Badji Mokhtar University - ANNABA
Faculty of Technology
Department of Computer Science

Course

Object-Oriented Programming: Application to the Java Language

Presented by: Dr. Hariati
Level: Second-year Bachelor's in Computer Science
Academic Year: 2024 - 2025

Object-Oriented Programming: Application to the Java Language

A Brief Introduction to Object-Oriented Programming

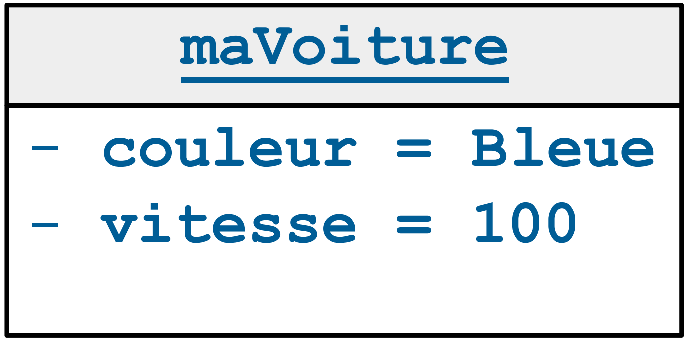
Structured Programming VS OOP

- Objectives of OOP
  - Facilitates code reuse, encapsulation, and abstraction
  - Facilitates code evolution
  - Improves the design and maintenance of large systems
  - Component-based programming. Software design in the manner of car manufacturing

- Structured Programming
  - Logical unit: the module
  - A section for variables
  - A section for functions
  - Each function solves a part of the problem
  - "Top-down" structuring of the program

OOP principles: Object-Based Programming

- Logical Unit: The Object
- An object is defined by
    - A state
    - A behavior
    - An identity



maVoiture

- couleur = Bleue

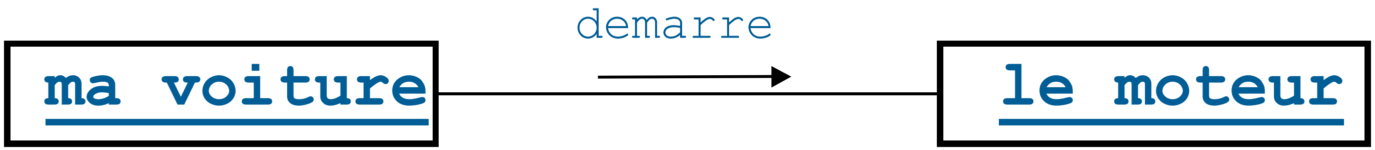
- vitesse = 100

- State: Represented by attributes (variables) that store values
- Behavior: Defined by methods (procedures) that modify states
- Identity: Allows distinguishing one object from another

OOP Principles

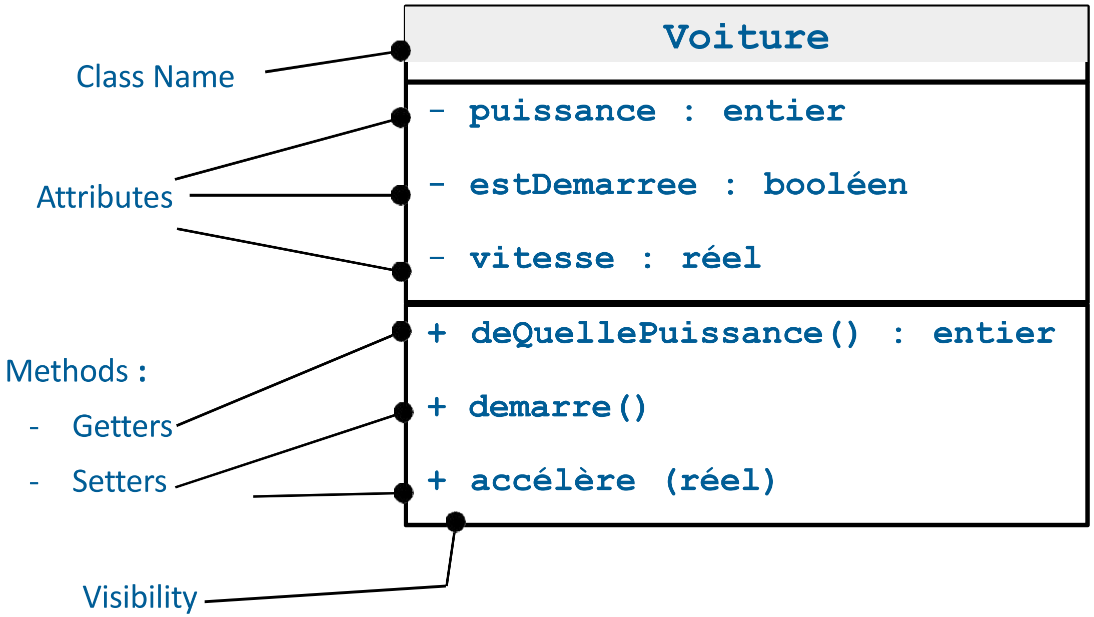
- Objects communicate with each other through messages

- An object can receive a message that triggers
  - a method that modifies its state and/or
  - a method that sends a message to another object



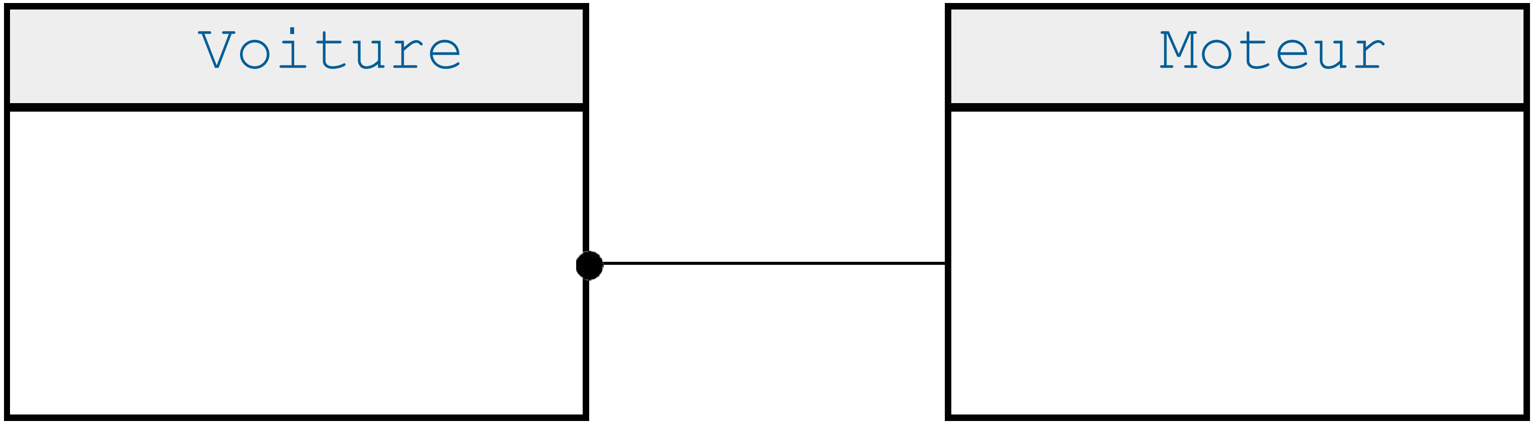
OOP Principles: Concept of Class

- Objects that have the same states and behaviors are grouped: this is a class
- Classes act as "templates" for creating objects. An object is an instance of a class.
- An OO (Object-Oriented) program consists of classes that allow the creation of objects that exchange messages.



OOP Principles

- The set of interactions between objects defines an algorithm.
- The relationships between classes reflect the decomposition of the program.



Object-Oriented Programming: Application to the Java Language

Introduction to the Java language

Java Functioning Principle

- Java Source
  - File used during the programming phase
  - The only file that is truly readable by the programmer!

- Java Bytecode
  - Object code intended to be executed on any "Java Virtual Machine“
  - Generated from the compilation of the source code

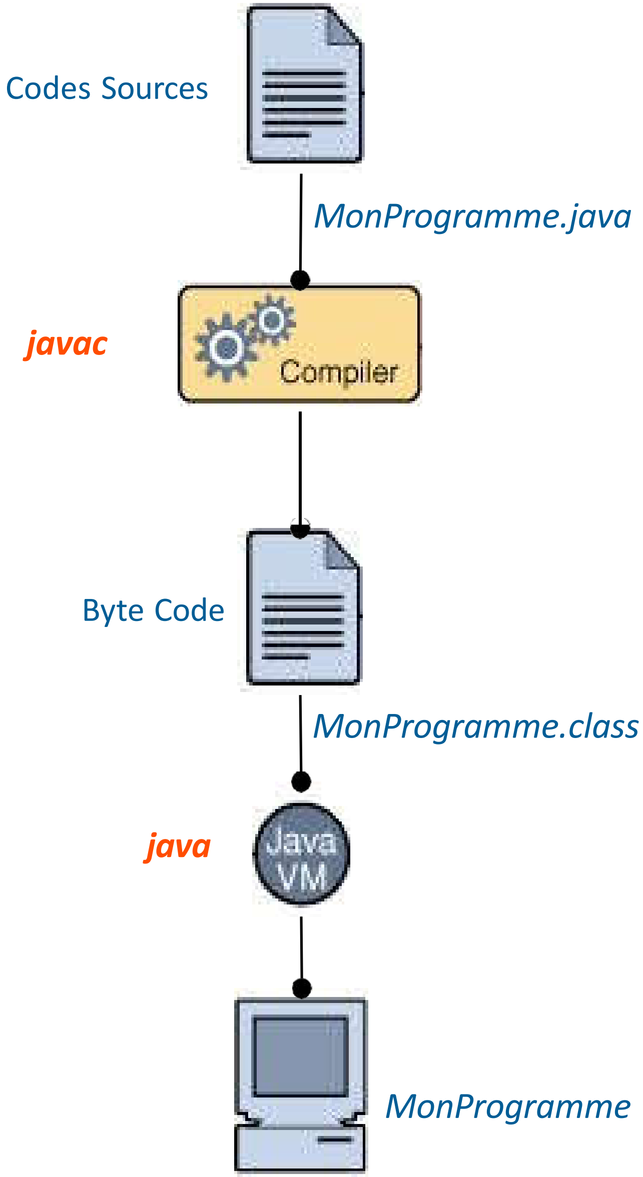
- Java Virtual Machine
  - Program that interprets Java Bytecode and runs on a specific operating system
  - Conclusion: It is sufficient to have a "Java Virtual Machine" to execute any Java program, even if it was compiled on a different operating system

Java Virtual Machines

- Web Browsers, Workstations, Network Computers
- WebPhones
- Mobile Phones
- Smart Cards
- …

Main Steps of a Development

- Source Code Creation
  - From specifications (for example, in UML)
  - Tool: text editor, IDE
- Compilation into Byte-Code
  - From the source code
  - Tool: Java compiler
- Deployment on the target architecture
  - Transfer of Byte-Code only
  - Tools: network, disk, etc.
- Execution on the target machine
  - Execution of Byte-Code
  - Tool: Java Virtual Machine



Java and its versions...

- Different versions of the virtual machine
  - Java Micro Edition (Java ME), which targets portable devices
  - Java Standard Edition (Java SE), which is aimed at client machines
  - Java Enterprise Edition (Java EE), which defines the framework for an application server

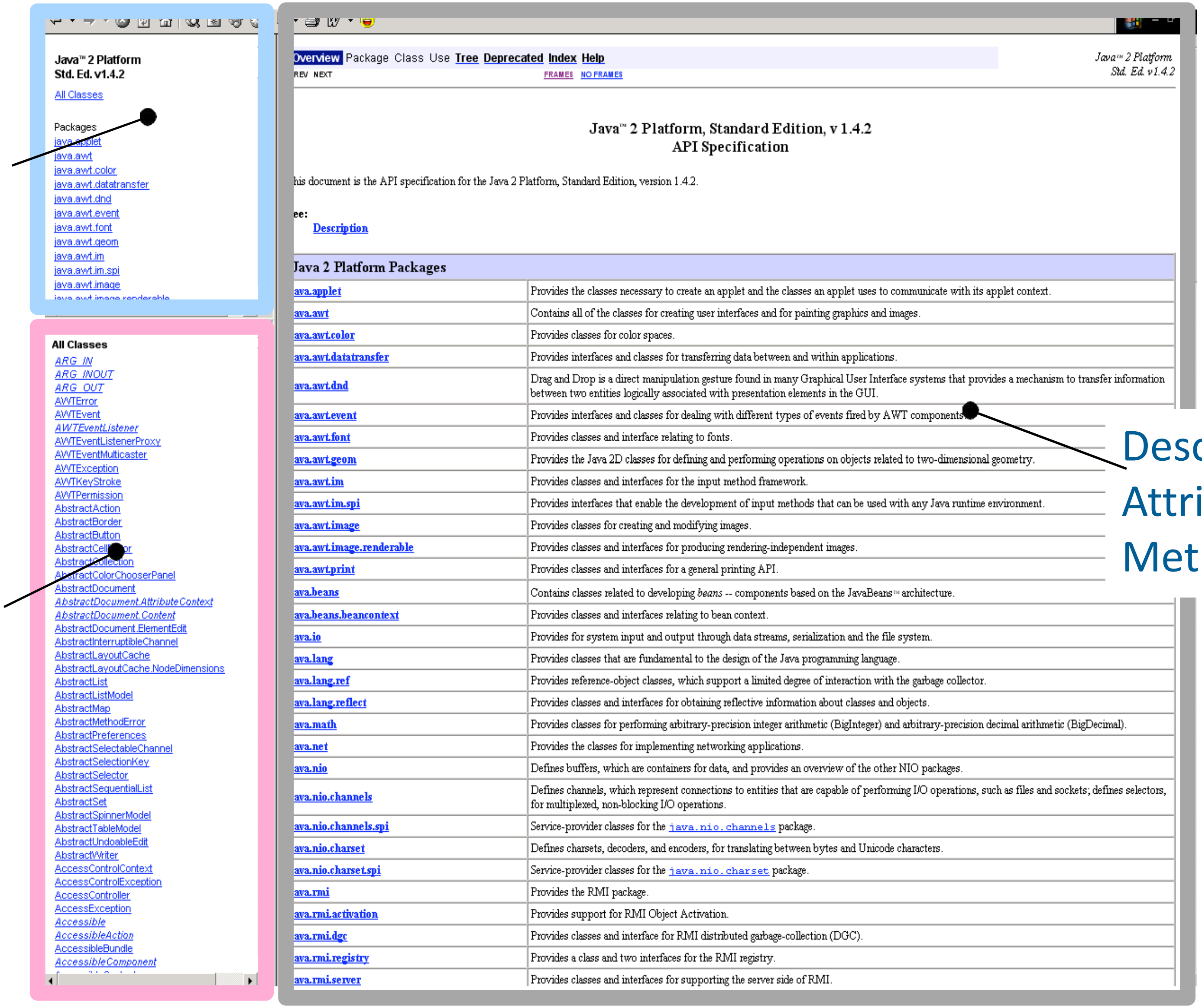
In the rest of the course, we will mainly focus on the APIs provided by Java SE

- Different purposes
  - SDK (Software Development Kit) provides a compiler and a virtual machine
  - JRE (Java Runtime Environment) only provides a virtual machine. Ideal for deploying your applications.

Tools…

- Simple editors or IDEs (Integrated Development Environments):
  - Eclipse
  - NetBeans
  - IntelliJ
  - …

The Java API



Packages

Classes

Description
Attributes
Methods

Java™ 2 Platform
Std. Ed. v1.4.2

All Classes

Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderable

All Classes
ARG_IN
ARG_INOUT
ARG_OUT
AWTError
AWTEvent
AWTEventListener
AWTEventListenerProxy
AWTEventMulticaster
AWTException
AWTKeyStroke
AWTPermission
AbstractAction
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMethodError
AbstractPreferences
AbstractSelectableChannel
AbstractSelectionKey
AbstractSelector
AbstractSequentialList
AbstractSet
AbstractSpinnerModel
AbstractTableModel
AbstractUndoableEdit
AbstractWriter
AccessControlContext
AccessControlException
AccessController
AccessException
Accessible
AccessibleAction
AccessibleBundle
AccessibleComponent

Java™ 2 Platform
Std. Ed. v1.4.2

Java™ 2 Platform, Standard Edition, v 1.4.2
API Specification

his document is the API specification for the Java 2 Platform, Standard Edition, version 1.4.2.

ee:
Description

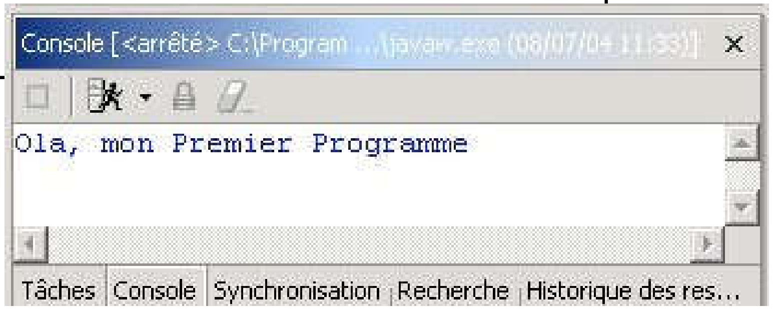
Java 2 Platform Packages	
ava.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
ava.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
ava.awt.color	Provides classes for color spaces.
ava.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
ava.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
ava.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
ava.awt.font	Provides classes and interface relating to fonts.
ava.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
ava.awt.im	Provides classes and interfaces for the input method framework.
ava.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
ava.awt.image	Provides classes for creating and modifying images.
ava.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
ava.awt.print	Provides classes and interfaces for a general printing API.
ava.beans	Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture.
ava.beans.beancontext	Provides classes and interfaces relating to bean context.
ava.io	Provides for system input and output through data streams, serialization and the file system.
ava.lang	Provides classes that are fundamental to the design of the Java programming language.
ava.lang.ref	Provides reference-object classes, which support a limited degree of interaction with the garbage collector.
ava.lang.reflect	Provides classes and interfaces for obtaining reflective information about classes and objects.
ava.math	Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal).
ava.net	Provides the classes for implementing networking applications.
ava.nio	Defines buffers, which are containers for data, and provides an overview of the other NIO packages.
ava.nio.channels	Defines channels, which represent connections to entities that are capable of performing I/O operations, such as files and sockets; defines selectors, for multiplexed, non-blocking I/O operations.
ava.nio.channels.spi	Service-provider classes for the `java.nio.channels` package.
ava.nio.charset	Defines charsets, decoders, and encoders, for translating between bytes and Unicode characters.
ava.nio.charset.spi	Service-provider classes for the `java.nio.charset` package.
ava.rmi	Provides the RMI package.
ava.rmi.activation	Provides support for RMI Object Activation.
ava.rmi.dgc	Provides classes and interface for RMI distributed garbage-collection (DGC).
ava.rmi.registry	Provides a class and two interfaces for the RMI registry.
ava.rmi.server	Provides classes and interfaces for supporting the server side of RMI.

Object-Oriented Programming: Application to the Java Language

Language Basics

First example of a program in Java

```
public class PremierProg {

    public static void main (String[] args)  {
        System.out.println("Ola, mon Premier Programme");
    }
}
```


Console [<arrêté> C:\Program ...\javaw.exe (08/07/04 11:33)]
Tâches
Console
Synchronisation
Recherche
Historique des res...

```
Ola, mon Premier Programme
```

- *public class PremierProg*
    - Class name
- *public static void main*
    - The main function, equivalent to the main function in C/C++
- *String[] args*
    - Allows retrieving arguments passed to the program at runtime
- *System.out.println("Ola … ")*
    - Display method in the console window

Implementation

- No separation between definition and implementation of operations
  - A single file "ClassName.java”
  - No header file like in C/C++

Class name = Java file name

- Compilation
  - javac NomDeClasse.java or javac *.java when multiple classes
  - Generation of a Byte-Code file « NomDeClasse.class »
  - No linking (only a verification)

Do not include the .class extension for execution

- Execution
  - java NomDeClasse
  - Select the main class to execute

Java Primitive Types

- Are not objects!!!
- Occupy a fixed memory space reserved at declaration
- Primitive types
  - Integers: **byte** (1 byte) - **short** (2 bytes) - **int** (4 bytes) - **long** (8 bytes)
  - Floating points (IEEE-754 standard): **float** (4 bytes) - **double** (8 bytes)
  - Booleans: **boolean** (true or false)
  - Characters: **char** (Unicode encoding on 16 bits)
- Each simple type has an object counterpart with conversion methods (see the Classes and Objects section)
- Autoboxing introduced since version 5.0 transparently converts primitive types into references

Initialization and Constants

- Initialization
  - A variable can receive a value at the time of its declaration :

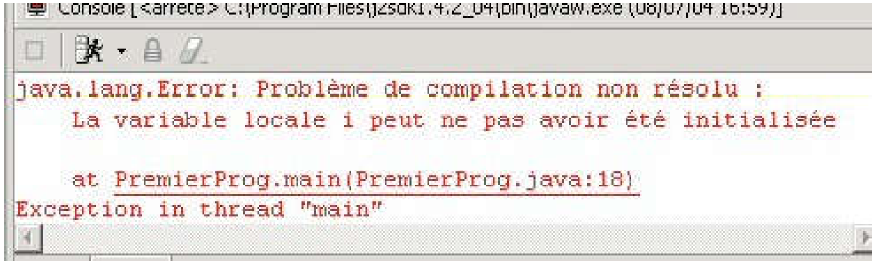
```
int n = 15;
boolean b = true;
```

  - This instruction plays the same role:

```
int n;

n = 15;
boolean b;
b = true;
```

Think about initialization to avoid a compilation error

```
 int n;
System.out.println(" n = " + n);
```


Console [<arrete> C:\Program Files\j2sdk1.4.2_04\bin\javaw.exe (08/07/04 16:59)]

```
java.lang.Error: Problème de compilation non résolu :
    La variable locale i peut ne pas avoir été initialisée

    at PremierProg.main(PremierProg.java:18)
Exception in thread "main"
```

- Constants
  - These are variables whose value can only be assigned once
  - They can no longer be modified
  - They are defined with the keyword **final**

```
final int n = 5;
final int t;
...
t = 8;
n = 10; // Error: n is declared final
```

Control structures

There is no **then** keyword in the Choice structure

- Choice
  - *If then else: « **if** condition {…} **else** {…} »*
- *Iterations*
  - *Loop: « **for** (initialization; condition; modification) { … } »*
  - *Loop (for each): « for (Type var : Collection) { … } »*
  - *While: « **while** (condition) { … } »*
  - *Do until: « **do** { … } **while** (condition) »*
- *Bounded selection*
  - ***Switch** case: « switch ident { **case** value0 : … **case** value1 : … **default**: …} »*
  - *The keyword **break** requests to exit the block.*

Remember to check if **break** is necessary in each **case**

Control structures

- Example: control structure

```
public class SwitchBreak {

    public static void main (String[] argv)
       { int n = ...;
       System.out.println("Valeur de n :" +
       n); switch(n) {
           case 0 : System.out.println("nul");
                    break;
           case 1 :
           case 2 : System.out.println("petit");
           case 3 :
           case 4 :
           case 5 : System.out.println("moyen");
                    break;
           default : System.out.println("grand");
       }
       System.out.println("Adios...");
    }
}
```

- Let's vary n

```
Value of      n :  0
nul
Adios...

Value of      n :  1
petit

Moyen

Adios...

Value of      n :  6
grand
Adios...
```

Ask yourself if break is necessary.

Operators on Primitive Types

- Arithmetic Operators
  - Unary: « +a, -b »
  - Binary: « a+b, a-b, a*b, a%b »
  - Increment and Decrement: « a++, b-- »
  - Compound Assignment: « +=, -=, *=, /= »

- Comparison Operators
  - « a==b, a!=b, a>b, a<b, a>=b, a<=b »

- Logical Operators
  - AND: « a && b", "a & b »
  - OR: « a || b", "a | b »

- Explicit Type Conversion (Casting)
  - « (NewType)variable »

Warning: Error

```
boolean t = true;
  if (t == true) {...}
```

Prefer:

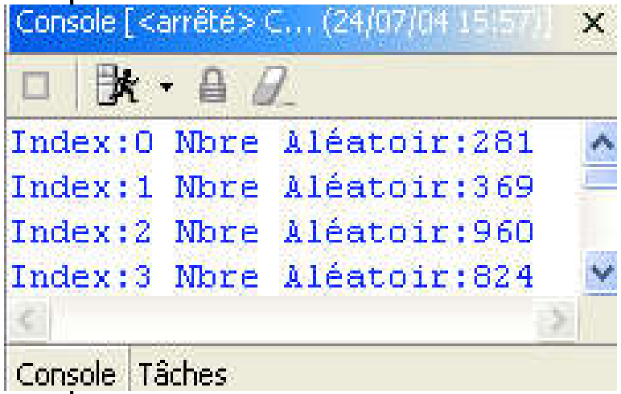
```
boolean t = true;

if (t) {...}
```

Operators on Primitive Types

- Example: Lottery Simulation
  - Not optimized but demonstrates the use of previous concepts

```
public class ExempleTypesPrimitifs {

    public static void main (String[] argv)  {
        int compteur = 0;

        while(compteur != 100)  {
            // Prend un nombre aléatoire
            double nbreAleatoir = Math.random()  * 1000;

            // Etablie un index de 0 à 10
            int index = compteur % 10;

            // Construction de l'affichage
            System.out.println("Index:" + index +
                "Nbre Aléatoir:" + (int)nbreAleatoir);

            // Incrémentation de la boucle
            compteur+= 1;
        }
    }
}
```

To be seen later…

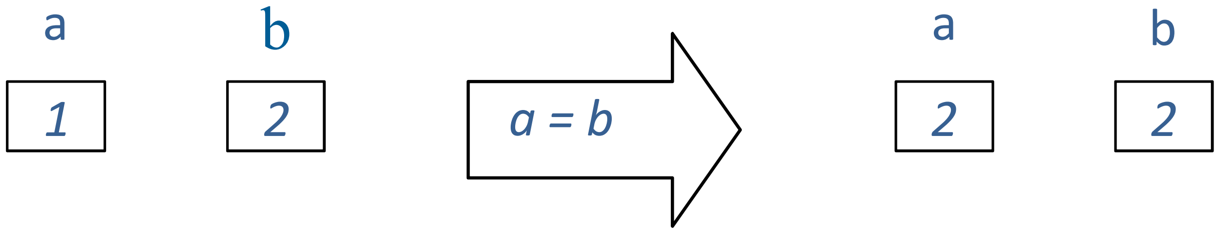


```
Index:0 Nbre Aléatoir:281
Index:1 Nbre Aléatoir:369
Index:2 Nbre Aléatoir:960
Index:3 Nbre Aléatoir:824
```

Assignment, Copying, and Comparison

- Assign and copy a primitive type
  - “a = b” means a takes the value of b
  - a and b are distinct
  - Any modification of a does not affect b

- Compare a primitive type
  - “a == b” returns "true" if the values of a and b are identical



Arrays in Java

- Arrays are considered as **objects**

- Provide ordered collections of elements

- The elements of an array can be:
    - Variables of a primitive type (int, boolean, double, char, …)
    - References to objects (to be discussed in the Classes and Objects section)

- Creating an array

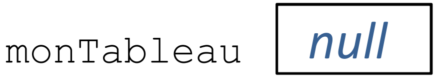
    ① Declaration = defining the type of the array
    ② Sizing = determining the size of the array
    ③ Initialization = initializing each element of the array

Arrays in Java: Declaration

① Declaration

- The declaration simply specifies the type of the array elements

```
int[] monTableau;
```



- Can also be written

```
int monTableau[];
```

Warning: An array declaration must not specify dimensions

```
int monTableau[5]; // Error
```

Arrays in Java: Sizing

② Sizing

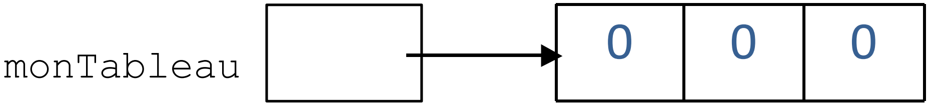
- The number of elements in the array is determined when the array object is actually created using the **new** keyword.

- The size set at the array's creation is fixed and cannot be changed later.

- Array length: « monTableau.**length** »

```
int[] monTableau; // Déclaration  monTableau =
new int[3]; // Dimensionnement
```

- Creating an array using **new**
  - Allocates memory based on the array type and size
  - Initializes the array content to 0 for primitive types



Arrays in Java: Initialization

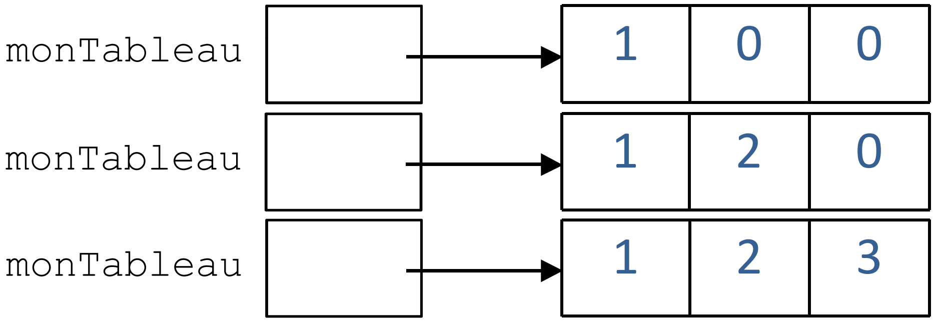
③ Initialization

- As in C/C++, indices start at zero.
- Accessing an element in an array follows this format.

```
monTab[varInt]; // varInt >= 0 et <monTab.length
```

- Java automatically checks the index when accessing an element (exception raised).

```
monTab[0] = 1;

monTab[1] = 2;

monTab[2] = 3;
```



- Another method: explicitly providing the list of its elements within {…}

```
int[] monTab = {1, 2, 3}
```

- is equivalent to

```
monTab = new int[3];
monTab[0] = 1; monTab[1] = 2; monTab[2] = 3;
```

Arrays in Java: Summary

① Declaration

```
int[] monTableau;
```

② Sizing

```
monTableau = new int[3];
```

③ Initialization

```
monTableau[0] = 1;
monTableau[1] = 2;
monTableau[2] = 3;
```

Or ①② and ③

```
int[] monTab = {1, 2, 3};
```

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

```
for (int current : monTableau) {
    System.out.println(curent);
}
```

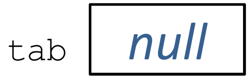
Same thing using the *for each loop*

Arrays in Java: Multidimensional Arrays

- Arrays whose elements are themselves arrays

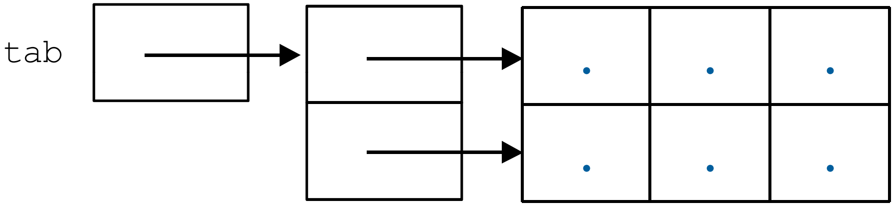
- Declaration

```
    type[][] monTableau;
```



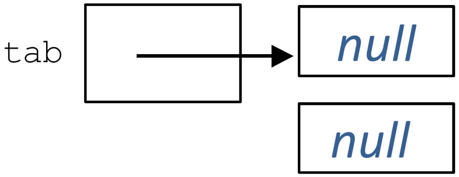
- Rectangular arrays
  - Sizing:

```
   monTableau = new type[2][3]
```

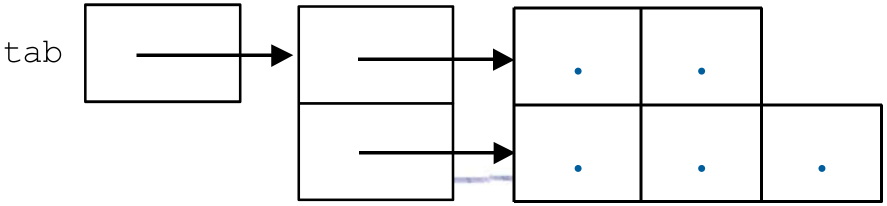


- Non-rectangular arrays
  - Sizing

```
monTableau = new type[2]
```



```
    monTableau[0] = new type[2]
    monTableau[1] = new type[3]
```



Small clarification about *"System.out.println(…)"*

- Usage: Display on screen
  - “System.out.println(…)”: moves to the next line
  - “System.out.print(…)”: does not move to the next line

- Different possible outputs
  - “out”: standard output
  - “err”: error output

- Everything that can be displayed…
  - Objects, numbers, booleans, characters, etc.

- Everything that can be done…
  - Wild concatenation between types and objects using “+”

```
System.out.println("a=" + a + "donc a < 0 est " + a < 0);
```

Comments and Formatting

- Source Code Documentation
  - Using Comments

```
// Comment on a full line
int b = 34; // Comment after some code

/* Beginning of the comment
** I can keep writing …
Until the compiler finds this */
```

  - Using the Javadoc tool (see the Javadoc section)

- Formatting
  - Facilitates proofreading
  - Ensures credibility!!!!
  - Indentation at each block level

```
if (b == 3) {
    if (cv == 5)
        { if (q) {
            ...
        } else {
            ...
        }
        ...
    }
    ...
}
```

Prefer

```
if (b == 3) {
if (cv == 5) {
if (q) {
...
} else {...}
...
}
...
}
```

Avoid