

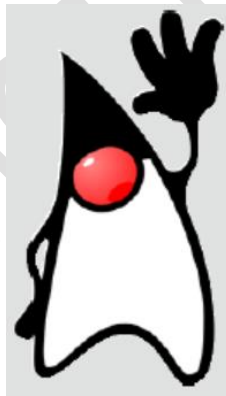


---

# JAVA FOUNDATIONS 1Z0-811

---

ORACLE ACADEMY



2 DE SEPTIEMBRE DE 2025

<https://academy.oracle.com/>

[HTTPS://GITHUB.COM/ISC-UPA/2025-3-TIID3C-POO](https://github.com/ISC-UPA/2025-3-TIID3C-POO)

## Contenido

1. Introduction .....	3
1.1. Technological Requirements: .....	3
1.2. Create Java Project: .....	4
1.3. Setting Up Java .....	5
2. Java Basics .....	7
2.1. The Software Development Process.....	7
2.2. What is my Program Doing? .....	8
2.3. Introduction to Object-Oriented Programming Concepts .....	8
3. Java Data Types.....	9
3.1. What is a Variable? .....	9
String x ="Sam"; .....	9
3.2. Numeric Data.....	9
Rules of Precedence.....	10
3.3. Textual Data.....	11
Primitives .....	11
Escape Sequence.....	12
3.4. Converting Between Data Types .....	12
3.5. Keyboard Input .....	14
Quiz 1: JFo - Section 3 - L1-L2 .....	14
Quiz 2: JFo - Section 3 - L3-L5 .....	14
4. Java Methods and Library Classes.....	15
4.1. What Is a Method? .....	15
4.2. The import Declaration and Packages.....	16
Quiz 1: JFo - Section 4 - L1-L2 .....	16
Quiz 2: JFo - Section 4 - L3-L5 .....	16
<b>4.3. The String Class</b> .....	17
4.4. The Random class .....	19
4.5. The Math Class .....	20
5. Decision Statements .....	21
5.1. Boolean Expressions and if/else Constructs.....	21
5.2. Understanding Conditional Execution.....	22
5.3. switch Statement.....	23
6. Loop Constructs .....	25
6.1. for Loops.....	25

6.2.	while and do-while loops.....	26
6.3.	Using break and continue Statements .....	27
7.	Creating Classes .....	28
7.1.	Creating a Class.....	28
7.2.	Instantiating Objects.....	28
7.3.	Constructors .....	28
7.4.	Overloading Methods .....	28
7.5.	Object Interaction and Encapsulation .....	28
7.6.	static Variables and Methods .....	28
8.	Arrays and Exceptions.....	28
8.1.	One-dimensional Arrays .....	28
8.2.	ArrayLists .....	28
8.3.	Exception Handling.....	28
8.4.	Debugging Concepts and Techniques.....	28
9.	JavaFX.....	28
9.1.	Introduction to Java FX.....	28
9.2.	Colors and Shapes.....	28
9.3.	Graphics, Audio and MouseEvents.....	28

→

# 1. Introduction

## 1.1. Technological Requirements:

Java JDK <https://www.oracle.com/java/technologies/downloads/>

VS Code <https://code.visualstudio.com/Download>

Extensions: **Extension Pack for Java**

jdk-8u202-windows-x64.exe

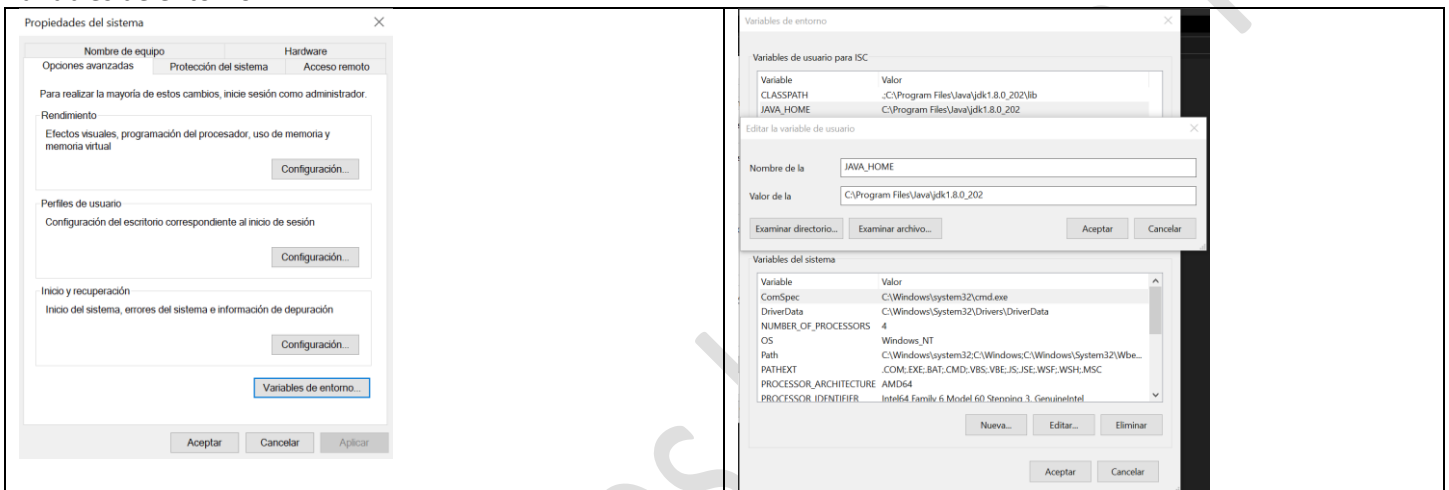
VSCodeSetup-x64-1.103.2.exe

Integrated Development Environment (IDE)

Eclipse IDE: <https://www.eclipse.org/downloads/packages/>

NetBeans IDE <https://netbeans.apache.org/download/index.html>

### Variables de entorno



Panel de control -> Sistema -> Configuración avanzada del sistema

Opciones avanzadas -> Variables de entorno -> Variables de Usuario

JAVA_HOME C:\Program Files\Java\jdk1.8.0_202	PATH %JAVA_HOME%\BIN
CLASSPATH .; %JAVA_HOME%\LIB	Probar Instalación desde CMD C:\>java -version (correr) C:\>javac -version (compilar)

C:\dev>java -version java version "1.8.0_202"  C:\dev>javac -version javac 1.8.0_202  C:\dev\poo>javac Hola.java  C:\dev\poo>java Hola Hello World!	public class Hola {  public static void main(String[] args) { System.out.println("Hello World!"); } }
--	--

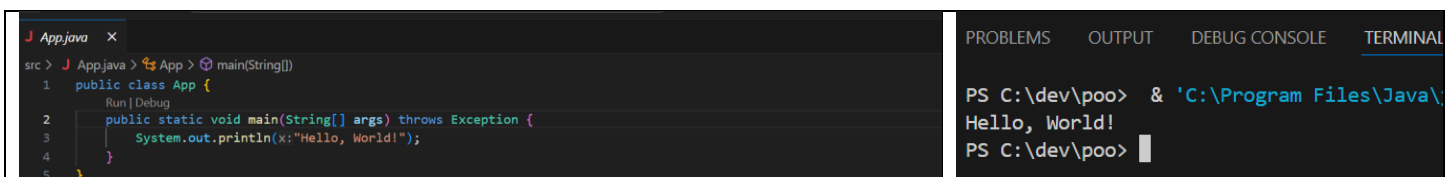
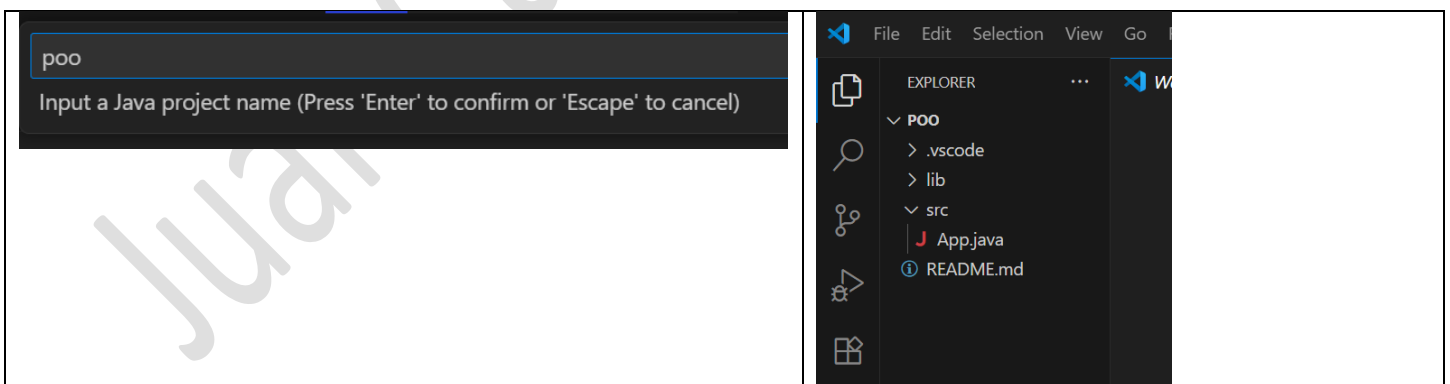
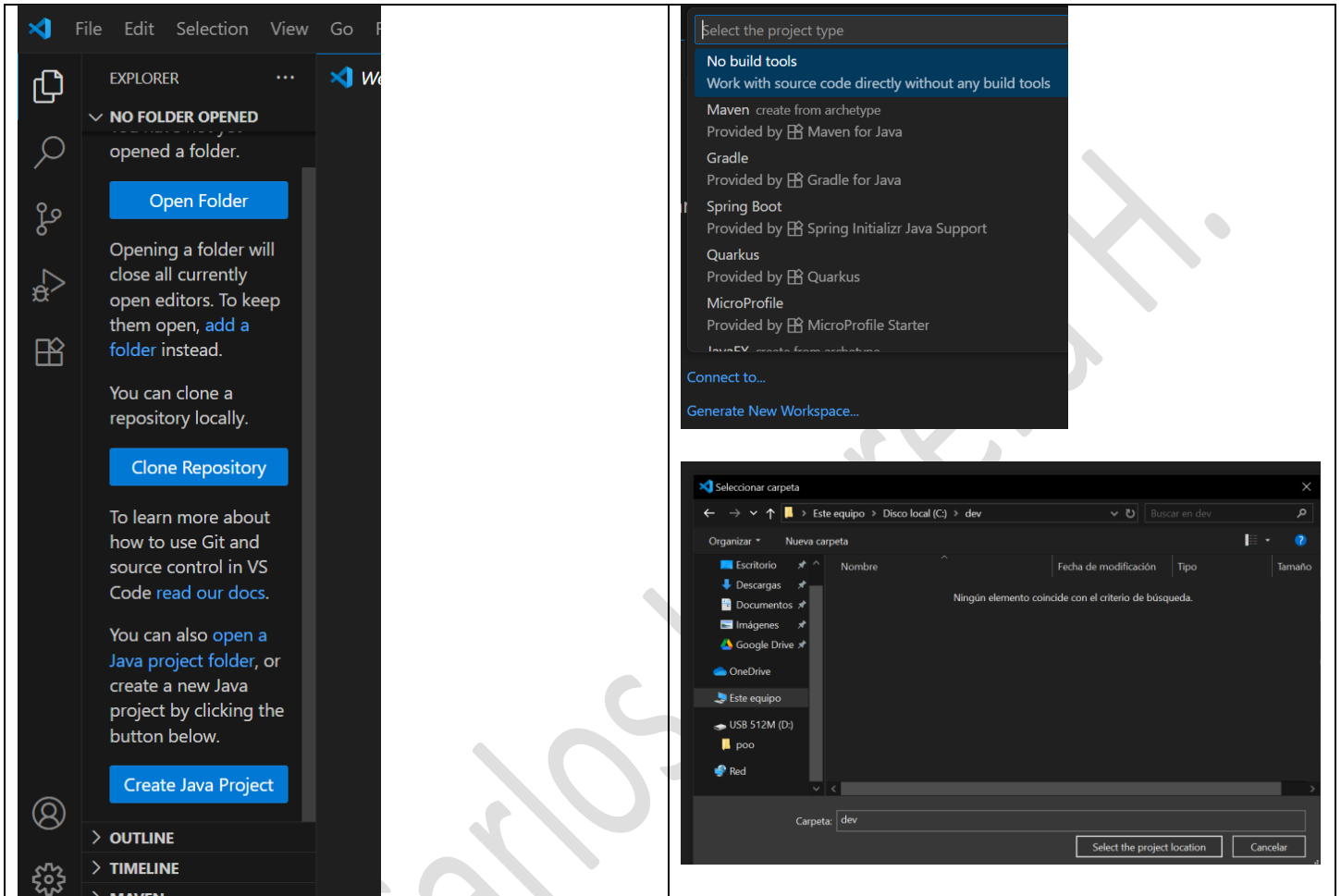
→

Crear un Directorio de trabajo

C:\> mkdir dev

C:\dev>

## 1.2. Create Java Project:



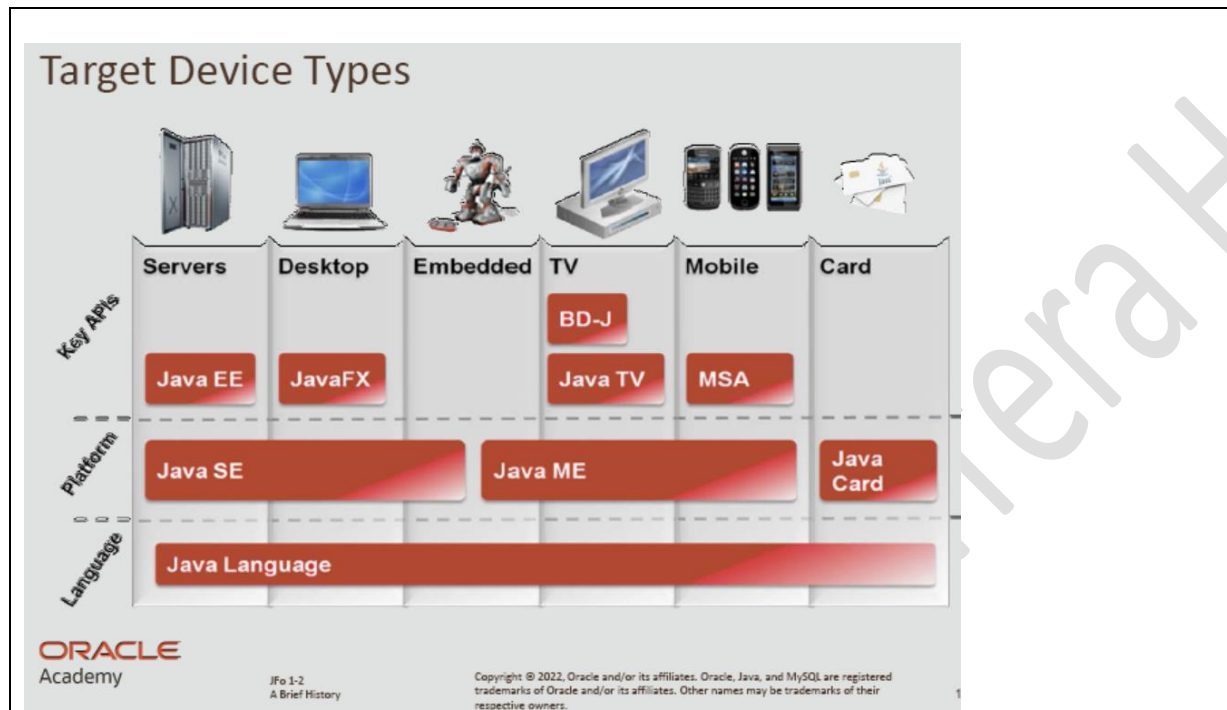
→

### 1.3. Setting Up Java

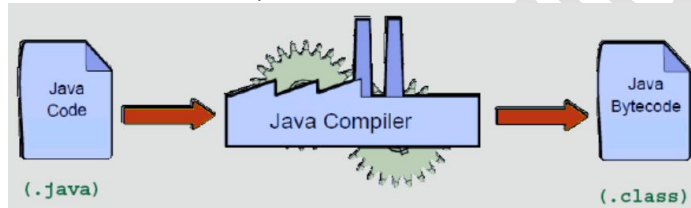
James Gosling is considered the “Father of Java”. Duke, the Java Mascot.

Oracle acquired Sun Microsystems in 2010, and released JDK 7 in 2011, and JDK 8 in 2014.

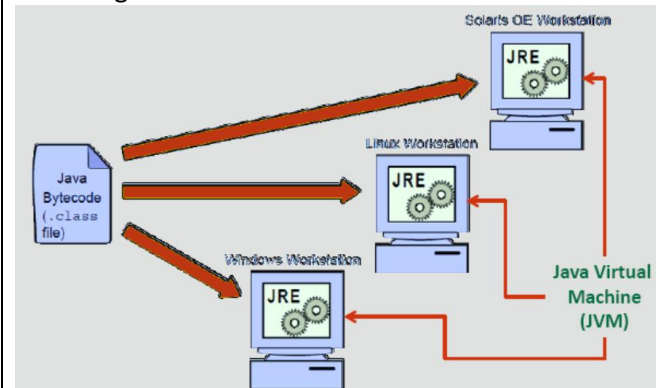
Jakarta EE Is used to create large enterprise, server-side, and client-side distributed applications



#### Java is Platform-Independent



#### Java Programs Run in a JVM



#### Java Runtime Environment (JRE)

Includes:

- The Java Virtual Machine (JVM)
- Java class libraries

Purpose:

- Read bytecode (.class)
- Run the same bytecode anywhere with a JVM

#### Java Development Kit (JDK)

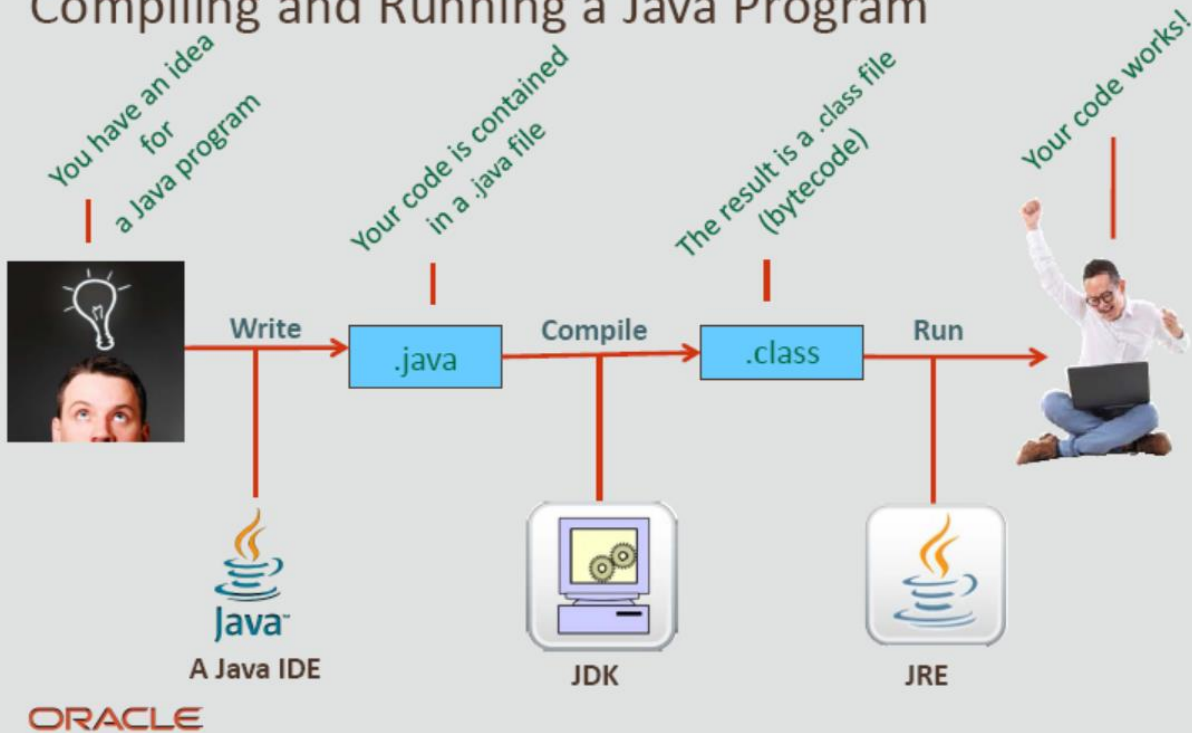
Includes:

- JRE Java Compiler
- Additional tools

Purpose:

Compile bytecode (.java → .class)

# Compiling and Running a Java Program



A Java IDE is used to **write** source code ( .java )



The JDK **compiles** bytecode ( .java → .class )



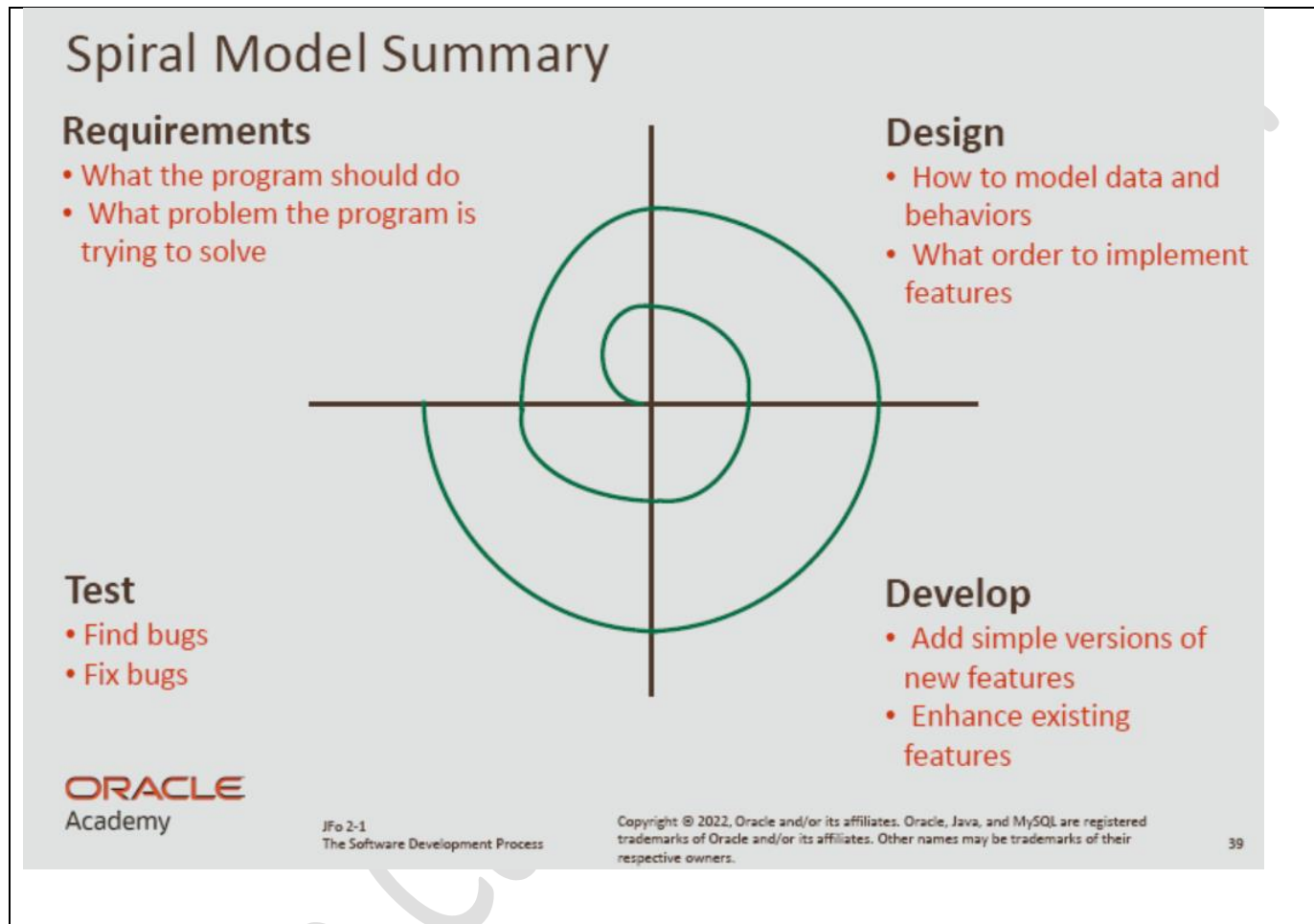
Bytecode **runs** in a JVM, which is part of the JRE

→

## 2. Java Basics

### 2.1. The Software Development Process

#### Spiral Model of Development



<https://objectstorage.uk-london-1.oraclecloud.com/n/lrvrlgaqj8dd/b/Games/o/JavaPuzzleBall/index.html>

→

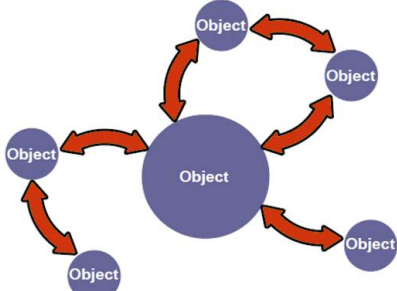


## 2.2. What is my Program Doing?

<p>Code within curly braces is called a block of code</p> <p>Indentation before a line of code (4 spaces)</p> <p>Whitespace</p> <p>End statements with semicolons (;)</p> <p>// Single-line comments</p> <p>Multi-line comments</p> <pre>/* Bienvenidos    a poo */</pre>	<pre>public static void NombreMetodo() { . . } NombreMetodo(); // llamar al método</pre> <p>Debug</p> <p>To set a breakpoint</p> <p>Press Step Over</p>
---	---

## 2.3. Introduction to Object-Oriented Programming Concepts

<p>Procedural languages ...</p> <ul style="list-style-type: none"> <li>Read one line at a time</li> <li>The C language is procedural</li> </ul>	<p>Object-oriented languages...</p> <ul style="list-style-type: none"> <li>Read one line at a time</li> <li>Model objects through code</li> <li>Emphasize object interaction</li> <li>Allow interaction without a prescribed order</li> <li>Java and C++ are object-oriented languages</li> </ul>
---	---

<p>Object-Oriented Programming</p> <ul style="list-style-type: none"> <li>Interaction of objects</li> <li>No prescribed sequence</li> </ul>	
---	--

## Modeling Properties and Behaviors

<p><b>Customer class</b></p> <pre>name address billing info age customer number order number  requestDiscount() setAddress() shop() displayCustomer()</pre> <p><b>Class name</b></p> <p><b>Fields</b></p> <p><b>Methods</b></p>	<p><b>Class declaration</b></p> <pre>1 public class Customer { 2     public String name = "Junior Duke"; 3     public int custID = 1205; 4     public String address; 5     public int orderNum; 6     public int age; 7 8     public void displayCustomer() { 9         System.out.println("Customer: "+name); 10    } //end method displayCustomer 11 } //end class Customer</pre> <p><b>Fields (Properties) (Attributes)</b></p> <p><b>Methods (Behaviors)</b></p>
---	---

## Quiz: JFo - Section 2 Questions 15



### 3. Java Data Types

#### 3.1. What is a Variable?

```
String x = "Sam";  
System.out.println("My name is " + x);
```

Variables03.java (There are 6 mistakes)

Type	Keyword	Example Values
Boolean	<code>boolean</code>	<code>true</code> , <code>false</code>
Integer	<code>int</code>	1, -10, 20000, 123_456_789
Double	<code>double</code>	1.0, -10.0005, 3.141
String	<code>String</code>	"Alex", "I ate too much dinner."

#### Variable Naming Conventions

- Begin each variable with a lowercase letter
- Subsequent words should be capitalized: `myVariable`
- Choose names that are mnemonic and that indicate the intent of the variable to the casual observer
- Remember that ...
- Names are case-sensitive
- Names can't include white space

```
Int studentAge = 20;  
String myCatchPhrase = "Enjoy Alex Appreciation Day!";
```

#### 3.2. Numeric Data

##### Integral Primitive Types

Type	Length	Number of Possible Values	Minimum Value	Maximum Value
<code>Byte</code>	8 bits	$2^8$ , or... 256	$-2^7$ , or... -128	$2^7-1$ , or... 127
<code>short</code>	16 bits	$2^{16}$ , or... 65,535	$-2^{15}$ , or... -32,768	$2^{15}-1$ , or... 32,767
<code>int</code>	32 bits	$2^{32}$ , or... 4,294,967,296	$-2^{31}$ , or... -2,147,483,648	$2^{31}-1$ , or... 2,147,483,647
<code>long</code>	64 bits	$2^{64}$ , or... 18,446,744,073,709,551 ,616	$-2^{63}$ , or... -9,223,372,036, 854,775,808L	$2^{63}-1$ , or... 9,223,372,036, 854,775,807L

`++`   `--`   `*=`   `/=`   `%=`   `++`   `--`   Pre/Post   `a+=b`   `a = a + ( b )`

```
// pre y post incremento y decremento
```

```
int players = 0;
System.out.println("players online: " + players++);
System.out.println("The value of players is " + players);
System.out.println("The value of players is now " + ++players);
System.out.println("The value of players is " + players);
```

#### Floating Point Primitive Types

Type	Float Length	When will I use this?
<b>float</b>	32 bits	Never
<b>double</b>	64 bits	Often

```
double x = 9/2;          double x = 9/2.0;
```

```
final double PI = 3.141592;
```

Final variable naming conventions:

- Capitalize every letter
- Separate words with an underscore  
MINIMUM\_AGE

#### Rules of Precedence

- Operators within a pair of parentheses
- Increment and decrement operators (++or --)
- Multiplication and division operators, evaluated from left to right
- Addition and subtraction operators, evaluated from left to right
- If operators of the same precedence appear successively, the operators are evaluated from left to right

```
int x = (((25 - 5) * 4) / (2 - 10)) + 4;
```

```
int y = 25 - 5 * 4 / 2 - 10 + 4;
```

→

### 3.3. Textual Data

Use the char data type

Use Strings

Concatenate Strings

Understand escape sequences

Understand print statements better

Char is used for a single character (16 bits) char shirtSize= 'M';	A String can handle multiple characters String greeting = "Hello World!"; // Hard-coding
---	---

#### Primitives

Type	Length	Data
boolean	1 bit	true / false
byte	8 bits	Integers
short	16 bits	Integers
int	32 bits	Integers
long	64 bits	Integers
float	32 bits	Floating point numbers
double	64 bits	Floating point numbers
char	16 bits	Single characters
<b>Where are Strings?</b>		
String is capitalized <ul style="list-style-type: none"><li>• Strings are an object, not a primitive</li><li>• Object types are capitalized by convention</li></ul>		

Combining multiple Strings is called concatenation

```
String totalPrice = "Total: $" + 3 + 2 + 1;
```

```
String totalPrice = 3 + 2 + 1 + "Total: $";
```

```
String totalPrice = "Total: $" + (3 + 2 + 1);
```

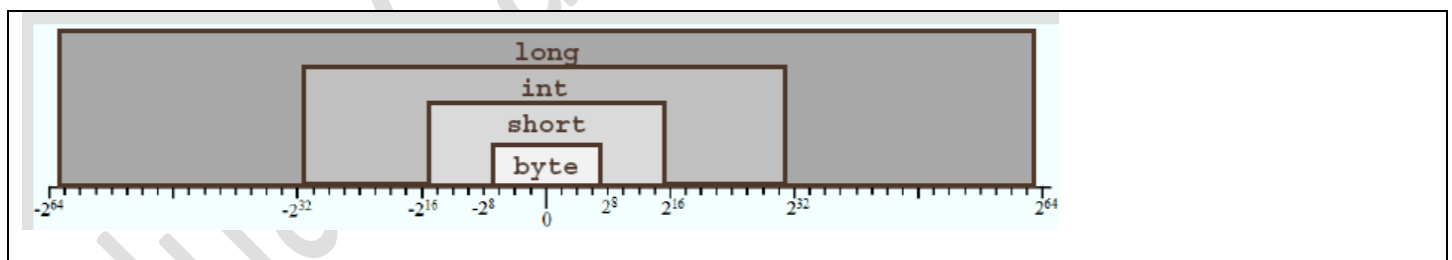
## Escape Sequence

Escape Sequence	Description	<pre>System.out.println("The cat said \"Meow!\" to me."); println() vs. print()  System.out.println("1\t2\t3\t\"Hola\" mundo"); 1    2    3    "Hola" mundo  System.out.println("Hola\nAdios"); Hola Adios</pre>
<code>\t</code>	Insert a new tab	
<code>\b</code>	Insert a backspace	
<code>\n</code>	Insert a new line	
<code>\r</code>	Insert a carriage return	
<code>\f</code>	Insert a formfeed	
<code>\'</code>	Insert a single quote character	
<code>\"</code>	Insert a double quote character	
<code>\\</code>	Insert a backslash character	

```
System.out.println("This is the first line."
    + "This is NOT the second line.");
sout tab  "Metodo abreviado"
```

## 3.4. Converting Between Data Types

<pre>double x = 9 / 2; // Should be 4.5 System.out.println(x); // prints 4.0  double y = 4; System.out.println(y); //prints 4.0</pre>	<pre>int    num1 = 7; double num2 = 2; double num3; num3 = num1 / num2; // num3 is 3.5</pre>
---	--



<ul style="list-style-type: none"> <li>• Automatic promotions: <ul style="list-style-type: none"> <li>- If you assign a smaller type to a larger type: <pre> byte → short → int → long </pre> </li> <li>- If you assign an integral value to a floating-point type: <pre> 4 → 4.0 </pre> </li> </ul> </li> <li>• Examples of automatic promotions: <ul style="list-style-type: none"> <li>- <code>long</code> intToLong = 6;</li> <li>- <code>double</code> intToDouble = 4;</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• When to cast: <ul style="list-style-type: none"> <li>- If you assign a larger type to a smaller type: <pre> byte ← short ← int ← long </pre> </li> <li>- If you assign a floating point type to an integral type: <pre> 3 ← 3.0 </pre> </li> </ul> </li> <li>• Examples of casting: <ul style="list-style-type: none"> <li>- <code>int</code> longToInt = (<code>int</code>) 20L;</li> <li>- <code>short</code> doubleToShort = (<code>short</code>) 3.0;</li> </ul> </li> </ul> <p>double pi = 3.1416 int entero = (int) pi</p>
---	---

127 in binary is 01111111; 128 in binary is 10000000.

Java uses the first bit in a number to indicates sign (+/-)

byte, short, and char values are automatically promoted to int prior to an operation

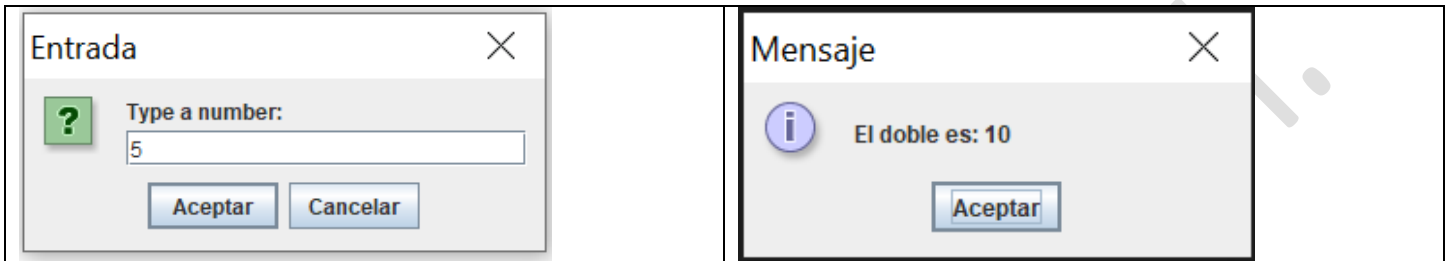
<ul style="list-style-type: none"> <li>• Solution using larger data type: <pre> int num1 = 53; int num2 = 47; int num3; num3 = (num1 + num2); </pre> <p>Changed from byte to int</p> </li> <li>• Solution using casting: <pre> int num1 = 53; // 32 bits of memory to hold the value int num2 = 47; // 32 bits of memory to hold the value byte num3; // 8 bits of memory reserved num3 = (byte)(num1 + num2); // no data loss </pre> </li> </ul>	<h3>Automatic Promotion</h3> <ul style="list-style-type: none"> <li>• Example of a potential problem: <pre> short a, b, c; a = 1; b = 2; c = a + b; //compiler error </pre> <p>a and b are automatically promoted to integers</p> </li> <li>• Example of potential solutions: <ul style="list-style-type: none"> <li>- Declare c as an <code>int</code> type in the original declaration: <pre> int c; </pre> </li> <li>- Type cast the (a+b) result in the assignment line: <pre> c = (short) (a+b); </pre> </li> </ul> </li> </ul> <p>int x = 123_456_789; int x = 123456789;</p> <p>intintVar1 = Integer.parseInt("100"); doubledoubleVar2 = Double.parseDouble("2.72");</p>
---	---

→

### 3.5. Keyboard Input

```
System.out.println("\033[H\033[2J"); // limpiar pantalla

String input = JOptionPane.showInputDialog(null, "Type a number:");
int number = Integer.parseInt(input);
number *= 2;
JOptionPane.showMessageDialog(null, "El doble es: " + number);
```



The Scanner searches for tokens

A few useful Scanner methods ...

- `nextInt()` reads the next token as an int
- `nextDouble()` reads the next token as a double
- `next()` reads the next token as a String

```
Scanner sc = new Scanner(System.in);
```

The Scanner class considers space as the default delimiter while reading the input

Reading from a File

- `nextLine()` advances this Scanner past the current line and returns the input that was skipped
- `findInLine("StringToFind")` Attempts to find the next occurrence of a pattern constructed from the specified String, ignoring delimiters

```
Scanner sc = new Scanner(MyClase.class.getResourceAsStream("texto.txt"));
```

```
Scanner sc = new Scanner(System.in);
int x = sc.nextInt();
double y = sc.nextDouble();
String z = sc.next();
String linea = sc.nextLine();
int numero = Integer.parseInt(z);
sc.close();
```

Quiz 1: JFo - Section 3 - L1-L2

Quiz 2: JFo - Section 3 - L3-L5



## 4. Java Methods and Library Classes

### 4.1. What Is a Method?

Instantiate an object

These classes outline objects' ...

Properties(fields)

Behaviors(methods)

#### Variables for Objects

```
int      age = 22;
String   str  = "Happy Birthday!";
Scanner  sc   = new Scanner();
Calculator calc = new Calculator();
```

Annotations:   
 - Under `int`: type   
 - Under `age`: name   
 - Under `22`: value   
 - Under `String`: type   
 - Under `str`: name   
 - Under `"Happy Birthday!"`: value   
 - Under `Scanner`: type   
 - Under `sc`: name   
 - Under `new Scanner()`: value   
 - Under `Calculator`: type   
 - Under `calc`: name   
 - Under `new Calculator()`: value

Method name

Method return type

Parameters

```
public double calculate(int x, double y){
    double quotient = x/y;
    return quotient;
} //end method calculate
```

Implementation

```
double tax = 0.05;
double tip = 0.15;

double person1 = 10;
double total1 = person1*(1 + tax + tip);
System.out.println(total1);

double person2 = 12;
double total2 = person2*(1 + tax + tip);
System.out.println(total2);
```

```
public void findTotal(double price, String name){
    double total = price * (1 + tax + tip);
    System.out.println(name + ": $ " + total);
} //end method findTotal
```

#### Method Arguments and Parameters

- An argument is a value that's passed during a method call:

```
Calculator calc = new Calculator();
calc.calculate(3, 2.0); //should print 1.5
```

Arguments

- A parameter is a variable that's defined in the method declaration:

```
public void calculate(int x, double y){
    System.out.println(x/y);
} //end method calculate
```

Parameters

#### Method Return Types

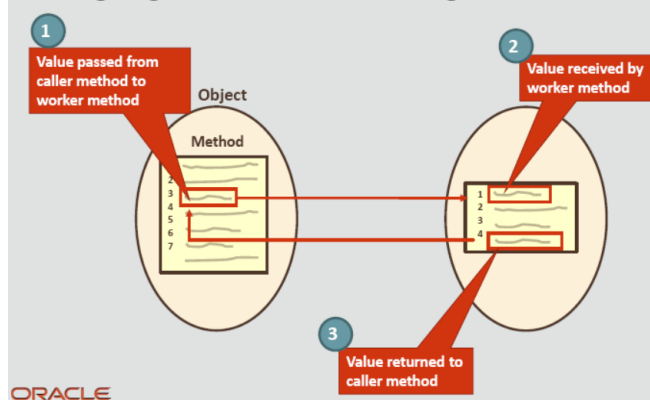
- Variables can have values of many different types:

int double long char float byte  
short String boolean Calculator

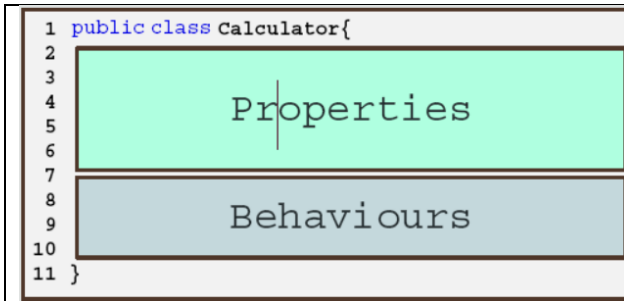
- Method calls also return values of many different types:

int double long char float byte  
short String boolean Calculator

#### Passing Arguments and Returning Values







```

public class Calculator{
    //Fields
    public double tax = 0.05;
    public double tip = 0.15;
    public double originalPrice = 10;

    //Methods
    public void findTotal(){
        //Calculate total after tax and tip
        //Print this value
    } //end method findTotal
} //end class Calculator

```

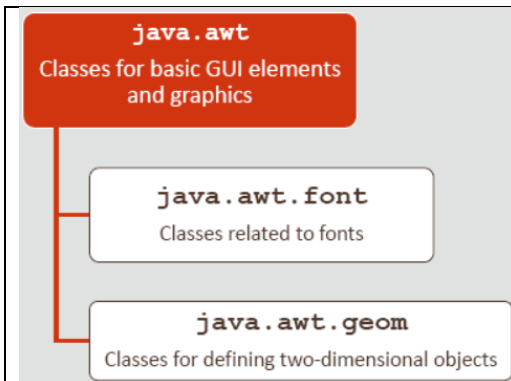
Calculator calc = **new** Calculator();

#### 4.2. The import Declaration and Packages

[java.base \(Java SE 17 & JDK 17\)](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/module-summary.html) <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/module-summary.html>

[Overview \(Java SE 15 & JDK 15\)](https://docs.oracle.com/en/java/javase/15/docs/api/index.html) <https://docs.oracle.com/en/java/javase/15/docs/api/index.html>

Package	Purpose
<b>java.lang</b>	Provides classes that are fundamental to the design of the Java language By default, the java.lang package is automatically imported into all Java programs
<b>javax.swing</b>	Provides classes to build GUI components
<b>java.net</b>	Provides classes for networking applications
<b>java.time</b>	Provides classes for dates, times, instants, and durations



import

java.util.Scanner

Package

Class Name

Import javax.swing.\*; // importar todas las clases

import keyword followed by the name of the package dot, the name of the class

import javax.swing.JOptionPane;

Package Name

Class Name

```

java.util.Scanner keyboard = new java.util.Scanner(System.in);
Scanner sc = new Scanner(System.in);
JOptionPane.showMessageDialog(null, "Hello!");

```

Quiz 1: JFo - Section 4 - L1-L2

Quiz 2: JFo - Section 4 - L3-L5



## 4.3. The String Class

java.lang.String

In Java, strings are not a primitive data type. Instead, they are objects of the String class.

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/module-summary.html>

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html>

### String Class Documentation: Method Summary

- `public int charAt(int index)`

Return type of the method

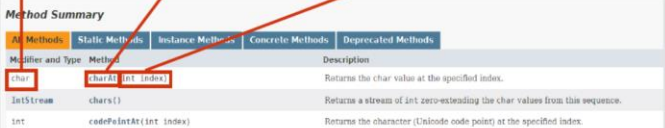
`int`

Name of the method

`charAt`

Data type of the parameter that must be passed into the method

`int`



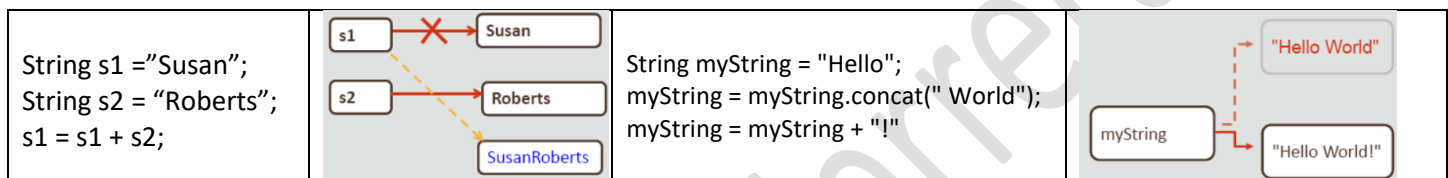
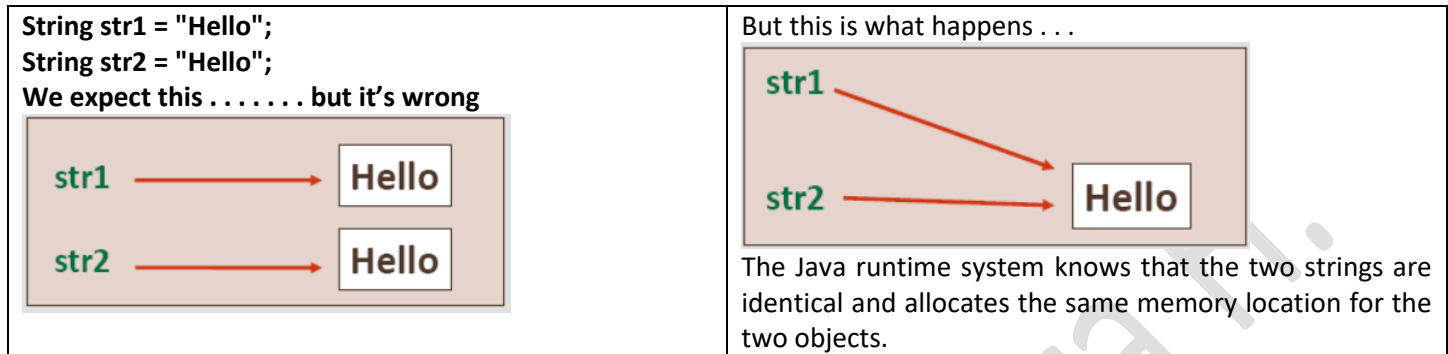
```
String str = "Hello, World";
```

H	e	l	l	o	,		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10	11

String str = new String("Hello, World");  
Not commonly used and not recommended

<code>public int length()</code>	Returns the length of this string Example: <code>LastName.length()</code>
<code>public char charAt(int index)</code>	Returns the char value at the specified index
<code>String concat(String str)</code>	Concatenates the specified string to the end of this string. <code>String producto = "coca";</code> <code>producto.concat("cola");</code> <code>producto= producto.concat("cola");</code> <code>producto = producto + "cola";</code>
Boolean <code>contains(CharSequence s)</code>	Returns true if and only if this string contains the specified sequence of char values.
<code>public int indexOf(String str)</code>	Returns the index within this string of the first occurrence of the specified substring
<code>str.indexOf(char c)</code>	Returns the index value of the first occurrence of c in String str
<code>s1.indexOf(char c, int beginIdx)</code>	Returns the index value of the first occurrence of c in String s1, starting from beginIdx to the end of the string
<code>str.substring(int beginIdx)</code>	Returns the substring from beginIdx to the end of the string
<code>str.substring(int beginIdx, int endIdx)</code>	Returns the substring from beginIdx up to, but not including, endIdx
<code>String replace(char oldChar, char newChar)</code> <code>str.replace(CharSequence target, CharSequence replacement)</code>	This method replaces <b>all</b> occurrences of matching characters in a string
<code>replaceFirst(String pattern, String replacement)</code>	replaces only the first occurrence of a matching character pattern in a string
<code>Int lastIndexOf(String str)</code> <code>Int lastIndexOf(String str, int fromIndex)</code> <code>String trim()</code> <code>String toLowerCase()</code>	

Strings Are Immutable, its value can't be changed.



## Comparing String

ASCII    '0' = 48            '1' = 49            'A' = 65            'a' = 97

The strings are compared character by character until their order is determined or until they prove to be identical

Syntax:            **s1.compareTo(s2)**            Example: `int a = "computer".compareTo("comparison");`

Returns an integer value that indicates the ordering of the two strings

- Returns == 0    when the two strings are lexicographically equivalent
- Returns < 0    when then the string calling the method is lexicographically first
- Returns > 0    when the parameter passed to the method is lexicographically first

→

#### 4.4. The Random class

```
import java.util.Random;
```

- `Random rand = new Random();`  
    **`rand.setSeed(5L);`      Colocar una semilla**
- `Math.random();` // entre 0 y 1

```
rand.nextInt(max - min + 1) + min;
```

```
(int) (Math.random() * (max - min + 1) ) + min;
```

Method	Produces
<code>boolean nextBoolean();</code>	A true or false value
<code>int nextInt();</code>	An integral value between <code>Integer.MIN_VALUE</code> and <code>Integer.MAX_VALUE</code>
<code>long nextLong();</code>	A long integral value between <code>Long.MIN_VALUE</code> and <code>Long.MAX_VALUE</code>
<code>float nextFloat();</code>	A decimal number between 0.0 (included) and 1.0 (excluded)
<code>double nextDouble();</code>	A decimal number between 0.0 (included) and 1.0 (excluded)

→

#### 4.5. The Math Class

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/module-summary.html>

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/math/package-summary.html>

The methods of the Math class are **static methods**

Some of the Methods Available in Math Class

Method Name	Description
abs(value)	absolute value
ceil(value)	rounds up
cos(value)	cosine, in radians
floor(value)	rounds down
log(value)	logarithm base e
log10(value)	logarithm base 10
max(value1, value2)	larger of two values
min(value1, value2)	smaller of two values
pow(base, exponent)	base to the exponent power
random()	random double between 0 and 1
round(value)	nearest whole number
sin(value)	sine, in radians
sqrt(value)	square root

**double a = Math.sqrt(121.0);**      **Math.E**      **Math.PI**

**360° = 2π rad**      **1° = π/180 rad**      **1 rad = 180/π °**

**BMI = Peso en libras / Altura en pulgadas<sup>2</sup> \* 703**

**IMC = Peso (kg) ÷ (Altura (m))<sup>2</sup>**

→

## 5. Decision Statements

### 5.1. Boolean Expressions and if/else Constructs

In Java the values for the boolean data type are true and false, instead of yes and no.

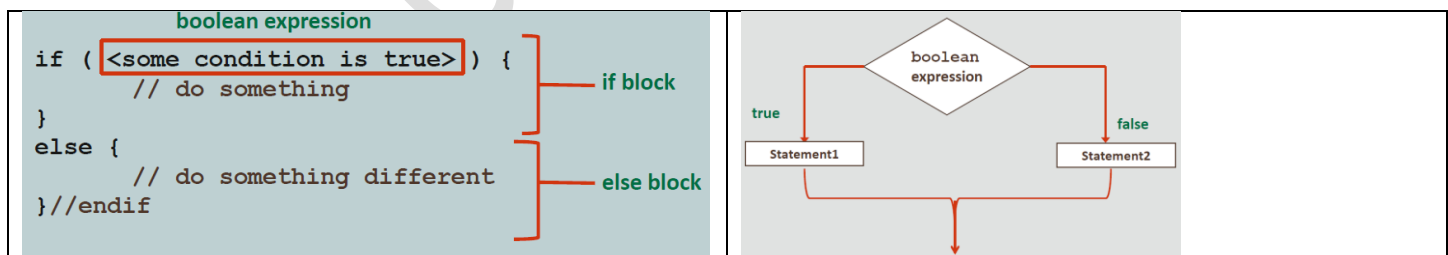
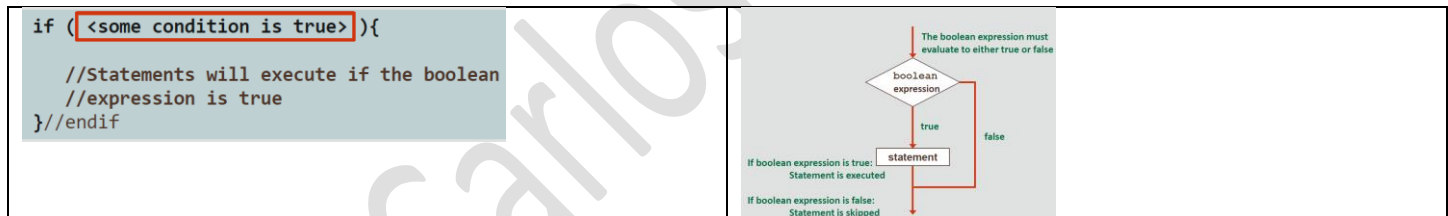
```
boolean bandera = true;
int x = 4;
boolean isFive = x == 5;
```

#### Relational Operators

Condition	Operator	Example
Is equal to	==	int i=1; (i == 1)
Is not equal to	!=	int i=2; (i != 1)
Is less than	<	int i=0; (i < 1)
Is less than or equal to	<=	int i=1; (i <= 1)
Is greater than	>	int i=2; (i > 1)
Is greater than or equal to	>=	int i=1; (i >= 1)

Conditional statements in Java are:

- if statement
- if/else statement
- switch statement



== compares the values of primitives  
== compares the objects' locations in memory

<pre>String x = "Ora"; String y = "cle"; String z = x + y; boolean test = (z == x + y); System.out.println(test); //false    Why?</pre>	<pre>String x = "Ora"; String y = "cle"; String z = x + y; boolean test = z.equals(x + y); System.out.println(test); //true     Why?</pre>
---	--

## 5.2. Understanding Conditional Execution

### Handling Multiple Conditions

<pre>int grade = 90; int numberDaysAbsent = 0;  if (grade &gt;= 88) {     if (numberDaysAbsent == 0) {         System.out.println("qualify");     } // endif } // endif</pre>	<pre>int grade = 90; int numberDaysAbsent = 0;  if ((grade &gt;= 88) &amp;&amp; (numberDaysAbsent == 0)) {     System.out.println("qualify"); } // endif</pre>
---	--

Logic Operator	Meaning
<b>&amp;&amp;</b>	AND
<b>  </b>	OR
<b>!</b>	NOT

<pre>boolean bandera = true; if (bandera) {     System.out.println("qualify"); } else {     System.out.println("fail"); }</pre>	<pre>boolean bandera = true; if (!bandera) {     System.out.println("fail"); } else {     System.out.println("qualify"); }</pre>
---	--

The && and || operators are short-circuit operators

Skipping the Second AND Test     $x=0$              $b = (x \neq 0) \ \&\& \ ((y / x) > 2);$

Skipping the Second OR Test     $x=0$              $b = (x \leq 10) \ || \ (x > 20);$

### Ternary Conditional Operator

Operation	Operator	Example
If condition is true: assign result = value1 Otherwise: assign result = value2	?:	<pre>result = condition ? value1 : value2</pre> <p>Example:</p> <pre>int x = 2, y = 5, z = 0; z = (y &lt; x) ? x : y;</pre>

<pre>int numberOfGoals = 1; System.out.println("I scored " + numberOfGoals + " "     + (numberOfGoals == 1 ? "goal" : "goals"));</pre>	
--	--

<pre> if (&lt;condition1&gt;){     //code_block1 } else if (&lt;condition2&gt;){     // code_block2 } else if (&lt;condition3&gt;){     // code_block3 } else {     // default_code } // endif </pre>	<pre> if (tvType == "color") {     if (size == 14) {         discPercent = 8;     } else {         discPercent = 10;     } //endif } //endif . . . . . if (tvType == "color") {     if (size == 14) {         discPercent = 8;     } //endif } else {     discPercent = 10; } //endif </pre>
---	--

### 5.3. switch Statement

<pre> switch (&lt;variable or expression&gt;) {     case &lt;literal value&gt;: //code_block1                         [break;]     case &lt;literal value&gt;: // code_block2                         [break;]     default: //default_code } //end switch </pre>	<p>switch : variable int, short, byte, char, or String</p> <p>case : valor</p> <p>break : Salir del switch</p> <p>default : No encuentra relacion</p>
--	---

<p><b>Solution: if/else Statement</b></p> <pre> Scanner in = new Scanner(System.in); System.out.println("Enter your grade"); int grade = in.nextInt(); if (grade == 9){     System.out.println("You are a freshman"); } else if (grade == 10) {     System.out.println("You are a sophomore"); } else if (grade == 11) {     System.out.println("You are a junior"); } else if (grade == 12) {     System.out.println("You are a senior"); } else {     System.out.println("Invalid grade"); } //endif </pre>	<p><b>Solution: switch Statement</b></p> <pre> Scanner in = new Scanner(System.in); System.out.println("What grade are you in?"); int grade = in.nextInt(); switch (grade) {     case 9:         System.out.println("You are a freshman");         break;     case 10:         System.out.println("You are a sophomore");         break;     case 11:         System.out.println("You are a junior");         break;     case 12:         System.out.println("You are a senior");         break;     default:         System.out.println("Invalid grade"); } //end switch </pre>
---	--



## What Is switch Fall Through?

- switch fall through is a condition that occurs if there are no break statements at the end of each case statement
- All statements after the matching case label are executed in sequence, regardless of the expression of subsequent case labels, until a break statement is encountered.

```
int month = 8;
month = in.nextInt();

switch (month) {
    case 1: case 3: case 5: case 7:
    case 8: case 10: case 12: System.out.print("31 days");
                                break;
    case 2: if(isLeapYear)){
        ..
    }
```

Only a single value can be tested

Known values

→

## 6. Loop Constructs

### 6.1. for Loops

El numero de ciclos o iteraciones es conocido

La inicialización de la variable solo se ejecuta la primera vez.

La ultima instruccion que se ejecuta **dentro** del ciclo es el incremento o decremento, posteriormente vuelve a iterar **mientras** se cumpla la condición.

#### for Loop Overview

- Syntax:

```
for(initialization; condition; update){  
    Code statement(s)  
    Code statement(s)  
} //end for
```

Header

Body

```
for ( ; ; ){  
    System.out.println("Al infinito  
                        y mas allá");  
}
```

```
System.out.println("Countup to Song: ");  
for (int i = 1; i < 9; i++) {  
    System.out.println(i);  
    // incremento implicito  
} //end for  
System.out.println("Mambo!");
```

```
System.out.println("Countdown to Launch: ");  
int i; // Scope  
for (i = 10; i >= 0; i--) {  
    System.out.println(i);  
} //end for  
System.out.println("Despegamos!: " + i );
```

#### Variable Scope

Variables cannot exist before or outside their block of code.

```
public class VariableScopeDemoClass{  
    int x = 0;  
  
    public static void main(String args[]){  
        int i = 1;  
  
        for(int j = 2; j <= 5; j++){  
            int k = 3;  
            System.out.println(x + i + j + k);  
        }  
    }  
}
```

X

i

j

k

```

import java.util.Scanner;
public class PracticeCode {
public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int N = 100;
    int total = 0;
    System.out.println("This program adds " + N + " numbers.");
    for(int i = 0; i < N; i++){
        System.out.println("Enter your next number:");
        int value = in.nextInt();
        total += value;
    } //end for
    System.out.println("The total is " + total + ".");
} //end method main

```

#### Variable Already Defined

```

public static void main(String[] args) {
    int i = 0;
    for(int i = 64; i > 0; i=i/2 ){
        System.out.print(i + " ");
    }
}

```

#### Out of Scope

```

public static void main(String[] args) {
    for(int j = 0; j <= 5; j++){
        System.out.print(j + " ");
    }
    for(int j = 5; j >= 0; j--){
        System.out.print(j + " ");
    }
    for(int k = 2; k <= 64; k=k*2){
        System.out.print(k + " ");
    }
}

```

## 6.2. while and do-while loops

### How Many Times to Repeat?

- In some situations, you don't know how many times to repeat something
- That is, you may need to repeat some code until a particular condition occurs

#### Pre-test

```

while (<boolean expression> ) {
    <statement(s)> ;
} //end while

```

#### Post-test (mínimo una vez)

```

do{
    <statement(s)>
}while(<condition>);

```

The do-while loop requires a **semicolon** after the condition at the end of the loop

```

int i = 10;
System.out.println("Countdown to Launch!");
while (i >= 0) {
    System.out.println(i);
    i--; // i++;
} //end while
System.out.println("Blast Off!");
. . . . .
int i = 10;
System.out.println("Countdown to Launch!");
do {
    System.out.println(i);
    i--;
} while (i >= 0);
System.out.println("Blast Off!");

```

## Standard for Loop Compared with while Loop

```
for (int i = 10; i >= 0; i--) {  
    System.out.println(i);  
}  
System.out.println("Blast Off!");
```

```
int i = 10;  
while (i >= 0) {  
    System.out.println(i);  
    i--;  
}  
System.out.println("Blast Off!");
```

```
Scanner console = new Scanner(System.in);  
int sum = 0;  
  
System.out.println("Enter a number (-1 to quit): ");  
int num = console.nextInt();  
while (num != -1) {  
    sum = sum + num;  
    System.out.println("Enter a number (-1 to quit): ");  
    num = console.nextInt();  
} // end while  
System.out.println("The sum is " + sum);
```

### 6.3. Using break and continue Statements

Use a **continue** statement to skip part of a loop      up  
Use a **break** statement to exit a loop      down  
Se pueden usar en cualquier ciclo: for, while, do while

```
while(condition){  
    statement1;  
    statement2;  
    continue;  
    statement3;  
    statement4;  
}  
statement; [statement outside the while loop]
```

Control passes to the loop condition

These statements are skipped in the current iteration

```
while(condition){  
    statement1;  
    statement2;  
    break;  
    statement3;  
    statement4;  
}  
statement; [statement outside the while loop]
```

Control passes to the statement outside the loop

```
int i = 0;  
while (i < 10) {  
    if (i == 4) {  
        break;  
    }  
    System.out.println(i+ "\t");  
    i++;  
}  
System.out.println("\n. . .Fin");
```

→

## 7. Creating Classes

- 7.1. Creating a Class
- 7.2. Instantiating Objects
- 7.3. Constructors
- 7.4. Overloading Methods
- 7.5. Object Interaction and Encapsulation
- 7.6. static Variables and Methods

## 8. Arrays and Exceptions

- 8.1. One-dimensional Arrays
- 8.2. ArrayLists
- 8.3. Exception Handling
- 8.4. Debugging Concepts and Techniques

## 9. JavaFX

- 9.1. Introduction to Java FX
- 9.2. Colors and Shapes
- 9.3. Graphics, Audio and MouseEvents

→

Juan Carlos Herrera H.