University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Object-Oriented Programming

Introduction to Java

23 février 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

- Object-Oriented Programming
- Java
- Control Structures
- Array in Java

- Object-Oriented Programming
- Java
- Control Structures
- Array in Java

Introduction

Object-oriented programing (OOP) is a programming paradigm that structures code around the concept of objects.

Object-oriented programs are often easier to understand, correct and modify.

Structured Programming

Wirth's equation :

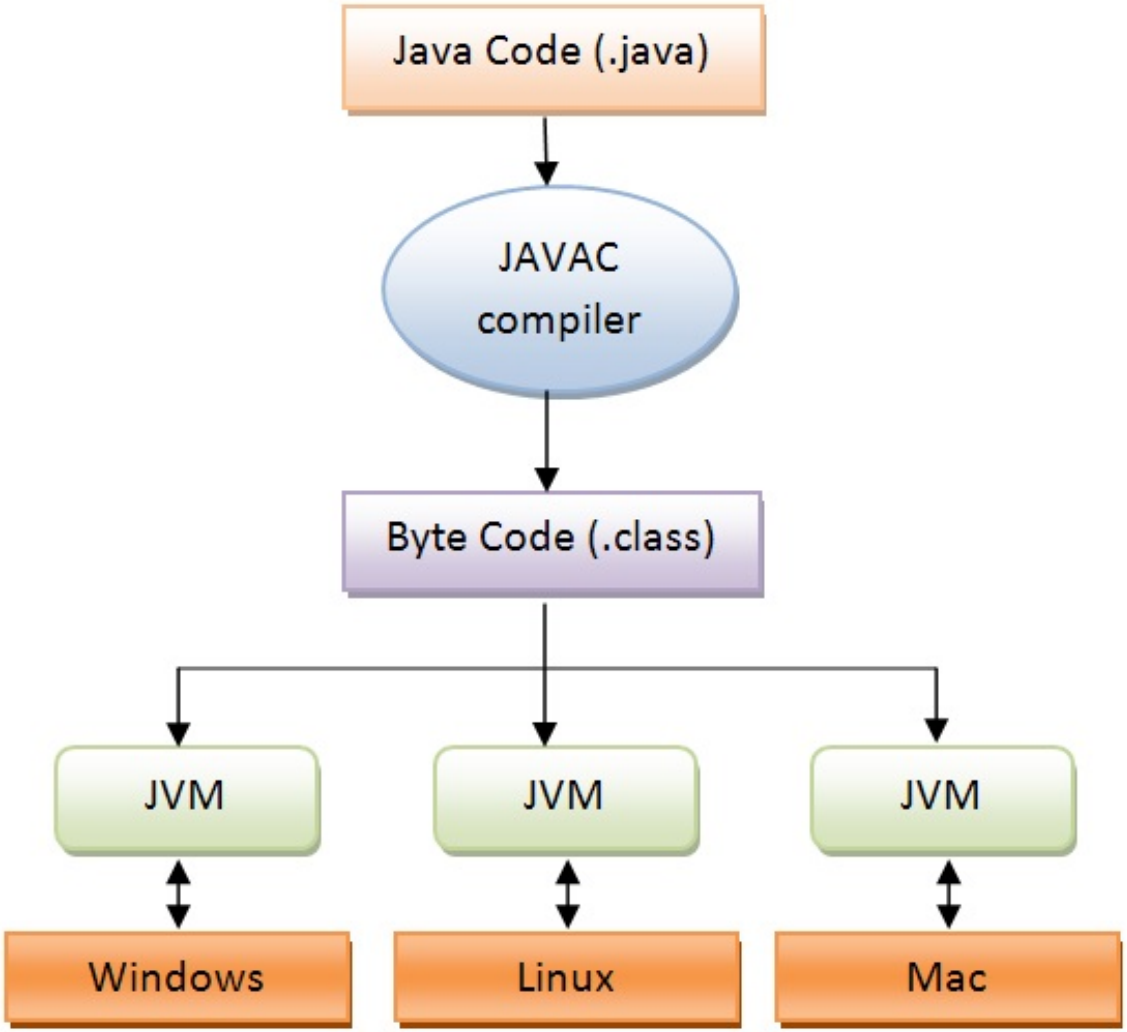
$$\text{Programs} = \text{Algorithms} + \text{Data Structures}$$

The choice of algorithms and the use of suitable data structures are the fundamental building blocks for writing software.

- Object-Oriented Programming
- Java
- Control Structures
- Array in Java

Write once, run anywhere

- Develloped by Sun Microsystems in 1991 (*James Gosling*).
- A key goal of Java is to be able to write programs that will run on a great variety of computer systems and computer-controlled devices.



Java Interpreter

IDE

There are many popular Java IDEs, including :

- Eclipse (www.eclipse.org)
- NetBeans(www.netbeans.org)
- IntelliJ IDEA (www.jetbrains.com)

First Program in Java

```
1  public class Welcome
2  {
3     /* main method begins execution of Java application
           */
4     public static void main(String[] args)
5     {
6        System.out.println("Hello World !");
7     } // end method main
8  } // end class Welcome
```

System.out.printf *:(f means "formatted") displays formatted data*

Input values

```
1  import java.util.Scanner;
2  public class Demo {
3    public static void main(String[] args) {
4      Scanner scan = new Scanner(System.in);
5      System.out.print("Enter any number: ");
6
7      int num = scan.nextInt();  // reads the number
8      scan.close();  // Closing Scanner after the use
9      System.out.println("The number entered : " + num);
10   }
11 }
```

string : nextLine() float : nextFloat

Good Programming Practices

- *Declare each variable in its own declaration. This format allows a descriptive comment to be inserted next to each variable being declared.*
- *Choosing meaningful variable names helps a program to be self-documenting.*

Entering Text in a Dialog

```
import javax.swing.JOptionPane;

public class EnteringText_InDialog {

  public static void main(String[] args) {

  String name = JOptionPane.showInputDialog("Your
        name:");

   // display the message to welcome the user by name
  JOptionPane.showMessageDialog(null, " "+ name);

  }
}
```

Good Programming Practices

Format a Code

In order to format a selected region of code or an entire file :

- Click menu : Source > Format, or
- Eclipse : CTRL + SHIFT + F
- Netbeans : ALT + SHIFT + F

Data Types in Java

Category	Data Types	Example
Primitive Data Types	byte, short, int, long, float, double, char, boolean	`int age = 30;`
Reference Data Types	String, Classes, Arrays, Interfaces, Enums, custom objects	`String s = "John";`
Derived Data Types	Arrays, Classes (created using primitive/reference types)	`int[] a={1, 2, 3};`
User-Defined Data Types	Custom classes and interfaces	`class MyClss {...}`

Data Types in Java

Data Type	Size (bits)	Range	Example
byte	8	-128 to 127	`byte b = 42;`
short	16	-32,768 to 32,767	`short s = 1000;`
int	32	-2^{31} to 2^{31} - 1	`int i = 123456;`
long	64	-2^{63} to 2^{63} - 1	`long l = 9876543210L;`
float	32	IEEE 754 single-precision	`float f = 3.14f;`
double	64	IEEE 754 double-precision	`double d = 2.71828;`
char	16	0 to 65,535 (Unicode characters)	`char c = 'X';`
boolean	-	true or false	`boolean b = true;`

Java Primitive Data Types

String

```
// Using a string literal
String str1 = "Hello, World!";

// Using the String constructor
String str2 = new String("Java");
```

Constants

A variable's value can not be changed after it has been assigned.

```
final double PI = 3.14159;
final int MAX_VALUE = 100;
```

Using the *static final* modifier combination :

```
public class Constants {
    public static final double PI = 3.14159;
    public static final int MAX_VALUE = 100;
  }
```

static : means that this variable belongs to the class itself, not to instances of the class.

Data Type Conversion

- Implicit Type Conversion (Widening) : automatic type conversion
- Explicit Type Conversion (Narrowing) : Also known as casting. converting a data value from one data type to another

Implicit conversion

A value of one data type is automatically and safely converted to another data type

1. Widening of Numeric Types
2. Promotion of Numeric Types
3. Boolean to Numeric Conversion
4. String to Numeric Conversion

Widening of Numeric Types

A smaller numeric data type is assigned to a larger numeric data type.

```
int smallerValue = 42;
long largerValue = smallerValue;
// Implicit conversion from int to long
```

Promotion of Numeric Types

Different numeric data types are mixed in an expression, the Java compiler promotes them to a common, larger data type before performing the operation.

```
int num = 5;
double result = num + 3.5;
//Implicit conversion of int to double for addition
```

Boolean to Numeric Conversion

In some cases, boolean values can be implicitly converted to numeric values (1 for true and 0 for false).

```
boolean flag = true;
int num = flag ? 1 : 0;
// Implicit conversion from boolean to int
```

Numeric to String Conversion

When you use the + operator to concatenate a String with a numeric value, the numeric value is implicitly converted to a String and then concatenated.

```
String str = "The answer is: ";
int answer = 42;
String result = str + answer;
// Implicit conversion of int to String
```

Explicit Type Conversion

- Converting a value from one data type to another.
- Specifying the target data type explicitly using casting operators. Explicit type conversion is used when you need to convert a larger data type to a smalle

```
double doubleValue = 1000.75;
int intValue = (int) doubleValue;
// Explicit casting from double to int
```

Conversion should be used judiciously, and programmers should be aware of the potential consequences and limitations when performing such conversions (Data Loss, Range Limitation, ...)

Parsing

Parsing refers to the process of extracting meaningful information or values from a textual representation, such as a string.

```
String strNumber = "42";
int number = Integer.parseInt(strNumber);

String strValue = "3.14159";
double value = Double.parseDouble(strValue);

String dateString = "2024-02-03";
SimpleDateFormat dateFormat = new
      SimpleDateFormat("yyyy-MM-dd");
Date date = dateFormat.parse(dateString);
```

- Object-Oriented Programming
- Java
- Control Structures
- Array in Java

if-else statement

The ***if-else*** statement is the most basic way to control program flow. The ***else*** is optional, so you can use ***if*** in two forms :

```
if(Boolean-expression)
  statement
```

or

```
if(Boolean-expression)
    statement
else
  statement
```

Executes one block of code if a specified condition is true and another block of code if the condition is false

Nested ifs

- A ***nested if*** is an ***if statement*** that is the target of another ***if*** or ***else***
- An ***else statement*** always refers to the nearest ***if statement*** that is within the same block

```
if(i == 10) {
  if(j < 20) a = b;
        if(k > 100) c = d; // this if is
              else a = c; // associated with this else
}
else a = d; // this else refers to if(i == 10)
```

if-else-if Ladder

A series of if statements followed by an ***else*** block, allowing for the evaluation of multiple conditions in sequence.

```
if(condition)
  statement;
  else if(condition)
      statement;
    else if(condition)
          statement;
        .
        .
        else
          statement;
```

Slection Statements Example

```
int num = 10;
int grade = 85;
if (num % 2 == 0)    // if-else statement
  System.out.println("Number is even");
else
  System.out.println("Number is odd");

      // else-if ladder
if (grade >= 90)
  System.out.println("Excellent!");
 else if (grade >= 80)
        System.out.println("Very good!");
      else if (grade >= 70)
             System.out.println("Good!");
           else
            System.out.println("Needs
                  improvement!");
```

switch statement

Multiway branch statement, execute one block of code from multiple options based on the value of an expression

```
switch (expression) {
  case value1: // code, if expression == value1
          break;
  case value2: // code, if expression ==
        value2
          break;
   // ... more cases
  default: // code to be executed if no match
        found
}
```

For versions of Java prior to JDK 7, expression must be of type byte, short, int, char, or an enumeration. Beginning with JDK 7, expression can also be of type String.

```
int day = 3;
String dayName;
switch (day) {
  case 1:  dayName = "Sunday";
           break;
  case 2: dayName = "Monday";
           break;
  case 3:  dayName = "Tuesday";
           break;
  case 4:  dayName = "Wednesday";
           break;
  case 5:  dayName = "Thursday";
           break;
  case 6:  dayName = "Friday";
          break;
  case 7:  dayName = "Saturday";
           break;
  default: dayName = "Invalid day";
}
```

Iteration Statements

- ***for*** loop : Executes a block of code a specified number of times.
- ***while*** loop : Executes a block of code as long as a specified condition is true.
- ***do-while*** loop : Executes a block of code at least once and then repeatedly executes the block as long as a specified condition is true.

"for-each" loop

```
public class ForEachLoopExample {

    public static void main(String[] args) {

        int[] numbers = {1, 2, 3, 4, 5};

        /* Using a for-each loop to print each element
              of the array*/
        for (int num : numbers) {
          System.out.println(num);
        }
    }
}
```

break statement

- Terminates the loop or switch statement and transfers control to the statement immediately following the loop or switch.
- continue statement : Skips the current iteration of a loop and proceeds to the next iteration.
- return statement : Exits the current method and optionally returns a value.

"break - example

Example in a for loop :

```
for (int i = 0; i < 10; i++) {
  if (i == 5)
    break; // Terminates the loop when i is
  System.out.println(i);
}
```

Example in a while loop :

```
int i = 0;
while (i < 10) {
  if (i == 5)
     break; // Terminates the loop when i is 5
  System.out.println(i);
  i++;
}
```

continue statement

continue

- Used within looping constructs (**for**, **while**, and **do-while** loops) to skip the current iteration of the loop and proceed to the next iteration.
- Unlike the break statement, which exits the loop entirely, continue merely skips the remaining code in the loop for the current iteration and then continues with the next iteration of the loop.

continue - example

Example in a for loop :

```
for (int i = 1; i <= 10; i++) {
  if (i == 5)
    continue; // Skip the rest of the loop body for i
          == 5
  System.out.println(i);
}
```

Example in a while Loop :

```
int i = 0;
while (i < 10) {
  i++; // Increment i at the beginning to avoid
       infinite loop
  if (i == 5) continue; // Skip printing 5
    System.out.println(i);
}
```

- Object-Oriented Programming
- Java
- Control Structures
- Array in Java

Array in Java

- In Java, an array is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created and cannot be changed after creation.
- Each item in an array is called an element, and each element is accessed by its numerical index, with the first element's index being **0**.

Creating Arrays

Creating Arrays :

```
int[]  myIntArray = new int[10]; // An array of 10
       integers;
String[] myStringArray = new String[5]; // An array of
       5 Strings
```

You can also initialize the array at the time of creation by enclosing the initial values in curly braces .

```
int[] myIntArray = {1, 2, 3, 4, 5};
String[] myStringArray = {"Hello", "World"};
```

Creating Arrays

Creating Arrays :

```
int[]  myIntArray = new int[10]; // An array of 10
       integers;
String[] myStringArray = new String[5]; // An array
       of 5 Strings
```

You can also initialize the array at the time of creation by enclosing the initial values in curly braces .

```
int[] myIntArray = {1, 2, 3, 4, 5};
String[] myStringArray = {"Hello", "World"};
```

Accessing Array Elements

Accessing Array Elements : Array indexes start at 0. So, the first element of an array is at index 0, the second is at index 1, and so on.

```
int firstElement = myIntArray[0]; // Access the first
      element
myIntArray[4] = 100; // Assign a value to the fifth
      element
```

Array Length : The length property of an array is used to find out the size of an array.

```
int arraySize = myIntArray.length;
```

Looping Through Arrays

You can loop through an array using a for loop or an enhanced for loop (also known as the "for-each" loop).

```
// Using a for loop
for (int i = 0; i < myIntArray.length; i++) {
  System.out.println(myIntArray[i]);
}
```

```
// Using an enhanced for loop
for (int element : myIntArray) {
  System.out.println(element);
}
```

Multidimensional Arrays

Java supports multidimensional arrays, which are arrays of arrays. The most common type is the two-dimensional array.

```
int[][] my2DArray = new int[10][20]; // A 2D array
      of size 10x20

my2DArray[0][0] = 1; // Assign a value to the
      first element

int[][] my2DArrayInitialized = {{1, 2}, {3, 4}};
      // Initialization
```

Array in Java

- Arrays have a fixed size and cannot grow or shrink once created.
- They can hold only one type of data.
- For more flexible operations like inserting, deleting, or resizing, consider using Java Collections Framework classes such as **ArrayList**.

```java
public class BubbleSort {
  public static void bubbleSort(int[] arr) {
    int n = arr.length;
    boolean swapped;
    for (int i = 0; i < n - 1; i++) {
      swapped = false;
      for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
          // swap arr[j] and arr[j+1]
          int temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
          swapped = true;
        }
      }
      if (!swapped)  // IF no two elements
        break;       //were swapped, then break
    }
  }
```

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

      int[] numbers = {64, 34, 25, 12, 22, 11, 90};

        bubbleSort(numbers);
      System.out.println("Sorted array: ");

      for (int number : numbers) {
       System.out.print(number + " ");
      }
    }
}
```

Questions ?