final practical exam

- you can pass either both the second final written exam and the second final practical exam or nothing

- at the second exam “restanta” you can present or re-present maximum 5 lab assigments in order to improve your lab activities mark

Course information

http://www.cs.ubbcluj.ro/~craciunf/MetodeAvansateDeProgramare/InfoEng

Lectures Overview

Object-oriented languages:

- Java
- C#

Lecture 1 and Lecture 2 (Lab 1, Lab2, and Sem1):
Java basics

Lecture 3 and Lecture 4 (Lab 3, Lab 4, and Sem2):
C# basics

After 4 lectures (and labs) :

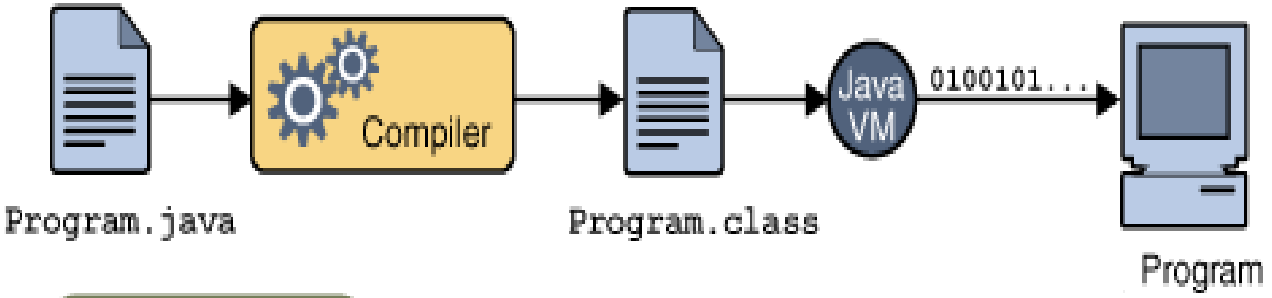
- practical exam at the lab, you will be able to write simple OO programs in Java and C#

Lectures 1+2 Bibliografy

- Bruce Eckel, *Thinking in Java*

- Java Tutorials from Oracle: http://docs.oracle.com/javase/tutorial/

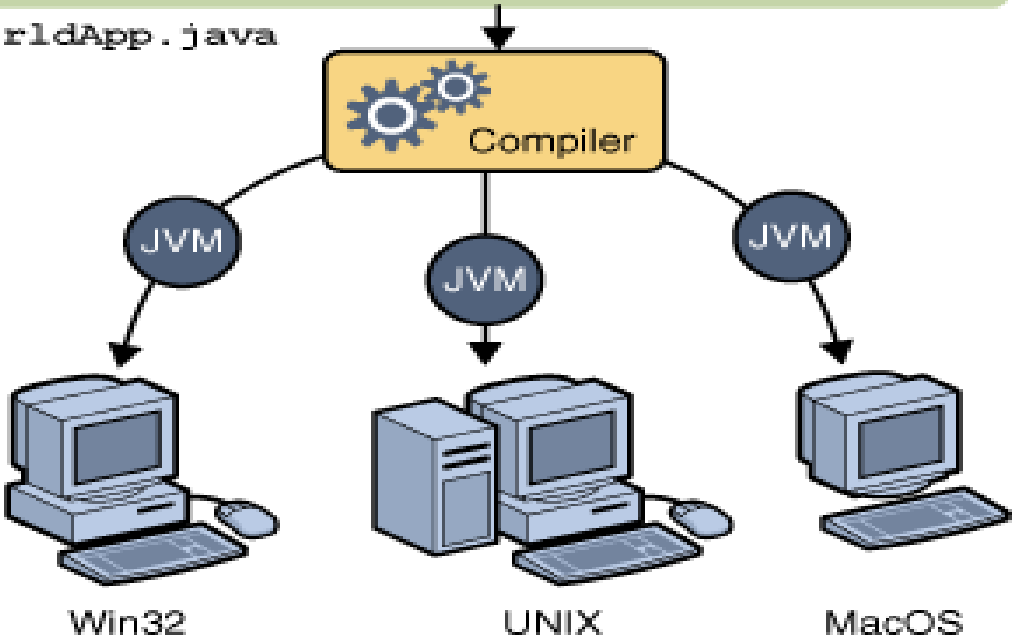
Java Technology



Java Program

```
class HelloWorldApp {
      public static void main(String[] args) {
            System.out.println("Hello World!");
      }
}
```

HelloWorldApp.java



Java language

Variables

Declaration:

```
type name_var1[=expr1][, name_var2[=expr2]…];
```

Primitive Data Types

Type	Nr. byte	Values	Default value
boolean	-	true, false	false
byte	1	-128 … +127	(byte)0
short	2	-2^{15} …$2^{15}-1$	(short)0
int	4	-2^{31} …$2^{31}-1$	0
long	8	-2^{63} …$2^{63}-1$	0L
float	4	IEEE754	0.0f
double	8	IEEE754	0.0d
char	2	Unicode 0, Unicode $2^{16}-1$	‘\u0000’ (null)

Java Language

Examples:

```
boolean gasit=true;
int numar=34, suma;
float eroare=0.45;
char litera='c';
litera='f';
litera=litera+1;
```

Java Comments

1. `// entire line`
2. `/* multiple`
   `lines */`
3. `/** used by documentation Javadoc tool`
   `multiple`
   `lines*/`

Obs: Comments canot be nested into strings.

```
/* … /*
*/ … */   NOT OK!
```

```
/* …
   //
   //
*/   OK!!
```

Java Constants

```
final type name [=value];
```

Examples:

a)
```
final int MAX=100;
```

b)
```
final int MAX;
…
MAX=100;
...
MAX=150; //error
```

Array

Array with one dimension

```
type[] name;
type name[];
```

Array allocation:

```
array_name=new type[dim]; //memory allocation
    //index  0 … dim-1
```

Accessing an array element: array_name`[index]`

Examples:

```
float[] vec;
vec=new float[10];
int[] sir=new int[3];
float tmp[];
tmp=vec;      //vec and tmp refer to the same array
```

One dimension Array

Built-in length: returns the array dimension.

```
int[] sir=new int[5];
int lung_sir=sir.length;  //lung=5;
sir[0]=sir.length;
sir.length=2;     //error

int[] y;
int lung_y=y.length;  //error, y was not created

double[] t=new double[0];
int lung_t=t.length;  //lung_t=0;
t[0]=2.3       //error: index out of bounds
```

the shortcut syntax to create and initialize an array:

```
int[] dx={-1,0, 1};
```

Rectangular multidimensional array

Declaration:

```
type name[][][]…[];
type[][][]…[] name;
```

Creation:

```
name=new type[dim1][dim2]…[dimn];
```

Accessing an element:

```
name[index1][index2]…[indexn];
```

Examples:

```
int[][] a;
a=new int[5][5];
a[0][0]=2;
int x=a[2][2];   //x=?
```

Non-Rectangular Multidimensional Array

Examples:

```
int[][] a=new int[3][];
for(int i=0;i<3;i++)
    a[i]=new int[i+1];
int x=a.length;      //x=?
int y=a[2].length;   //y=?
```

Declaration+creation+initialization:

```
char[][] lit={{'a'},{'b'}};
int[][] b={{1,2},
           {2,5,8},
            {1}};
double[][] mat=new double[][]{{1.3, 0.5}, {2.3, 4.5}};
```

Char and String

```
char[] sir={'a','n','a'}; //comparison and printing
                          //is done character by character
sir[0]='A';
```

A constant Sequence of Chars :

```
"Ana are mere";           //object of type String
```

String class is immutable:

```
String s="abc";
s=s+"123";          //concatenating strings
String t=s;//t="abc123"
t+="12";            //t=?, s=?
```

String content can not be changed: `t[0]='A';`

```
char c=t.charAt(0);
```

```
method length(): int lun=s.length();
t.equals(s)
/*Returns true if and only if the argument is a String object that
represents the same sequence of characters as this object. */
compareTo(): int rez=t.compareTo(s)
/*Compares two strings lexicographically. Returns an integer indicating
whether this string is greater than (result is > 0), equal to (result is =
0), or less than (result is < 0) the argument.*/
```

Operators

arihtmetic: `+, -, *, /, %`

relational: `>, >=, <, <=, !=, ==`

increment/decrement: `++, --`

```
prefix:  int a=2;
     int b=++a; //a=2, b=2
postfix: int a=2;
     int b=a++; //a=3, b=2
```

assignment: `=, +=, -=, *=, /=`

conditional: `&&, ||, !`

`bitwise:` - shift `>>, <<, >>>,`

- conditional `&, |, ~ (not), ^ (exclusive or)`

ternary operator: `?:`

```
ex: logical_expr ? expr_1 : expr_2
 If logical_expr is TRUE then expr1 else expr2
```

Operator precedence

Operators	Precedence (higher precedence on top, the same precedence on the same line)
1. Postfix	`expr++ expr--`
2. Unari	`++expr --expr +expr -expr ~ !`
3.	`* / %`
4.	`+ -`
5.	`<< >> >>>`
6.	`< > <= >= instanceof`
7.	`== !=`
8.	`&`
9.	`^`
10.	`|`
11.	`&&`
12.	`||`
13.	`? :`
14.	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Statements

Sequential Composition:

```
{
     instr1;
     instr2; …
}
```

Conditional:

```
if (logica_expr)
    instr;

if (logical_expr)
    instr1;
else
    instr2;
```

Obs: `logical_expr` is evaluated to `true or false`. Numerical values are not allowed.

Loop Statements

While statement:

```
while(logical_expr)
    instr
```

do-while statement:

```
do
    instr
while(logical_expr);
```

Obs: `Instr` is executed as long as `logical_expr is true.`

Loop Statement

FOR statement:

```
for(initialization;termination; step)
    instr
```

Obs: none of the `initialization, termination, step` are mandatory

```
int suma=0;
for(int i=1;i<10;i++)
   suma+=i;

for(int i=0,suma=0;i<10;i++)
        suma+=i;

    for(;;)
        // instruction
```

Enhanced FOR (EACH) statement

Syntax(JSE >=5):

```
for(Type elemName : tableName)
   instr;
```

```
int[] x={1, 4, 6, 9, 10};
for(int el:x)
  System.out.println(el);

for(int i=0;i<x.length;i++)
        System.out.println(x[i]);
```

Obs: Table elements cannot be modified by using enhanced for statement

```
 int[] x={1,4,6,10};
   for(int el:x){
      System.out.print(" "+el);
      el+=2;
   }
//1 4 6 10
   for(int e:x){
      System.out.print(" "+e);        //?
   }
```

Return statement:

```
return;
return value;
```

Break statement: terminates the execution of a loop

```
int[] x= { 2, 8, 3, 5, 12, 8, 62};
 int elem = 8;
boolean gasit = false;
 for (int i = 0; i < x.length; i++) {
            if (x[i] == elem) {
                gasit = true;
                break;
            }
    }
```

Continue statement

- skips the current iteration of a loop statement
- stops the execution of the loop instructions and forces the re-evaluation of the loop termination condition

```
int[] x= { 2, 8, 3, 5, 12, 8, 62};
int elem = 8;
int nrApar=0;
 for (int i = 0; i < x.length; i++) {
      if (x[i] != elem)
        continue;
      nrApar++;
}
```

Switch statement

```
switch(integral-selector) {
    case integral-value1 : statement; [break;]
    case integral-value2 : statement; [break;]
    case integral-value3 : statement; [break;]
    case integral-value4 : statement; [break;]
    case integral-value5 : statement; [break;]
    // ...
    default: statement;
}
```

Switch example

```
switch (luna) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12: nrZile = 31; break;
            case 4:
            case 6:
            case 9:
            case 11: nrZile = 30; break;
            case 2: if ( anBisect(an) )
                            nrZile = 29;
                    else
                            nrZile = 28;
                    break;
            default:
                System.out.println("Luna invalida");
        }
```

A simple Java program

```
//Test.java
    public class Test {
        public static void main(String[] args) {
            System.out.println("Hello");
            for(String el : args)
            System.out.println(el);
        }
    }
```

Compilation:

```
javac Test.java
```

Execution:

```
java Test

java Test ana 1 2 3
```

!!! You can use int value=Integer.parseInt(args[i]) in order to transform a string value into an int value.

Object-oriented programming Concepts

Class: represents a new data type

- Corresponds to an implementation of an ADT.

Object: is an instance of a class.

- The objects interact by messages.

Message: used by objects to communicate.

- A message is a method call.

Encapsulation(hiding)

- data (state)
- Operations (behaviour)

Inheritance: code reusing

Polymorphism – the ability of an entity to react differently depending on the context

Java Classes and Objects

- Class Declaration/Definition:

```
//ClassName.java
   [public] [final] class ClassName{
   [data (fields) declaration]
   [methods declaration and implementation]
   }
```

1. A class defined using `public` modifier it is saved into a file with the class name `ClassName.java.`
2. A file `.java` may contain multiple class definitions, but only one can be public.
3. Java vs. C++:
   - No 2 different files (.h, .cpp).
   - Methods are implemented when are declared.
   - A class declaration does not end with `;`

Java Classes and Objects

- Examples:

```
//Persoana.java
public class Persoana{
//...
}

// Complex.java
class Rational{
//...
}

class Natural{
//...
}

public class Complex{
//...
}
```

Java Classes and Objects

- Class Members (Fields) declaration:

```
... class ClassName{
    [access_modifier][static][final] Type name[=init_val];
}
```

modiaccess_modifier can be public, protected, private.

1. Class members can be declared anywhere inside a class.
2. Access modifier must be given for each field.
3. If the access modifier is missing, the field is visible inside the package (directory).

Java Classes and Objects

- Examples:

```
//Persoana.java
 public class Persoana{
    private String nume;
    private int varsta;
 //...
 }

 //Punct.java
 public class Punct{
 private double x;
 private double y;
 //...
 }
```

Java Classes and Objects

- Initializing fields
  - At declaration-site:

```
private double x=0;
```

  - in a special initialization block:

```
public class Rational{
 private int numarator;
 private int numitor;
 {
     numarator=0;
     numitor=1;
 }
 //...
}
```

  - in constructor.

Any field that is not explicitly initialized will take the default value of its type.

Constructors

- The constructor body is executed after the object memory space is allocated in order to initialize that space.

```
[...] class ClassName{
   [access_modifier] ClassName([list_formal_parameters]){
   //body
}
}
```

```
acces_modifier ∈ {public, protected, private}
```

```
list_formal_parameters takes the following form:
   Type1 name1[, Type2 name2[,...]]
```

1. The constructor has the same name as the class name (case sensitive).
2. The constructor does not have a return type.
3. For a class without any declared constructor, the compiler generates an implicit public constructor (without parameters).

Overloading Constructors

- A class can have many constructors, but they must have different signatures. .

```
//Complex.java
public class Complex{
  private double real, imag;

public Complex(){       //implicit constructor
 real=0;
  imag=0;
}

  public Complex(double real){
     this.real=real;
     imag=0;
  }

  public Complex(double real, double imag){ //...
   }

  public Complex(Complex c){ //...
   }
}
```

this

- It refers to the current (receiver) object.
- It is a reserved keyword used to refer the fields and the methods of a class.

```
//Complex.java
public class Complex{
  private double real, imag;
  //...
  public Complex(double real){
    this.real=real;
    imag=0;
  }

  public Complex(double real, double imag){
      this.real=real;
      this.imag=imag;
  }

  public Complex suma(Complex c){
  //...
  return this;
  }
}
```

Calling another constructor

- `this` can be used to call another constructor from a given constructor.

```
//Complex.java
public class Complex{
  private double real, imag;

 public Complex(){
  this(0,0);
  }

 public Complex(double real){
     this(real,0);
  }

public Complex(double real, double imag){
     this.real=real;
     this.imag=imag;
  }
  //...
 }
```

Calling another constructor

1. The call of another constructor must be the first instruction in the caller constructor.
2. The callee constructor cannot be called twice.
3. It is not possible to call two different constructors.
4. A constructor cannot be called from a method.

```
 //Punct.java
public class Punct{
   private int x, y;

   public Punct(){
     this(0,0);
   }

   public Punct(int x, int y){
       this.x=x;
       this.y=y;
   }

  public void muta(int dx, int dy){
       this(x+dx, y+dy);
   }
}
//Erorrs?
```

Creating objects

- Operator **`new:`**

```
Punct p=new Punct();   //the parentheses are compulsory
Complex c1=new Complex();
Complex c2=new Complex(2.3);
Complex c3=new Complex(1,1.5);

Complex cc; //cc=null, cc does not refer any object
cc=c3;      //c3 si cc refer to the same object in the memory
```

1. The objects are created into the heap memory.
2. The operator new allocates the memory for an object;

Defining methods

```
[...] class ClassName{
   [access_modifier] Result_Type methodName([list_formal_param]){
       //method body
   }
}
```

`access_modifier ∈ {public, protected, private}`

`list_formal_param` **takes the form** `Type1 name1[, Type2 name2[, ...]]`

`Result_Type` poate can be any primitiv type, reference type, array, or `void.`

1. If the access_modifier is missing, that method can be called by any class defined in that package (director).
2. If the return type is not `void,` then each execution branch of that method must end with the statement `return.`

Defining methods

```
//Persoana.java
public class Persoana{
   private byte varsta;
   private String nume;
   public Persoana(){
      this("",0);
   }
   public Persoana(String nume, byte varsta){
    this.nume=nume;
    this.varsta=varsta;
   }
   public byte getVarsta(){
      return varsta;
   }
   public void setNume(String nume){
    this.nume=nume;
   }
   public boolean maiTanara(Persoana p){//...
   }
}
```

Overloading methods

- A class may contain multiple methods with the same name but with different signature. A signature = return type and the list of the formal parameters

```
public class Complex{
  private double real, imag;
  // constructors ...
  public void aduna (double real){
   this.real+=real;
   }
   public void aduna(Complex c){
       this.real+=c.real;
       this.imag+=c.imag;
   }
   public Complex aduna(Complex cc){
   this.real+=cc.real;
   this.imag+=cc.imag;
   return this;
   }
}
//Erorrs?
```

- Java does not allow the operators overloading.
- Class String has overloaded operators `+` and `+=`.

```
String s="abc";
     String t="EFG";
     String r=s+t;
     s+=23;
     s+=' ';
     s+=4.5;
//s="abc23 4.5";
//r="abcEFG"
```

- Destructor: In Java there is no any destructor.
  - The garbage collector deallocates the memory .

Objects as Parameters

- Objects can be formal parameters for the methods

- A method can return an object or an array of objects.

```
public class Rational{
private int numarator, numitor;
//Constructors ...

public void aduna(Rational r){
//...
}
public Rational scadere(Rational r){
//...
}

}
```

Passing arguments

- Primitive type arguments (boolean, int, byte, long, double) are passed by value. Their values are copied on the stack.
- Arguments of reference type are passed by value. A reference to them is copied on the stack, but their content (fields for objects, locations for array) can be modified if the method has the rights to accces them.

1. There is not any way to change the passing mode(like & in C++).

```
class Parametrii{
    static void interschimba(int x, int y){
        int tmp=x;
        x=y;
        y=tmp;
  }
  public static void main(String[] args) {
        int x=2, y=4;
        interschimba(x,y);
        System.out.println("x="+x+" y="+y);   //?
    }
}
```

Passing arguments

```
class B{
    int val;
    public B(int x){
        this.val=x;
    }
    public String toString(){
        return ""+val;
    }
     static void interschimba(B x, B y){
        B tmp=x;
        x=y;
        y=tmp;
        System.out.println("[Interschimba B] x="+x+" y="+y);
    }
    public static void main(String[] args) {
        B bx=new B(2);
        B by=new B(4);
        System.out.println("bx="+bx+" by="+by);
        interschimba(bx,by);
        System.out.println("bx="+bx+" by="+by);   //?
     }
}
```

Passing arguments

```
class B{
    int val;
    public B(int x){
        this.val=x;
    }
    public String toString(){
        return ""+val;
    }
     static void interschimbaData(B x, B y){
        int tmp=x.val;
        x.val=y.val;
        y.val=tmp;
        System.out.println("[InterschimbaData] x="+x+" y="+y);
    }
    public static void main(String[] args) {
        B bx=new B(2);
        B by=new B(4);
        System.out.println("bx="+bx+" by="+by);
        interschimbaData(bx,by);
        System.out.println("bx="+bx+" by="+by);   //?
     }
}
```

Array of objects

- Each array element must be allocated and intialized.

```
public class TablouriObiecte {
   static void genereaza(int nrElem, Complex[] t){
        t=new Complex[nrElem];
        for(int i=0;i<nrElem;i++)
            t[i]=new Complex(i,i);
   }
   static Complex[] genereaza(int nrElem){
        Complex[] t=new Complex[nrElem];
        for(int i=0;i<nrElem;i++)
            t[i]=new Complex(i,i);
        return t;
   }
   static void modifica(Complex[] t){
        for(int i=0;i<t.length;i++)
            t[i].suma(t[i]);
   }
//...
```

Array of objects

```
  static Complex suma(Complex[] t){
        Complex suma=new Complex(0,0);
        for(int i=0; i<t.length;i++)
            suma.aduna(t[i]);
        return suma;
    }
   public static void main(String[] args) {
        Complex[] t=genereaza(3);
        Complex cs=suma(t);
        System.out.println("suma "+cs);
        Complex[] t1=null;
        genereaza(3,t1);
        Complex cs1=suma(t1);
        System.out.println("suma "+cs1);
        modifica(t);
        System.out.println("suma dupa modificare "+suma(t));

    }
}
```

The methods toString and equals

```
public class Complex{
    private double real, imag;
    public Complex(double re, double im){
      //...
    }
    public String toString(){
        if (imag>=0)
            return "("+real+"+"+imag+"i)";
        else
            return "("+real+imag+"i)";
    }

    public boolean equals(Object obj){
        if (obj instanceof Complex){
            Complex c=(Complex)obj;
             return (real==c.real) && (imag==c.imag);
        }
        return false;
    }
//...
}
```

Static methods

- Are declared using the keyword **`static`**
- **They are shared by all class instances**

```
public class Complex{
    private double real, imag;
    public Complex(double re, double im){
     //...
    }
    public static Complex suma(Complex a, Complex b){
        return new Complex(a.real+b.real, a.imag+b.imag);
    }
//...
}
```

Static methods

- They are called using the class name:

```
Complex a,b;
//... initialization a and b
Complex c=Complex.aduna(a, b);
```

1. A static method cannot use those fields (or call those methods) which are not static. It can use or call only the static members.
2. This does not make sense inside the body of a static method

Static fields

```
public class Natural{
   private long val;
  public static long MAX=232.... //2^63-1
 //....
}
public class Produs {
    private static long counter;
    private final long id=counter++;
    public String toString(){
        return ""+id;
    }
   //....
```

Static fields are shared by all class instances. They are allocated only once in the memory.

Static fields

- Initialization:
  - At declaration site:

```
public static long MAX=2000;
```

  - In a special intialization block

```
public class Natural {
        public  static long MAX;
 static {
     MAX=2000;
 }
```

If a static field is not intialized, it will take the default value of its type:

```
private static long counter; //0
```