Programming with Java

procedures, functions and String

P. Bakowski

bako@ieee.org

Procedures

A procedure is a piece of code that has a name and can be called from the rest of the program. A procedure is defined necessarily in a class.

```
public class Lines1{
public static void main(String args[]){
drawLigne();
System.out.println("A text well delimited");
drawLigne();
System.out.println("After the line");
System.out.println();
drawLigne();
}
public static void drawLigne(){
for(int i=0;i<60;i++){System.out.println('*');}
System.out.println();}
}
```

Local variables

```
public class LocalVar {
  public static void drawLigne(){
  int a=0;              //  local variable
  while (a<60){
  System.out.println('*');
  a=a+1;
  }
  System.out.println() ;
  }
  public static void main(String args[]){
  int a=20;
  drawLigne();
  System.out.println(a);
  }
}
```

```
Java/Test>java LocalVar
************************************************************
20
```

Arguments of a procedure

```
public class ProcedureWithArguments {
  public static void drawLigne(int lengthLine){
  for (int i=0;i<lengthLine;i++){
  System.out.print('*');}
  System.out.println();
  }
  public static void main(String args[]){
  int k=50;
  drawLigne(k);
  drawLigne(k/2);
  drawLigne(k/4);
  }
}
```

To call a procedure that takes arguments, we write the name of the procedure followed by the values of the arguments between parentheses.

Locality of parameters

In Java the parameter passing is done by value. The parameter is a variable created when calling the procedure that receives a copy of the value passed to it.
A priori, a parameter change does not affect the corresponding variable from the main program.

```
public class Localite {
public static void increaseWrong(int val){
val=val+1;
}
public static void main(String args[]){
int k=10;
increaseWrong(k);
Keyboard.writeIntln(k);
}
}
```

The displayed value is ???

Overloading

It would be interesting to have several variants of drawLine. At first, one could imagine simply give them different names:

```
drawLine();
drawLineOfGivenLength(int k);
drawLineOfGivenLengthWithSymbol(int k,char s);
```

But it is easier for the programmer who uses procedures not cluttering memory with several names.
To solve this problem, java uses overloading. Overload gives the possibility to declare multiple methods with the same name.
They are then distinguished by their argument list.

```
drawLine();
drawLine(int k) ;
drawLine(int k, char s) ;
```

Overloading

```
public class LineMania{
public static void drawLine(int length,char symbol){
for(int i=0;i<length;i++) {
System.out.print(symbol);}
System.out.println();}

public static void drawLine(int length){
  drawLine(length,'*');}

public static void drawLine(){
  drawLine(60,'*');}

public static void main(String args[]){
drawLine();
drawLine(20);
System.out.println();
drawLine(100,'-');
}
}
```

Passing array as argument

For an array to be passed as a parameter, java pass array reference (address, in fact). As a result, the original array and that seen by the procedure occupy the same memory space. One consequence of this is that passing an array as a parameter does not copy it (Here `t` is the same address that `tab`).

```
public class TestTab1{
  public static void main(String args[]){
  int tab[]={8,2,1,23};
  displayArray(tab);
  }
  public static void displayArray(int t[]){
  for(int i=0;i<t.length;i++) {
    System.out.println(t[i]);
    if(i!=t.length-1)System.out.println(' ');
    }
  System.out.println();
  }
}
```

Procedure in different class

You can create libraries procedures.
You already use a: `class Keyboard`.

We may want to use our class LigeMania the same way.

To use the methods of a class A from class B:
- the code of the two classes must be in the same folder,
- you can call methods A simply by prefixing the name of the method A. (here `LineMania.`).

```
public class TestLines {
  public static void main(String args[]){
  LineMania.drawLine(100,'-');
  }
}
```

Functions

Function resembles a procedure, but its task is to return a value, which will be used later. The most immediate are mathematical functions.

The function **cos** of the **Math** class, calculates the cosine of its argument. It is used as follows:

```
double x= 3;
double y= Math.cos(x/2);
```

From a practical point of view :
- in a procedure the return value is **void**
- a function returns the non-void result using the keyword **return**.

Functions

```
public class AbsoluteValue {
  public static double absoluteValue(double x){
  double result;
  if(x<0)
    result=-x;
  else
    result= x;
  return result ;
  }
  public static void main(String args[]){
  double v=Keyboard.readDouble();
  System.out.println(absoluteValue(v));
  }
}
```

Functions that return an Array

A function may very well return an array. Simply declare it as a return type. The following function returns a table, for example, the size and content of the places are given as parameters.

```
public static double[]
  createArray(int size, double value) {
  double[] t=new double[size]; // creation of array
  for(int i=0;i<size;i++)
  t[i]= value;
  return t ;  // return of the address of array - t
}
```

Characters strings : String

Four ways to create a string (**String**)
- By writing this string with quotation marks in the program.
- Using the **new** statement.
- By means of the **+** operator:
  the result of a concatenation is a **new String**
- By means of a method that returns a **new String**.

```
String s = "Hen Hau";

char[] tab = {'H','e','n',' ','H','a','u'};
String s = new String(tab);
String s =
new String(new char[]{'H','e','n',' ','H','a','u'});
```

Method `length()`

Length can be called not only on a variable but on other kinds of expressions calculating a `String` object:

- a string between quotation marks : `"Hen Hau".length()`
– a string created by `new` : `(new String()).length()`
– the result of concatenation : `(s + "qsd").length()`

```
String s = "Hen Hau";
System.out.println(s.length());
```

Tableaux et chaînes

Common features:
- there are two separate phases: declaration and value creation
- after the declaration a variable contains `null`
- the explicit creation of a new string with a `new` statement
- there is a possibility of implicit creation using a special syntax
  for arrays - braces `{..}` , for strings – quotes `".."`.
- strings and arrays have a length greater than or equal to 0 ;
  it is fixed and time-invariant.

Unlike arrays for strings there is not a specific syntax for direct access to the characters in a string.

The strings are the first example of an object in this lecture.
Types of objects are as arrays the reference types, that is to say that the variables of these types contain the address of objects or arrays in memory.

Some methods on String

```
public class ExChaine2 {
  public static void main(String args[]){
  String s1 = "Hen Hau";
  String s2 ;
System.out.println("Type the string:");
s2 = Keyboard.readString();
System.out.println(s1.charAt(0));
System.out.println(s1.toUpperCase());
System.out.println(s1.compareTo(s2));
System.out.println(" "+s1.equals(s2));
}
}
```

Note: there is no method or other way to change a character in a string: the string once created, can not change. For a more or less equivalent to a change of character, create a new string by concatenating portions of the original string with a different character.

Comparing the strings

```
public class CompString{
public static void main(String args[]){
String s1,s2,s3;
boolean b1,b2;
s1 = "tati";  s2 = "ta";
s2 = s2 + "ti";  s3 = s1 ;
System.out.println("s1="+s1+"s2="+s2+"s3="+s3);
b1 = (s1==s2);
b2 = (s1==s3);
System.out.println("s1==s2?"+b1);
System.out.println("s1==s3?"+b2);
}
}
```

As for an array, the operators == and ! = compare not the content, but just their memory address.

The question is:

the two strings, are they at the same address ?

Parameters of the method `main()`

```
public class LineCommand{
  public static void main(String args[]){
    for(int i=0;i<args.length;i++){
      System.out.println(args[i]);
    }
  }
}
```

```
Java/Test> java LineCommand one 12 56 two
one
12
56
two
```

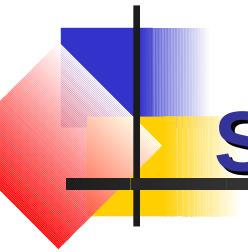
Conversion : `String` and other types

```
public class StringInt2{
  public static void main(String args[]){
    int x ;
    String s= «12»;
    x = Integer.parseInt(s);
    System.out.println(x);
  }
}
```

To convert a value of type **`double`**, use the **`Double.parseDouble`** method (s) and the type **`boolean`**, the method **`Boolean.parseBoolean`** (s).

To convert the other way, an **`int`** into a **`String`**, the easiest way is to use the concatenation operator:
`"" +12` (for double `"" +12.3` for Boolean `"" + true`).

We concatenate the data with the empty string value to convert.



Summary

Procedures with and without arguments

Overloading

Functions

Chains – `String` (our first class)

Converting between `String` and other data types