Introduction to Java language

UA.DETI.POO

Java..?

Feb 2020	Feb 2019	Change	Programming Language	Ratings	Change
1	1		Java	17.358%	+1.48%
2	2		C	16.766%	+4.34%
3	3		Python	9.345%	+1.77%
4	4		C++	6.164%	-1.28%
5	7	^	C#	5.927%	+3.08%
6	5	v	Visual Basic .NET	5.862%	-1.23%
7	6	v	JavaScript	2.060%	-0.79%
8	8		PHP	2.018%	-0.25%
9	9		SQL	1.526%	-0.37%
10	20	^^	Swift	1.460%	+0.54%
11	18	^^	Go	1.131%	+0.17%
12	11	v	Assembly language	1.111%	-0.27%
13	15	^	R	1.005%	-0.04%

https:// www.tiobe.com/ tiobe-index/

programming paradigms

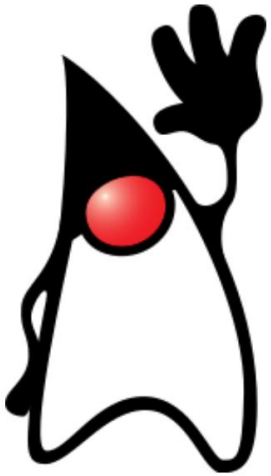
- Programming languages are based on abstractions.
  - Structured
  - imperative
  - Functional
  - Modular
  - Data Type Abstraction (ADT)
  - object oriented
- A programming paradigm determines the abstraction that the programmer can establish about the structuring and execution of the program.

What is Object Orientation?

- Most common programming paradigm
  - Affects analysis, design (design), and programming
- Object-oriented analysis
  - Determines **what** the **system** should do: Which actors are involved? What are the activities to be carried out?
  - Decomposes the system into **objects:** What are they? What tasks will each object have to do?
- Object-oriented design
  - Defines **how** the system will be implemented
  - Models the relationships between objects and actors (you can use a specific language like UML)
  - Uses and reuses abstractions such as classes, objects, functions, frameworks, APIs, design patterns

The Java language

- Java is one of the object-oriented languages – It also supports other paradigms (structured, imperative, generic, concurrent, reflective).
- It was developed in the 1990s by *Sun Microsystems.*
  - Syntax similar to C/C++
- In 2008, it was acquired by *Oracle.*
- Official page: – https://www.java.com

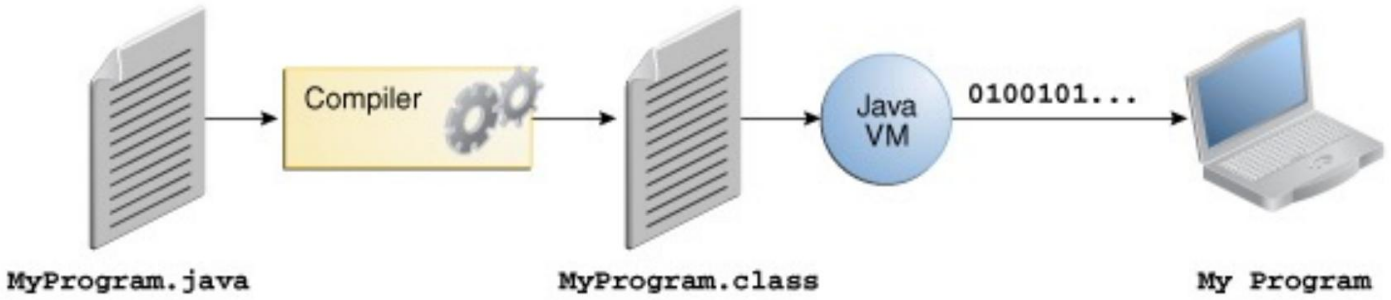


General features

- Open source software, available under the terms of the GNU General Public License
- Ease of internationalization (supports UNICODE characters natively)
- Vast set of libraries
- Facilities for creating distributed programs and multitasking
- Automatic freeing of memory by garbage collector process (garbage collector)
- Dynamic code loading
- Portability

Write and run programs

- All source code is written in plain text files ending with the .java **extension.**
  - Are compiled with the **javac** compiler to files **.class.**
- A .class file contains bytecode that is executed by a virtual machine. – does not contain native code of the processor – Runs on top of a Java Virtual Machine instance - JVM.



Java Virtual Machine

- **Advantages => great portability**
  - The JVM is a program that loads and executes the Java applications by converting the bytecodes into native code.
  - Thus, these programs are independent of the platform on which they work.
  - The same .class file can be run on different machines (running Windows, Linux, Mac OS, etc.).
- **Disadvantage => lower performance**
  - Code is slower compared to running native code (eg written in C or C++).

Basic structure of a Java program

- What in other languages is called a main program is in Java a class declared as a **public class** in which we define a function called **main()**
  - Declared as public static void
  - With an args parameter, of type String[]
- This is the default format, absolutely fixed

```
// inclusão de pacotes/classes externas
//    o pacote java.lang é incluído automaticamente

public class Exemplo {
      //    declaração de dados que compõem a classe
      // declaração e implementação de métodos
      public static void main(String[] args) {
            /* início do programa */
      }
}
```

simple example

```
package class01;

public class MyFirstClass {

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

```
Hello Eclipse!
```

Variables and primitive types

- If we want to store data we need to define variables, with a given type
  - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Type	Size	Range	Default
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	$[-2^{63}, 2^{63}-1]$	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

Example with primitive types

```
package class01;

public class Tests {

    public static void main(String[] args) {
        boolean varBoolean = true; char
        varChar = 'A';
        byte varByte = 100;
        double varDouble = 34.56;

        System. out.println(varBoolean);
        System. out.println(varChar);
        System. out.println(varByte);
        System. out.println(varDouble);
    }
}
```

```
true

THE
100
34.56
```

Declaring and Initializing Variables

- Local variables can be initialized in a variety of ways.
  shapes:
  - at the time of definition:
    ```
    double weight = 50.3; int
    day = 18;
    ```
  - using an assignment statement (symbol '='): `double weight;`
    ```
    weight = 50.3;
    ```
  - reading a value from the keyboard or another device:
    ```
    double km;
    km = sc.nextDouble();
    ```

operators

- Operators take one, two, or three arguments and produce a new value.

- Java includes the following operators: – assignment: = – arithmetic: *, /, +, -, %, ++, -- – relational: <, <=, >, >=, ==, != – logical: !, ||, && – bit manipulation: &, ~, |, ^, >>, << – ternary decision operator ?

Expressions with operators

- assignment

```
int a = 1; // a takes the value 1
int b = a; // b takes the value of variable a
a = 2; // a gets the value 2, b has the value 1
```

- Arithmetic

```
double x = 2.5 * 3.75 / 4 + 100; // priority?
double y = (2.5 * 3.75) / (4 + x); int num =
57 % 2; // remainder of division by 2
```

- Relational

```
boolean res = (x >= y);
boolean e = (x == y); // and <- "x equals y"?
```

- Logical

```
char code = 'F';
boolean capitalLetter = (code >= 'A') && (code <= 'Z');
```

Unary Arithmetic Operators

- The unary increment operators (++) and decrement (--) can be used with numeric variables.

- When placed before the operand they are pre-increment (++x) or pre-decrement (--x). – the variable is first changed before being used.
- When placed after the operand they are post-increment (x++) and post-decrement (x--)
  - the variable is first used in the expression and then changed.

```
int a = 1;
int b = ++a; // a = 2, b = 2
int c = b++; // b = 3, c = 2
```

Constants / Literals

- Literals are invariant values in the program

  23432, 21.76, false, 'a', "Text", ...

- Usually the compiler knows how to determine its type and interpret it.

```
int x = 1234;
char ch = 'Z';
```

- In ambiguous situations we can add special characters: –

  l/L = long, **f/F** = float, **d/D** = double

  – 0x/0Xvalue = hexadecimal value
  – 0value = octal value.

```
long a = 23L;
double d = 0.12d;
float f = 0.12f; // mandatory
```

Variable type conversion

- We can store a value with less storage capacity in a variable with greater storage capacity
- The respective conversion will be done automatically:
  - byte -> short (or char) -> int -> long -> float -> double
- Reverse conversion generates a compilation error.
  - However, we can always perform an explicit conversion using a conversion operator:

```
int a = 3;
double b = 4.3;
double c = a; // automatic conversion from int to double
a = (int) b; // b is forcibly converted/truncated to int
```

Print variables and literals

- System.out.println(...);
  - write whatever is between (..) and change the line
- System.out.print(...);
  - writes whatever is between (..) and does not change lines
- Examples

```
String name = "Adriana";
int x = 75;
double r = 19.5;
System.out.println(2423);
System.out.print("Good morning " + name +"!");
System.out.println();
System.out.println("Value integer: " + x);
System.out.println("Final note: " + r);
```

```
2423
Good morning Adriana!
Value Integer: 75
Final grade: 19.5
```

read data

- We can use the Scanner class to read data from from the keyboard.

```
import java.util.Scanner;
...
Scanner sc = new Scanner(System.in);
```

- Useful methods of the Scanner class:
  - nextLine() – read an entire line (String)
  - next() – read a word (String)
  - nextInt() – read an integer (int)
  - nextDouble() – read a real number (double)

Example

```
import java.util.Scanner;
public class Tests {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.print("What is your name? ");
		String name = sc.nextLine();
		System.out.print("How old are you? ");
		int age = sc.nextInt();
		System.out.print("How much do you weigh? ");
		double weight = sc.nextDouble();
		System.out.println("Name: " + Name);
		System.out.println("Age: " + age + " years old");
		System.out.println("Weight: " + Weight + "Kgs.");
		sc.close();
	}
}
```

```
What is your name? Ana Lima
How old are you? 28
What's your weight? 55
Name: Ana Lima
Age: 28 years
Weight: 55.0Kgs.
```

operator precedence

- The order of execution of operators follows rules of precedence. int a = 5;

  int b = -15;
  double c = ++ab/30;

- To change the order and/or clarify complex expressions it is suggested to use parentheses.

  c = (++a)-(b/30);

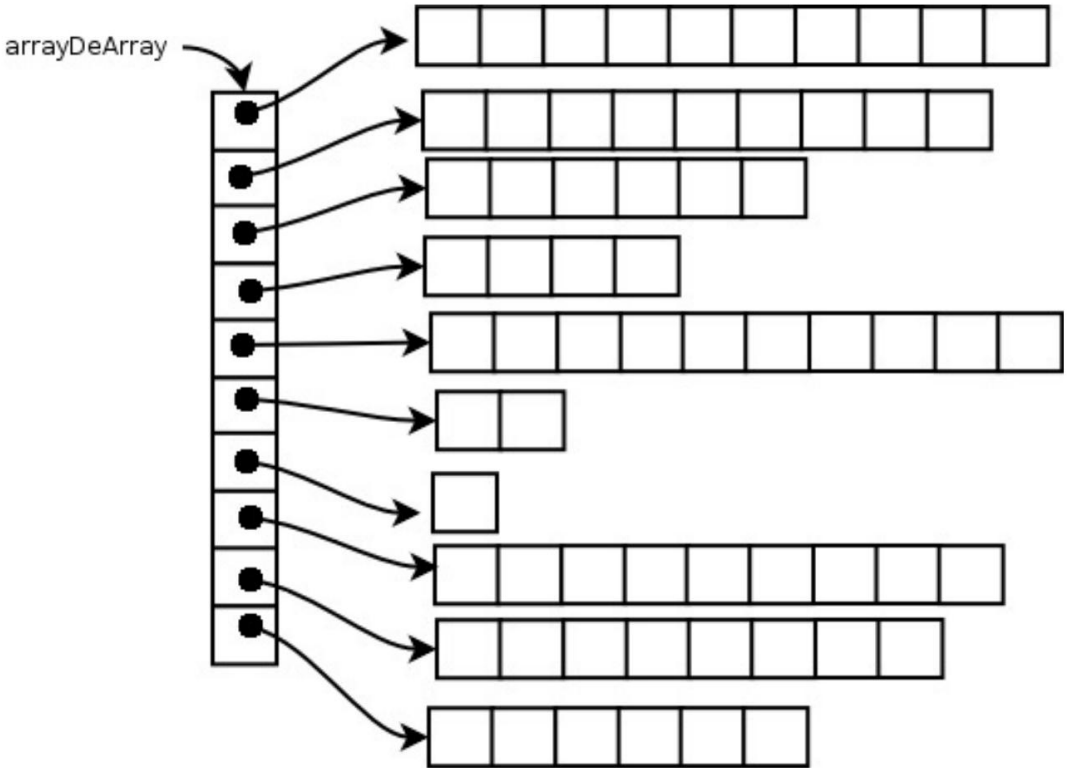
Operator Precedence

Operators	Precedence
postfix	*expr++ expr--*
unary	*++expr --expr +expr -expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Referenced types

- Variables of these types do not contain the values but the addresses for accessing the effective values



- Include:
  - Vectors (arrays)
  - Objects

vectors

- We can declare **vectors** (arrays) of variables of the same type

```
int[] vet1; int
vet2[]; // alternative syntax and equivalent to the previous one
```

- In addition to the declaration, we still need to define its **dimension.**
  - initialization with default values:

```
int[] v1 = new int[3]; // vector with 3 elements: 0, 0 ,0
```

  - declaration and initialization with specific values

```
int[] v2 = { 1, 2, 3 }; // vector with 3 elements: 1, 2, 3
// or
int[] v3 = new int[] { 1, 2, 3};
```

Vectors in Java

- Arrays in Java have **a fixed dimension,** not can increase in size at runtime

- The **new** instruction creates an array with the indicated dimension and initializes all positions
  - For **primitive** types with the default value
  - For **references,** with the value null

Access to vector elements

- Elements are accessed through indexes.
  - The index of the first element is 0 (zero).

```
int[] table = new int[3]; // indexes between 0 and 2
table[0] = 10;
table[1] = 20;
table[2] = 30;
table[3] = 11; // error!!
```

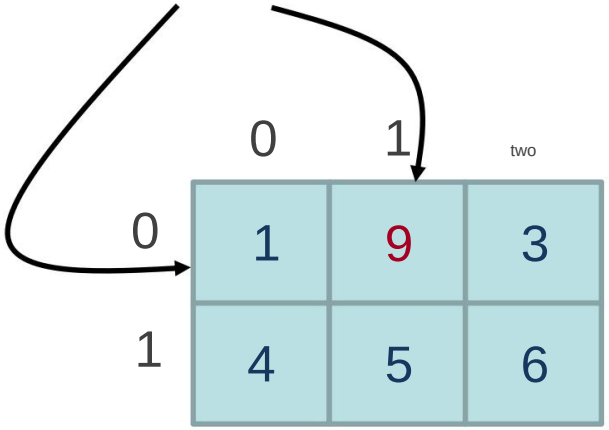
- The size of a vector ***v*** is given by ***v.length.***

```
System.out.println(table.length); // 3
```

multidimensional vectors

- It is possible to create multidimensional vectors, ie vectors of vectors:

```
int[][] a = { { 1, 2, 3, }, { 4, 5, 6, } } ;
System.out.println(a.length); // two
System.out.println(a[0].length); // 3
a[0][1] = 9;
```



multidimensional vectors

- Are vectors of vectors (arrays of arrays)
  - Are implemented using nesting/cascade

```
int table[][]= new int[30][20];
```

  - Defines table as being of type int[][]
  - Dynamically reserves a vector of 30 elements, each one of them of type int[20]
  - Reserves 30 vectors of 20 integers and stores the reference (address) for each of these in the 30-position vector

summary

v Programming paradigms v

Structure of a program in java

– Main class, main function

v Data

– Primitive types,

variables v Operators and

precedences v Expressions with

operators v Vectors