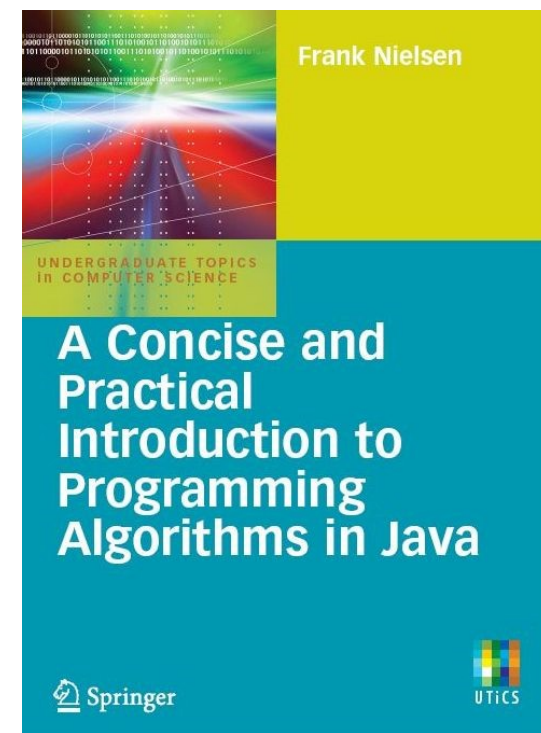


A Concise and Practical Introduction to Programming Algorithms in Java



Chapter 5: Classes and objects

Frank NIELSEN



✉ nielsen@lix.polytechnique.fr

So far...: Executive summary

- **Lecture 1:** Variable, Expression, Assignment
- **Lecture 2:** Loops (`for while do`)
Conditional structures (`if else switch`)
Boolean predicate and connectors (`|| &&`)
Loop escape `break`
- **Lecture 3:** functions (`static`) and recursion (terminal or not)
- **Lecture 4:** Objects



Indenting source code (. java)

- Increase code readability
- Avoid mistyping bugs (matching { })

Source code formatter, pretty printer, beautifier

Different conventions are possible (but choose one)

Implemented more or less in Software (S/W) Nedit, Jcreator, Jindent, etc...



Indenting source code (. java)

<http://java.sun.com/docs/codeconv/>

```
if (condition) {  
    statements;  
}
```

```
if (condition) {  
    statements;  
} else {  
    statements;  
}
```

```
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

Examples for `if else` conditions



Indenting source code (. java)

```
for (initialization; condition; update) {  
    statements;  
}
```

```
while (condition) {  
    statements;  
}
```

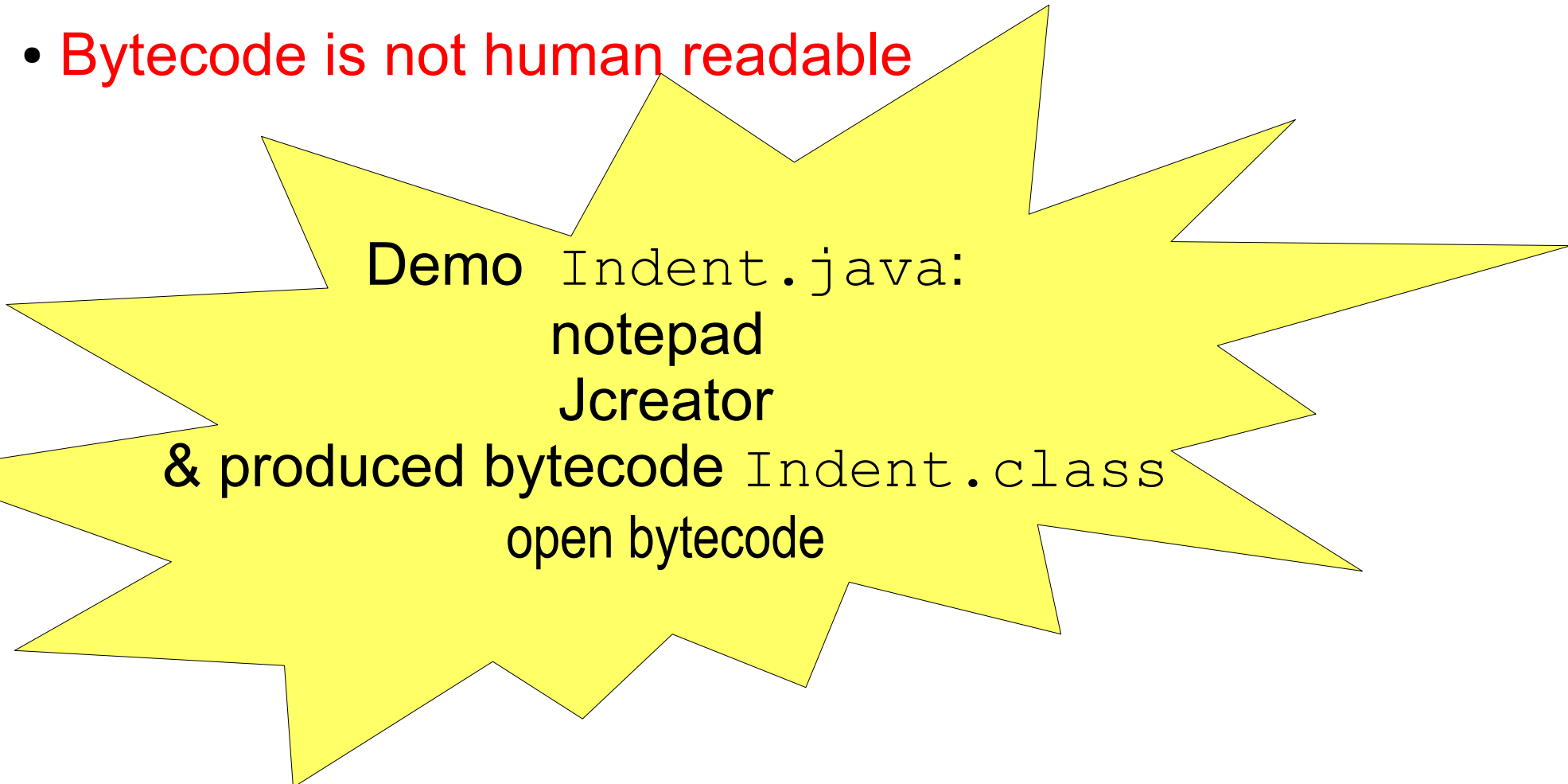
```
do {  
    statements;  
} while (condition);
```

<http://java.sun.com/docs/codeconv/>



Indenting source code (.java)

- Bytecode size and indentation:
Does not change fundamentally
- Bytecode is not human readable



Demo `Indent.java`:
notepad
Jcreator
& produced bytecode `Indent.class`
open bytecode

Indenting source code (. java)

Sometimes in Java code (Internet), comments include commands for generating automatically documentation by other tools:
... Like **javadoc** (paradigm literate programming, etc.)

La classe TC se trouve a:

<http://www.enseignement.polytechnique.fr/informatique/profs/Julien.Cervelle/TC/>

	Change l'entrée des méthodes lire pour que la lecture se fasse à partir d'une chaîne
<code>static double</code>	<code>lireDouble()</code> lecture d'un double sur l'entrée
<code>static void</code>	<code>lireFichier(java.lang.String fileName)</code> Change l'entrée des méthodes lire pour que la lecture se fasse à partir d'un fichier
<code>static int</code>	<code>lireInt()</code> lecture d'un int sur l'entrée
<code>static java.lang.String</code>	<code>lireLigne()</code> lecture d'une ligne sur l'entrée

Class TC

<http://java.sun.com/j2se/javadoc/>



Functions in Java

- Static functions that returns a `type` (eventually `void`)
- Functions are called inside the `main` procedure (or in other function body)
- Displaying and calling function are different (be not confused with SciLab or Maple)
`System.out.println(function()) ;`
- Java is a compiled OO language , not an interpreter

Functions: void/display

**Java cannot cast void type into a String,
so the compiler javac generates an error.
(type checking)**

```
class Functions
{
    static void PascalTriangle(int depth)
    { // ...
        return ;
    }

    public static void main(String[] toto)
    {
        System.out.println(PascalTriangle(5));
    }
}
```

'void' type not allowed here



Functions: void/display

Java is *not an interpreter* like SciLab or Maple

**Functions are called within a block of instructions...
... not in the console!!!!**

```
class Functions  
{
```

```
    static double f(double x)  
    {return x;}
```

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

```
C:\J2>f(3)  
'f' n'est pas reconnu en tant que commande interne  
ou externe, un programme exécutable ou un fichier de commandes.
```

Variables: static or not...

Static variables are declared in the **class body**

```
class Toto
{
static int count1, count2;
...
}
```

Otherwise non-static variables (usual) are declared in **function bodies** (main, etc.)

```
public static void main(String[] args)
{double x;int i;}
```

- Variables are kept in memory in their function scope { . . . }
- Static variables are kept in memory and can be shared by several functions...



static or not...

```
class Functions
```

```
{  
    static int count1, count2;
```

```
    static void f1() {count1++;}
```

```
    static void f2() {count2++;}
```

```
    public static void main(String[] args)
```

```
    {
```

```
        count1=0;
```

```
        count2=0;
```

```
        for(int i=0;i<1000;i++)
```

```
        {
```

```
            double rand=Math.random();
```

```
            if (rand<0.5)
```

```
                {f1();}
```

```
            else
```

```
                {f2();}
```

```
        }
```

```
        System.out.println("count1:"+count1);
```

```
        System.out.println("count2:"+count2);
```

```
    }
```



Java is pass by value

```
class BadSwap
```

```
{
```

```
    static void swap(int arg1, int arg2)
```

```
    {
```

```
        int tmp;
```

```
        tmp=arg1;
```

```
        arg1=arg2;
```

```
        arg2=tmp;
```

```
    }
```

```
public static void main(String[] toto)
```

```
{
```

```
    int a=3;
```

```
    int b=2;
```

```
    System.out.println("a:"+a+" b:"+b);
```

```
    swap(a,b);
```

```
    System.out.println("After the swap...");
```

```
    System.out.println("a:"+a+" b:"+b);
```

```
}
```

```
}
```

...and arrays and objects are pass by reference



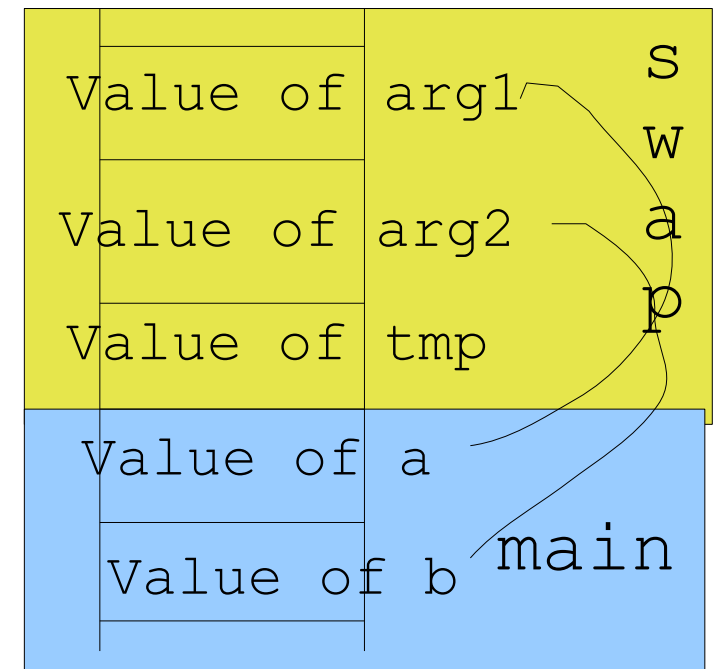
Managing memory & functions

When calling a function f , the current function (`main`) indicates
.....where to write the **value** of the result

To obtain the result, function f uses a **local memory**

In that local memory, **values** of arguments are available

```
//current function body {}  
int a=3,b=2;  
swap(a,b)
```

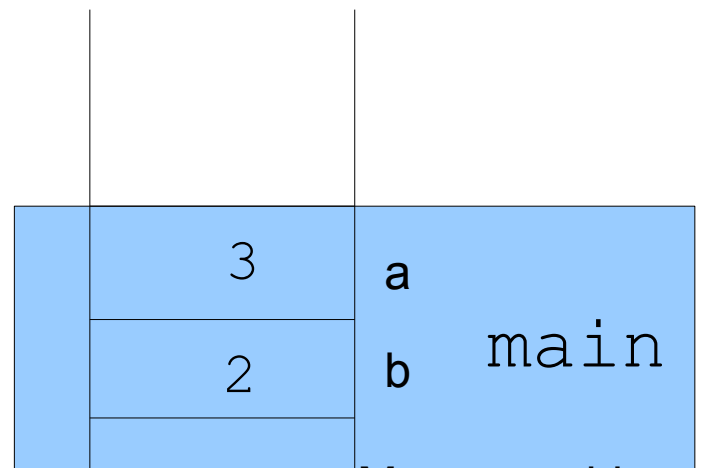


1. Create memory for local variables of function main
2. Assign values for a and b

```
static void swap(int arg1, int arg2)
{
    int tmp;

    tmp=arg1;
    arg1=arg2;
    arg2=tmp;
}
```

```
public static void main(String[] toto)
{
    int a=3;
    int b=2;
    swap(a,b);
}
```



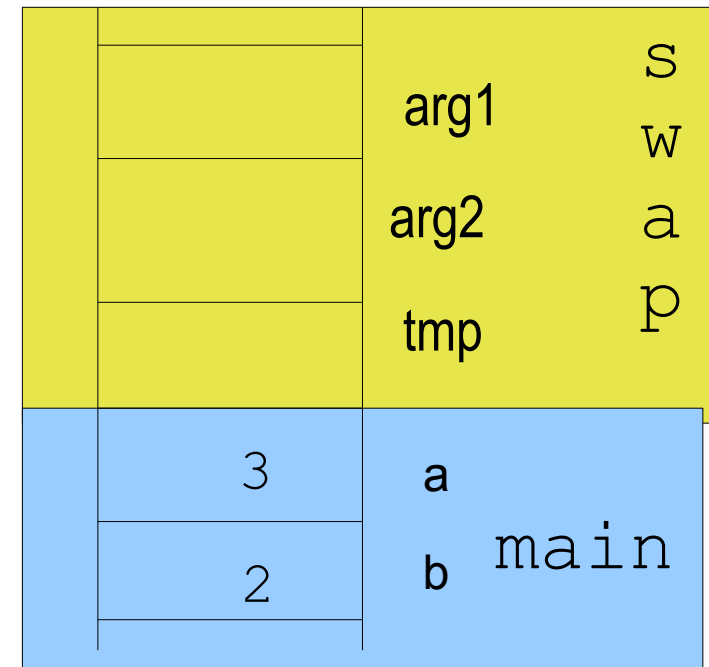
Memory address

3. create local space for function swap

```
static void swap(int arg1, int arg2)
{
    int tmp;

    tmp=arg1;
    arg1=arg2;
    arg2=tmp;
}
```

```
public static void main(String[] toto)
{
    int a=3;
    int b=2;
    swap(a,b);
}
```



4. evaluate expression for getting values of arg1 and arg2 swap(a,b) becomes swap(3,2)

```
static void swap(int arg1, int arg2)
```

```
{  
  int tmp; // 0 is default value
```

```
  tmp=arg1;  
  arg1=arg2;  
  arg2=tmp;  
}
```

```
public static void main(String[] toto)  
{
```

```
  int a=3;  
  int b=2;  
  swap(a,b) ;
```

```
}
```

	3	arg1	swap
	2	arg2	
	0	tmp	
	3	a	main
	2	b	



5. Execute instruction `tmp=arg1`

```
static void swap(int arg1, int arg2)
```

```
{
  int tmp; // 0 is default value
```

```
  tmp=arg1;
  arg1=arg2;
  arg2=tmp;
}
```

```
public static void main(String[] toto)
```

```
{
  int a=3;
  int b=2;
  swap(a,b) ;
```

```
}
```

	3	arg1	s
	2	arg2	w
	3	tmp	a
			p
	3	a	
	2	b	main



6. Execute instruction `arg1=arg2`

```
static void swap(int arg1, int arg2)
```

```
{  
  int tmp; // 0 is default value
```

```
  tmp=arg1;  
  arg1=arg2;  
  arg2=tmp;  
}
```

```
public static void main(String[] toto)
```

```
{  
  int a=3;  
  int b=2;  
  swap(a,b) ;
```

```
}
```

		arg1	arg2	tmp	swap
	2				
	2				
	3				
	3	a	b		main
	2				



7. Execute the sequence of instructions in the swap block

Notice that here the swapped has been performed

```
static void swap(int arg1, int arg2)
```

```
{  
    int tmp; // 0 is default value
```

```
    tmp=arg1;  
    arg1=arg2;  
    arg2=tmp;  
}
```

```
public static void main(String[] toto)
```

```
{  
    int a=3;  
    int b=2;  
    swap(a,b) ;
```

```
}
```

		arg1	S
	2		
	3		
	3		
	3	a	main
	2		



5. Execute the sequence of instructions in the swap block

```
static void swap(int arg1, int arg2)
```

```
{
  int tmp; // 0 is default value
```

```
  tmp=arg1;
  arg1=arg2;
  arg2=tmp;
}
```

```
public static void main(String[] toto)
```

```
{
  int a=3;
  int b=2;
  swap(a,b) ;
```

```
}
```

	2	arg1	s
	3	arg2	w
	3	tmp	a
			p
	3	a	
	2	b	main



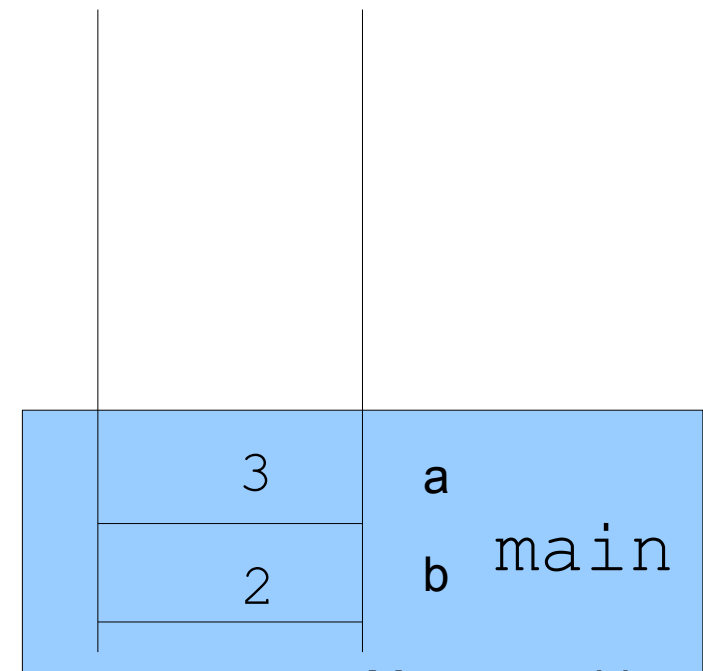
8. Return result of function swap (here void!!!)
9. Release memory allocated for swap

```
static void swap(int arg1, int arg2)
{
    int tmp; // 0 is default value

    tmp=arg1;
    arg1=arg2;
    arg2=tmp; // we omitted return ;
}

public static void main(String[] toto)
{
    int a=3;
    int b=2;
    swap(a,b) ;
}
```

**Variables a and b
have kept their original values**

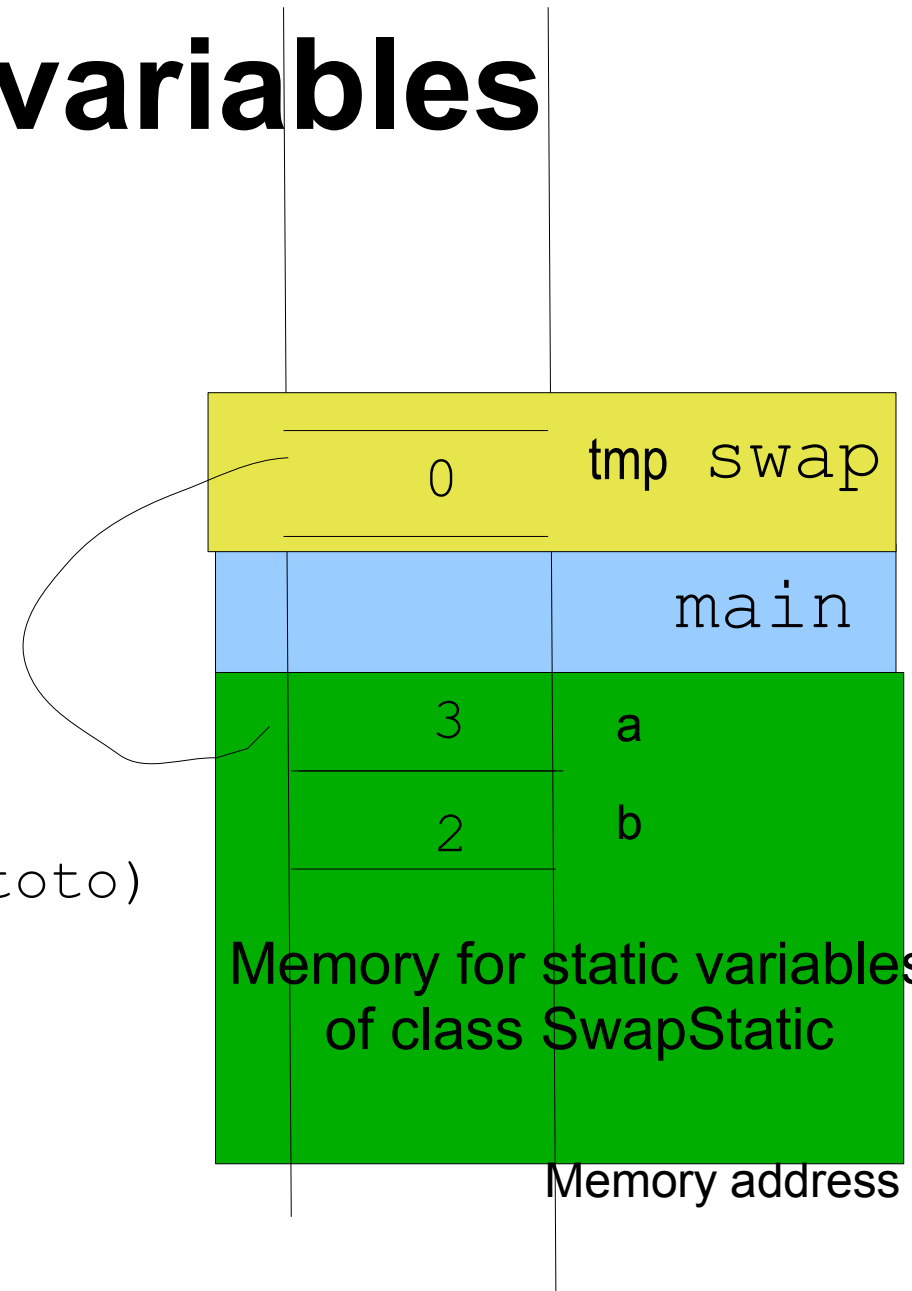


Memory for static variables

```
class SwapStatic
{
    static int a,b;

    static void swap()
    {
        ...
    }

    public static void main(String[] toto)
    {
        a=3;
        b=2;
        swap();
    }
}
```



By passing using static

```
class SwapStatic
{
    static int a,b;

    static void swap()
    {
        int tmp;// ok not to be static

        tmp=a;
        a=b;
        b=tmp;
    }

    public static void main(String[] toto)
    {
        a=3;
        b=2;

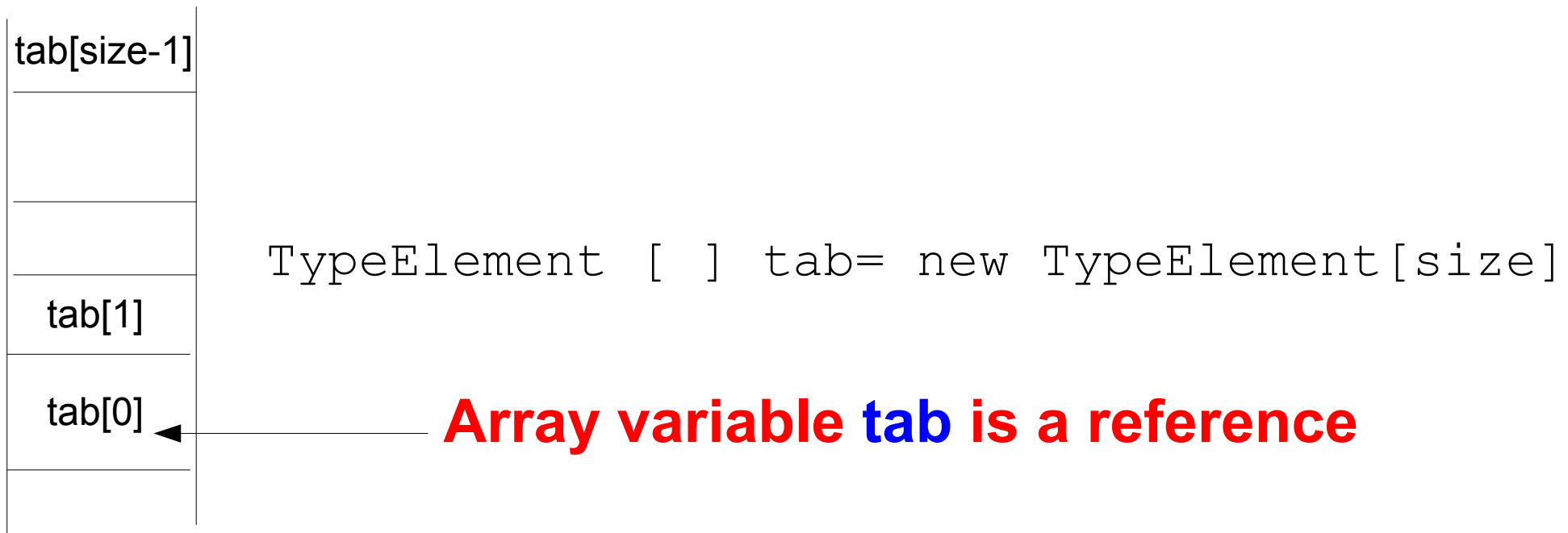
        System.out.println("a:"+a+" b:"+b);
        swap();
        System.out.println("After the swap...");
        System.out.println("a:"+a+" b:"+b);
    }
}
```



Memory for arrays and pass by reference

Arrays are allocated a continuous memory location for storing TYPE elements

The value of the array variable is a reference to the beginning of the array



Memory for arrays (heap)



Memory management using new

```
Type [] tab=new Type[Expression];
```

- Evaluate `Expression` to get an **integer value**.
- Arrays are stored not in the local function memory, but rather in the **global program memory**:
heap, tas en francais
- A cell (array element) in the heap (program memory) is accessible by any function which has as a local (non-static) variable a reference to the array.

```

class ArrayReference
{

    public static void swap(int [] t, int i, int j)
    {
        int tmp;
        tmp=t[i];
        t[i]=t[j];
        t[j]=tmp;
    }

    public static void Display(int [] tab){... »

    public static void main(String[] args)
    {
        //int [] array=new int[10];
        int [] array={0,1,2,3,4,5,6,7,8,9};
        Display(array);
        swap(array,1,2);
        Display(array);
    }
}

```

0	1	2	3	4	5	6	7	8	9
0	2	1	3	4	5	6	7	8	9



Memory management using new

```
class BuildArray{

    // Return a reference to an array
    public static int [] BuildArray(int size, int defaultval)
    {
        int [] result=new int[size];
        for(int i=0;i<size;i++) result[i]=defaultval;
        return result;
    }

    public static void Zero(int[] tab, int pos)
    {
        tab[pos]=0;
    }

    public static void main(String [] argarray)
    {
        int v []=BuildArray(10,4);
        Display(v);
        Zero(v,2);
        Display(v);
    }
}
```



4	4	4	4	4	4	4	4	4	4
4	4	0	4	4	4	4	4	4	4



Synopsis of this lecture

- Objects and records (fields, *enregistremments*)
- Object constructors
- Class type variables: References
- Functions on objects: Methods
- Array of objects
- Examples

Why do we need objects?

Encapsulate functions/data acting on a same domain

For example, the `String` type

Allows one to work on **complex entities**: **Data structures**

For examples:

- Dates are triplets of numbers (MM/DD/YYYY)
- 2D point with co-ordinates (x,y)
- Student: Lastname, Firstname, Group, etc.

These are called **object records** (fields)



Java is an oriented-object (OO) programming language



Declaring classes and objects

- Choose record/field names (enregistrement)
- Define a type of each record
- Similar to variables but without keyword `static`
- Class is then a **new type** with name...
... the class name

Toy example

```
public class Date
{
    int dd;
    int mm;
    int yyyy;
}
```

Fields (champs/enregistrements) are also called **object variables**
Do not have the leading keyword `static`

Let `day` be a variable of type `Date` then

```
day.dd    day.mm    dd.yyyy
```

are variables of type `int`

Toy example

```
public class Student
{
    String Lastname;
    String Firstname;
    int Company;
    double [ ] Marks;
    . . .
}
```

Class Student encapsulates data attached to a student identity.

Constructors

To use an object, we first need to **build** it

We construct an object using the instruction `new`

But first, we need to define a **constructor** for the class
A constructor is a **method** (non-static function) ...
...bearing the class' name

This method does not return a result but assigns...
...values to the object's field

Use `this.field` to access field of the object

Constructors

```
public class Date
{
    int dd;
    int mm;
    int yyyy;
    // Constructor
    public Date(int day, int month, int year)
    {this.dd=day;
    this.mm=month;
    this.yyyy=year; }
}
```

Create an object of type Date

```
Date day=new Date(23,12,1971);
```



Public class YYY stored in YYY.java

```
public class Date
{
    int dd;
    int mm;
    int yyyy;

    public Date(int day, int month, int year)
    {
        this.dd=day;
        this.mm=month;
        this.yyyy=year;
    }
}
```

Filename: Date.java

```
class TestDate{
    public static void main(String args)
    {
        Date day=new Date(23,12,1971);
    }
}
```

Filename: TestDate.java



Constructors

- Possibly several constructors (with different signatures)
- Best, to define a single one with all fields initialized
- Keyword `this` means the currently built object
(*not compulsory* to write it explicitly but recommended)

```
public Date(int day, int month, int year)
{
    dd=day;
    this.mm=month;
    yyyy=year;
}
```

- If no constructor is built, the system uses the by-default one
(not recommended)

```
Date day=new Date(); // see poly pp. 59–61
day.yyyy=1971;
```



The `null` object

- This object is common to all classes
- Not possible to assign its fields
- Nor retrieve values of its fields, either
(exception `NullPointerException` raised)
- Used for initializing a variable of type object:
`Student stud=null;`
- It is often recommender to check if an object is null or not:

```
if ( stud!=null) stud.company=2;
```

Functions/methods on objects

Objects can be parameters of functions

```
static TypeF F(Object1 obj1, ..., ObjectN objN)
```

Functions may return an object as a result:

```
static boolean isBefore (Date d1, Date d2)  
static Date readDate()
```

Example

```
public class Date
{
    int dd;
    int mm;
    int yyyy;

    public static final String[ ] months={
        "January", "February", "March", "April", "May",
        "June", "July", "August", "September", "October",
        "November", "December"
    };

    // Constructor
    public Date(int day, int month, int year)
    {
        this.dd=day;
        this.mm=month;
        this.yyyy=year;
    }
}
```

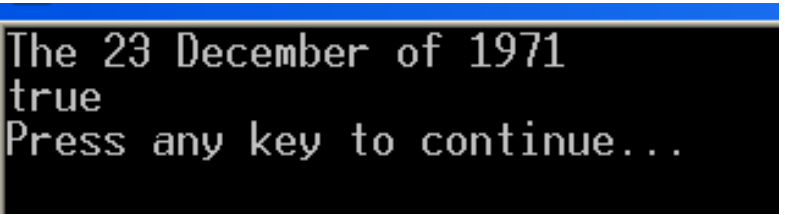


Example

```
class TestDate{
static void Display(Date d){
    System.out.println("The "+d.dd+" "+Date.months[d.mm-1]+" of "+d.yyyy);
}
```

```
static boolean isBefore(Date d1, Date d2)
{
boolean result=true;
if (d1.yyyy>d2.yyyy) result=false;
if (d1.yyyy==d2.yyyy && d1.mm>d2.mm) result=false;
if (d1.yyyy==d2.yyyy && d1.mm==d2.mm && d1.dd>d2.dd)
result=false;
return result;
}
```

```
public static void main(String[] args)
{
Date day1=new Date(23,12,1971);
Display(day1);
Date day2=new Date(23,6,1980);
System.out.println(isBefore(day1,day2));
}
```



```
The 23 December of 1971
true
Press any key to continue...
```



```

class TestDate{
    ...

    static Date lireDate()
    {
        int jj, mm, aaaa;
        System.out.println("Jour?");
        jj=TC.lireInt();
        System.out.println("Mois?");
        mm=TC.lireInt();
        System.out.println("Annee?");
        aaaa=TC.lireInt();
        Date day=new Date(jj,mm,aaaa);
        return day;
    }

```

```

Jour?
26
Mois?
06
Annee?
2003
The 26 June of 2003

```

```

public static void main(String[] args)
{
    Display(lireDate());
}

```



Variable of Type Object: Reference

A variable of type Object is a **reference** on that object

It stores the memory address of this referenced object

Thus when we write:

```
Date day1=new Date(23,12,1971);  
Date day2=day1;
```

```
Display(day2);  
day2.mm=6;  
Display(day1);
```

```
The 23 December of 1971  
The 23 June of 1971
```

The date d1 is not copied, only the reference of...

...d1 is assigned to d2



Copying objects...

To copy (clone) an object to another we need to do it fieldwise

```
// Two Scenarii:  
// day2 has already been created...  
day2.dd=day1.dd;  
day2.mm=day1.mm;  
day2.yyyy=day1.yyyy;  
  
// day2 object has not yet been created...  
static Date Copy(date day1)  
{  
    Date newdate=new Date (day1.dd,day1.mm,day1.yyyy);  
    return newdate;  
}  
...  
Date d2=Copy(d1);
```



Comparing two objects...

Do not use `==` for object equality

To compare objects, use a **tailored predicate**:

```
static boolean isEqual(Date d1, Date d2)
{
    return (d1.dd == d2.dd &&
            d1.mm == d2.mm &
            d1.yyyy== d2.yyyy);
}
```

Comparing two objects...

```
public static void main(String[] args)
{
    Date day1=new Date(23,12,1971);
    Date day2=day1; // beware not copying here.
    Just memory reference
    Date day3=new Date(23,12,1971);

    System.out.println(isEqual(day1,day3));

    System.out.println(day1);
    System.out.println(day2);
    System.out.println(day3);
}
```

```
true
Date@3e25a5
Date@3e25a5
Date@19821f
```

Physical (memory) versus logical equality

Array of objects

- Since classes defines new types...
... we can create **array of objects**

- To build an array of objects: `new nameT[sizearray]`

```
Date [ ] tabDates=new Date[31];
```

- When an array of object is built, the elements `Date[i]` are all initialized to the `null` object.

Example

```
public class XEvent
{
    Date when;
    String what;

    public XEvent(Date d, String text)
    {
        this.when=d;
        this.what=text;
    }
}
```

Filename XEvent.java

```
public class Date
{
    ...
    void Display()
    {
        System.out.println(dd+" "+months[mm-1]+" "+yyyy);
    }
    ...
}
```

Filename Date.java




```
public class TestXEvent
{
    public static void Display(XEvent e)
    {
        System.out.print(e.what+": ");
        e.when.Display();
    }

    public static void main(String [] args)
    {
        Date d1=new Date(26,6,2008);
        XEvent e1=new XEvent(d1,"Birthday Julien");

        Display(e1);

        XEvent [] tabEvent=new XEvent[5];
        tabEvent[0]=e1;
    }
}
```



```

public class TestXEvent
{
    public static void Display(XEvent e)
    {
        System.out.print(e.what+": ");
        e.when.Display();
    }

    public static boolean older(XEvent e1, XEvent e2)
    {
        return Date.isBefore(e1.when,e2.when);
    }

    public static XEvent oldest(XEvent[] tab)
    {
        XEvent result=tab[0];
        for(int i=1;i<tab.length;++i)
            if (older(tab[i],result)) result=tab[i];
        return result;
    }

    public static void main(String [] args)
    {
        Date d1=new Date(26,6,2003);
        XEvent e1=new XEvent(d1,"Birthday Julien");
        Date d2=new Date(20,11,2000);
        XEvent e2=new XEvent(d2,"Birthday Audrey");
        Date d3=new Date(23,6,1971);
        XEvent e3=new XEvent(d3,"Birthday Me");
        Display(e1);
        XEvent [] tabEvent=new XEvent[3];
        tabEvent[0]=e1;tabEvent[1]=e2;tabEvent[2]=e3;
        System.out.print("Oldest person::");Display(oldest(tabEvent));
    }
}

```

Birthday Julien: 26 June 2003
 Oldest person::Birthday Me: 23 June 1971
 Press any key to continue...



Objects with array members

Fields of objects may be arrays themselves

```
always built with new Type[sizearray]  
// sizearray might be an expression, i.e., 3*n+2
```

It is not necessary at compile time to know statically...
.... the array sizes

```
class Polynome{  
int degree;  
double [ ] coefficients;  
};
```

Strings: Basic objects in Java

- A string of character is **an object** with type **String**
- A variable of type String is a **reference** on that object:

```
String school= "Ecole Polytechnique";  
String vars=school;
```

- Once built, a string object **cannot** be modified
- Beware: use only for moderate length strings,
otherwise use the **class StringBuffer**

Class String: Some methods

A method is a function or procedure on an object class

Method Length(): gives the number of characters

```
String s= 'anticonstitutionnellement';  
System.out.println(s.length());
```

Method equals():

s1.equals(s2): Predicate that returns true **if and only if** the two strings s1 and s2 are made of the same sequence of characters.

```
String s1='Poincare';  
String s2=TC.lireMotSuivant();  
System.out.println(s1.equals(s2));
```

Beware: s1==s2 is different!

It compares the **reference** of the strings.
(*Physical versus logical equality test*)



Class String in action...

stringmethod.java

```
1 class stringmethod{
2
3
4     public static void main(String[] args)
5     {
6         String name="Ecole Polytechnique";
7         String promotion="2008";
8
9         String fullname=name+" "+promotion;
10
11         System.out.println(fullname);
12         System.out.println("Length:"+fullname.length());
13
14         System.out.println("Type a sentence so that I check whether it is Poincare or not");
15         String pattern="Poincare";
16         String query=TC.lireMotSuivant();
17
18         System.out.println(pattern.equals(query));
19     }
20 }
```

C:\PROGRA~1\XINXS~1\JCREAT~1\GE2001.exe

```
Ecole Polytechnique 2008
Length:24
Type a sentence so that I check whether it is Poincare or not
Poincare
true
Press any key to continue...
```



Class String: More methods

Method [charAt\(\)](#):

`s.charAt(i)` gives the character at the $(i+1)$ th position in string `s`.

```
String s = '3.14159265';  
System.out.println(s.charAt(1));
```

Method [compareTo\(\)](#):

`u.compareTo(v)` compares lexicographically the strings `u` with `v`.

```
String u='lien', v='lit', w='litterie';  
System.out.println(u.compareTo(v));  
System.out.println(v.compareTo(w));
```

From Javadoc...

		Compares this String to another Object.
<code>int</code>	<u><code>compareTo</code></u> (<u><code>String</code></u> anotherString)	Compares two strings lexicographically.

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html>



Lexicographic total order on strings

- If there is a position k at which strings differ:

`this.charAt(k) - anotherString.charAt(k) :`

```
String s1="Marin",s2="Martin"; // -11 from i to t
```

```
int index=3;// meaning 4th pos
```

```
System.out.println(s1.compareTo(s2));
```

```
System.out.println(s1.charAt(index)-s2.charAt(index));
```

- else the difference of string lengths:

`this.length() - anotherString.length() :`

```
String s3="Bien",s4="Bienvenue";
```

```
System.out.println(s3.compareTo(s4));
```

```
System.out.println(s3.length()-s4.length());
```

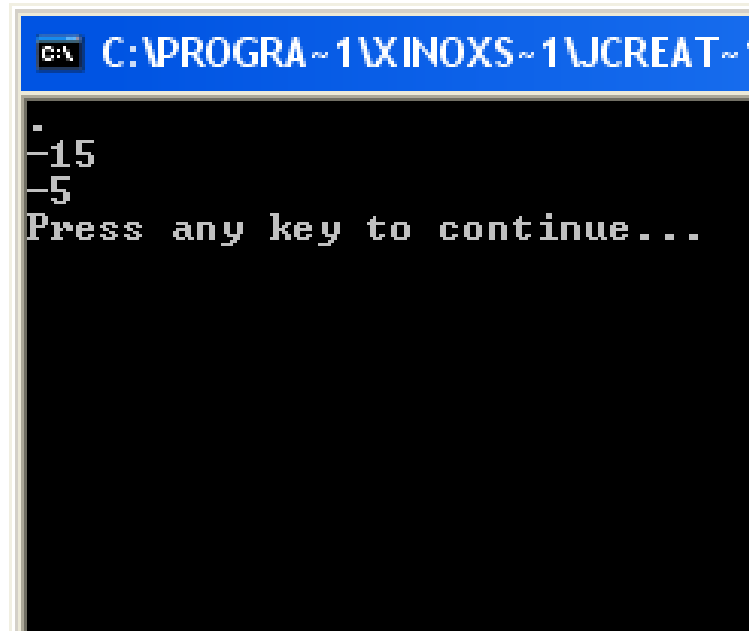
Class String: More methods

method2.java

```
class stringmethod2
{
    public static void main(String[] args)
    {
        String s="3.141559";

        System.out.println(s.charAt(1));

        String u="lien", v="lit", w="litterie";
        System.out.println(u.compareTo(v));
        System.out.println(v.compareTo(w));
    }
}
```



```
C:\PROGRA~1\XINXS~1\JCREAT~
.
-15
-5
Press any key to continue...
```



Demystifying the main function

```
class ClassName
{
    public static void main(String[ ] args)
    {
        ...
    }
}
```

Function main has an array of string of characters as arguments
These strings are stored in args[0], args[1], ...
... when calling java main s0 s1 s2 s3

Use Integer.parseInt() to convert a string into an integer

```
D:\J>javac main.java
D:\J>java main a small test to parse as a command line
0:a
1:small
2:test
3:to
4:parse
5:as
6:a
7:command
8:line
D:\J>
```



Parsing arguments in the main function

parsingarg.java

```
1 class parsingarg{
2
3
4
5     public static void main(String[] args)
6     {
7
8         String first=args[0];
9
10        for(int i=1; i<args.length;i++)
11            if (first.compareTo(args[i])>0)
12                first=args[i];
13
14        System.out.println("Lexicographically maximum string is:"+first);
15    }
16
17 }
```

```
D:\J>java parsingarg lit lien litterie
Lexicographically maximum string is:lien
D:\J>
```



Parsing arguments in the main function

parsingarg2.java

```
1 class parsingarg2{
2
3
4
5     public static void main(String[] args)
6     {
7
8         int first=0;
9
10        for(int i=1; i<args.length;i++)
11            if (Integer.parseInt(args[first])>Integer.parseInt(args[i]))
12                first=i;
13
14        System.out.println("Location of minimum argument:"+first);
15    }
16
17 }
```

```
D:\J>javac parsingarg2.java
```

```
D:\J>java parsingarg2 9 4 6 2 6 4 1 3 5 4 6
Location of minimum argument:6
```

```
D:\J>
```



More evolved Java program skeleton...

```
class Point
{
int x,y;
Point(int xx, int yy){x=xx;y=yy;}

public void Display()
{System.out.println("("+x+", "+y+") ") }
} // end of class Point
```

```
class Skeleton
{
// Static class variables
static int nbpoint=0;
static double x;
static boolean [] prime;

static int f1(int p){return p/2;}
static int f2(int p){return 2*p;}

public static void main(String [] argArray)
{
    System.out.println(f2(f1(3))+ " versus (!=) "+f1(f2(3)));
    Point p,q;
    p=new Point(2,1); nbpoint++;
    q=new Point(3,4); nbpoint++;
    p.Display();q.Display();
}
```

2 versus (!=) 3
(2,1)
(3,4)



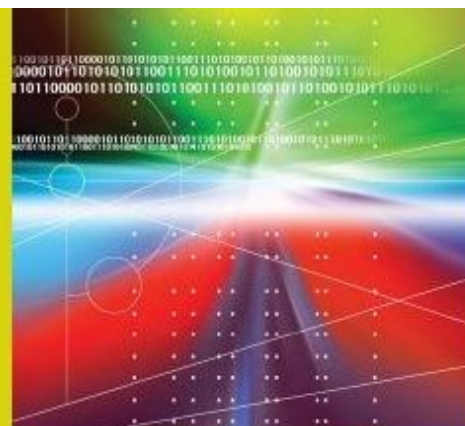
UNDERGRADUATE TOPICS IN COMPUTER SCIENCE

UTiCS Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.



Nielsen

Frank Nielsen



UNDERGRADUATE TOPICS IN COMPUTER SCIENCE

Frank Nielsen

A Concise and Practical Introduction to Programming Algorithms in Java

This gentle introduction to programming and algorithms has been designed as a first course for undergraduates, and requires no prior knowledge.

Divided into two parts the first covers programming basic tasks using Java. The fundamental notions of variables, expressions, assignments with type checking are looked at before moving on to cover the conditional and loop statements that allow programmers to control the instruction workflows. Functions with pass-by-value/pass-by-reference arguments and recursion are explained, followed by a discussion of arrays and data encapsulation using objects.

The second part of the book focuses on data structures and algorithms, describing sequential and bisection search techniques and analysing their efficiency by using complexity analysis. Iterative and recursive sorting algorithms are discussed followed by linked lists and common insertion/deletion/merge operations that can be carried out on these. Abstract data structures are introduced along with how to program these in Java using object-orientation. The book closes with an introduction to more

evolved algorithmic tasks that tackle combinatorial optimisation problems.

Exercises are included at the end of each chapter in order for students to practice the concepts learned, and a final section contains an overall exam which allows them to evaluate how well they have assimilated the material covered in the book.

COMPUTER SCIENCE

ISBN 978-1-84882-338-9

springer.com

A Concise and Practical Introduction to Programming
Algorithms in Java

A Concise and Practical Introduction to Programming Algorithms in Java

 Springer

