JAVA FOUNDATIONS 1Z0-811

ORACLE ACADEMY

2 DE SEPTIEMBRE DE 2025

https://academy.oracle.com/

HTTPS://WWW.YOUTUBE.COM/USER/ORACLELEARNING

HTTPS://WWW.ORACLE.COM/LATAM/EDUCATION/

HTTPS://GITHUB.COM/ISC-UPA/2025-3-TIID3C-POO

Juan Carlos Herrera H.

Contenido

Juan Carlos Herrera H.

→

Juan Carlos Herrera H.

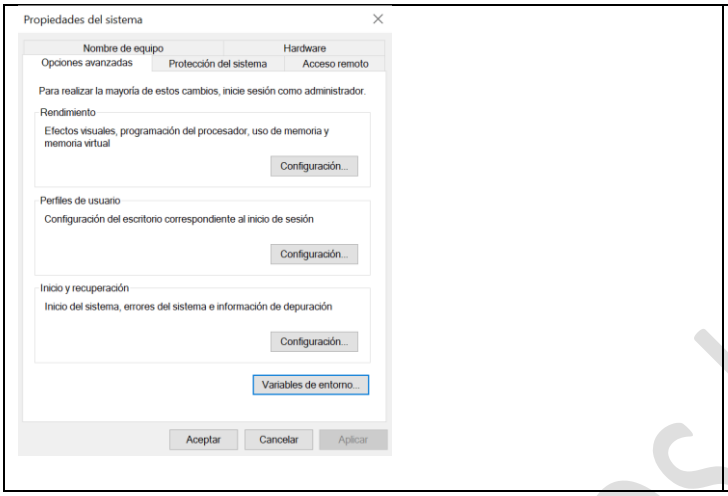
1. Introduction

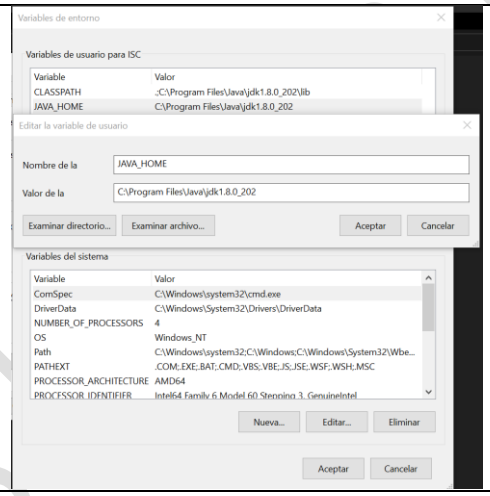
1.1. Technological Requirements:

Java JDK https://www.oracle.com/java/technologies/downloads/ jdk-8u202-windows-x64.exe
VS Code https://code.visualstudio.com/Download VSCodeSetup-x64-1.103.2.exe
Extensions: **Extension Pack for Java**

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 -> Configuracion 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!
```

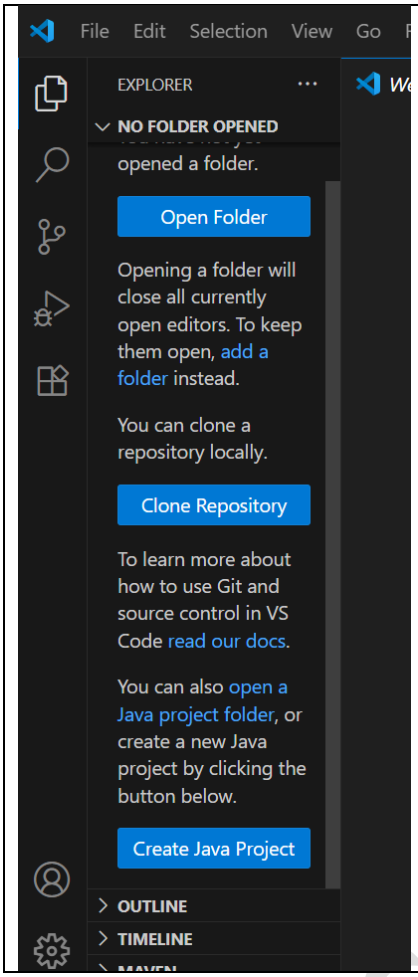
```
public class Hola {

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

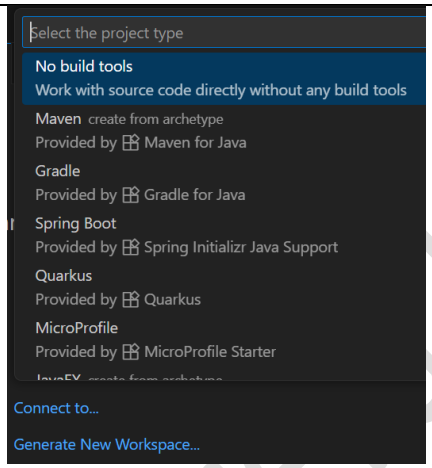
→

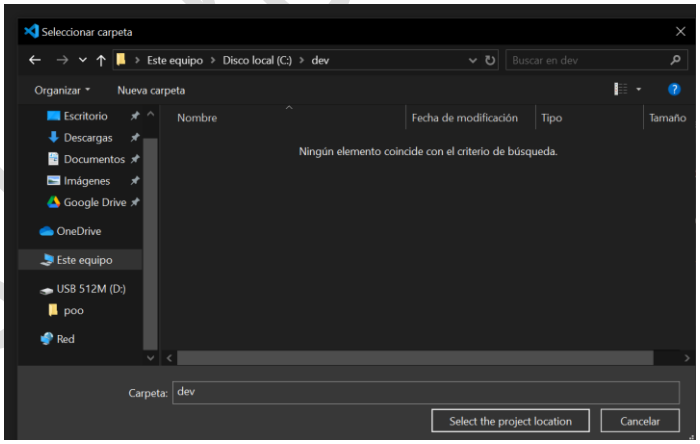
Juan Carlos H.

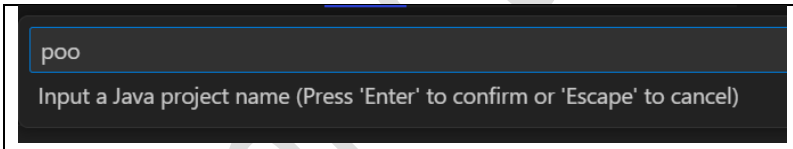
Crear un Directorio de trabajo
C:\> mkdir dev
C:\dev>

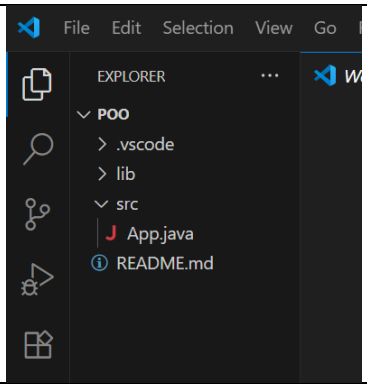
1.2. Create Java Project:

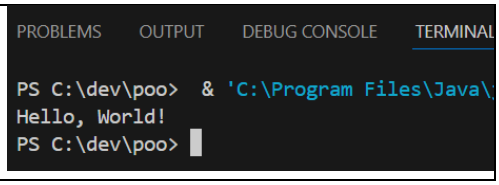












→

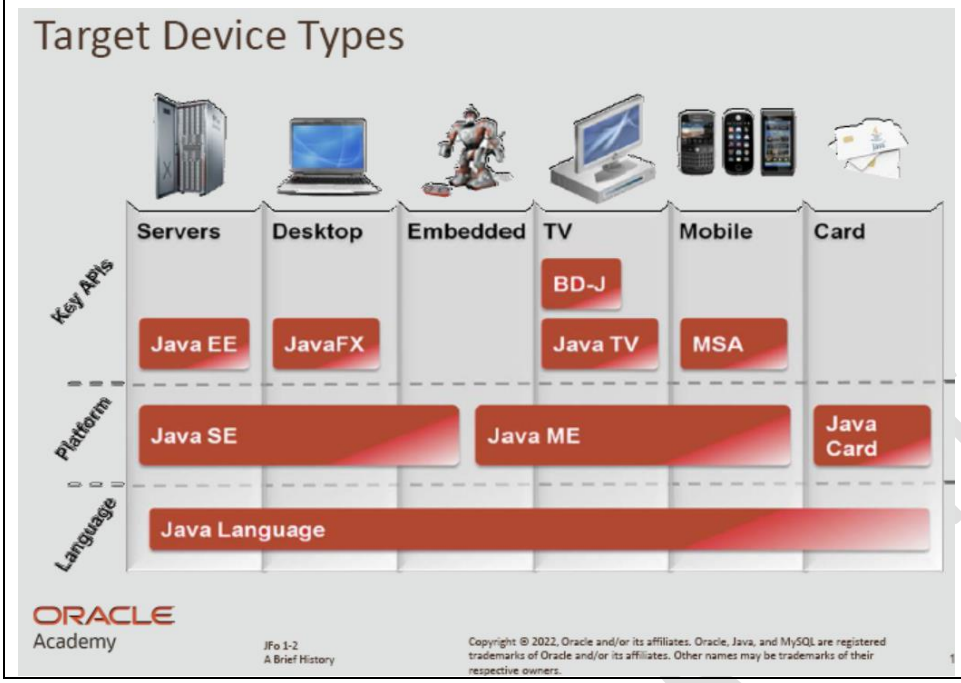
Juan Carlos Ruiz H.

1.3. Setting Up Java

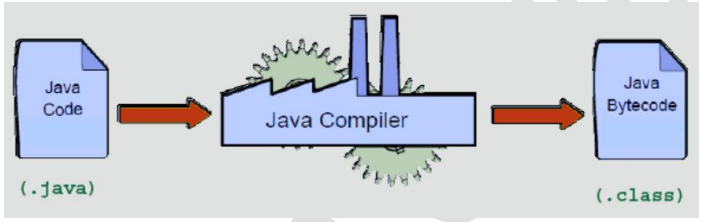
James Gosling is considered the “Father of Java”. Duke, the Java Mascot.

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

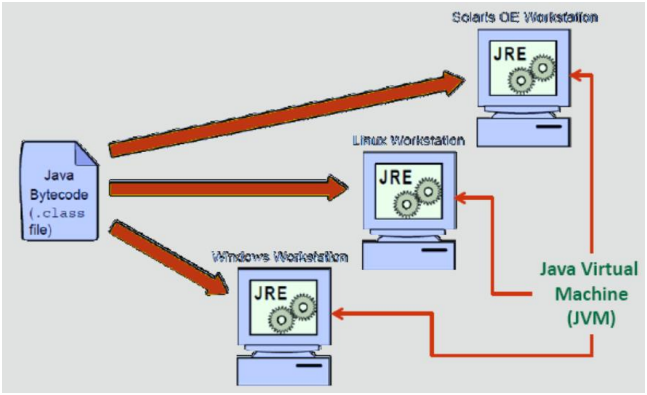


Java is Platform-Independent



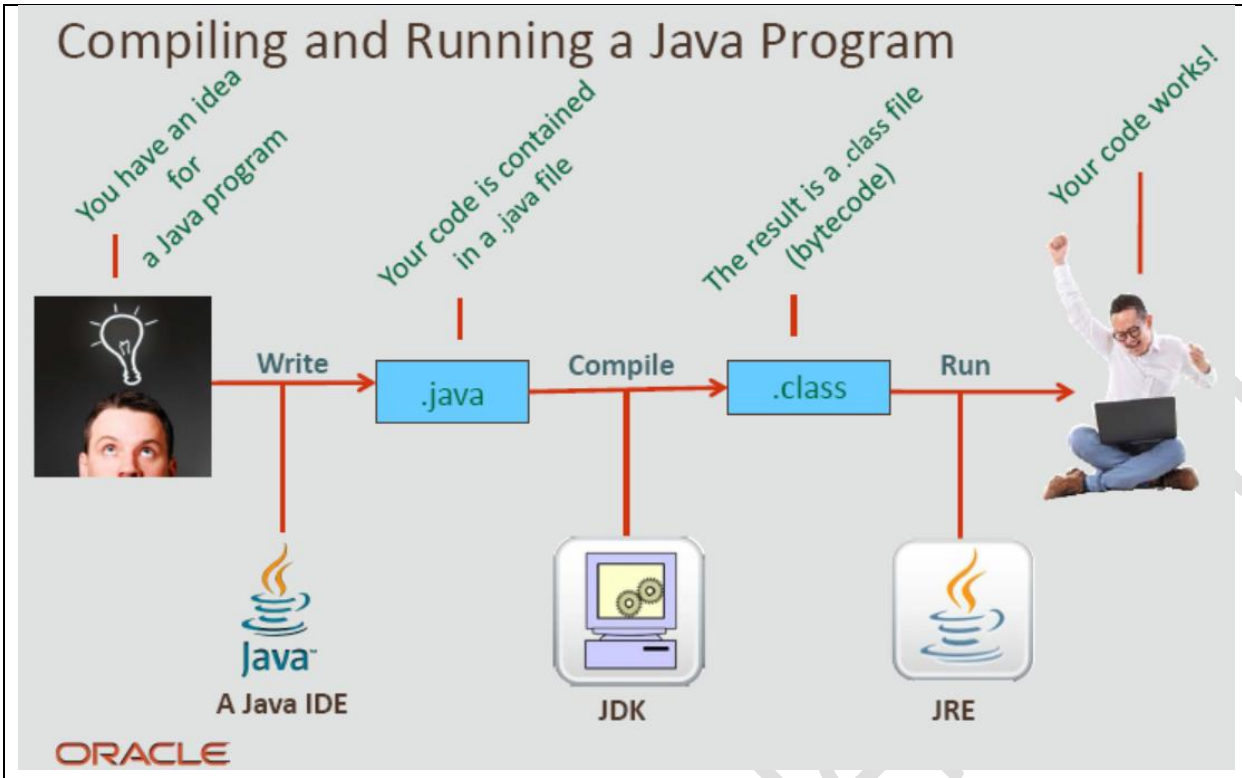
-

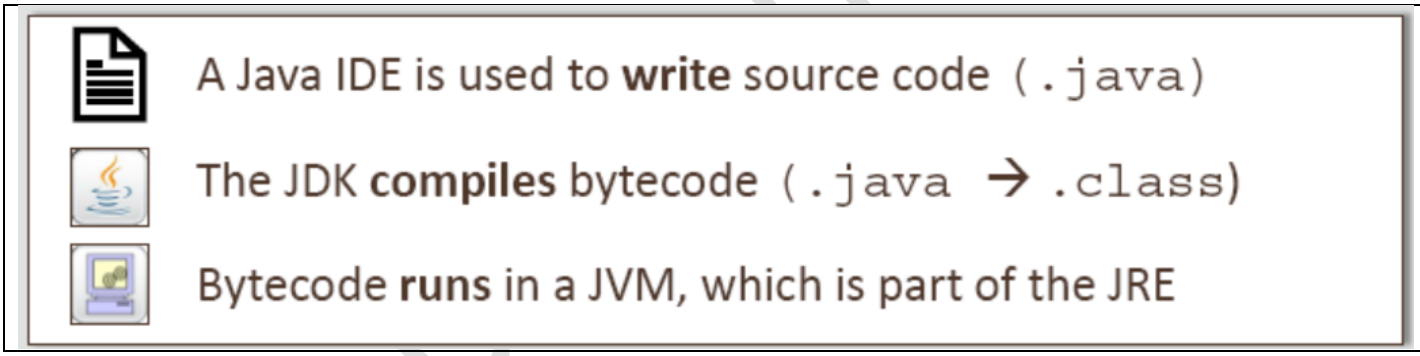
Java Programs Run in a JVM



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)

Juan Carlos Olivera H.


Compiling and Running a Java Program
You have an idea for a Java program
Your code is contained in a .java file
The result is a .class file (bytecode)
Your code works!
Write
.java
Compile
.class
Run
Java
A Java IDE
JDK
JRE
ORACLE

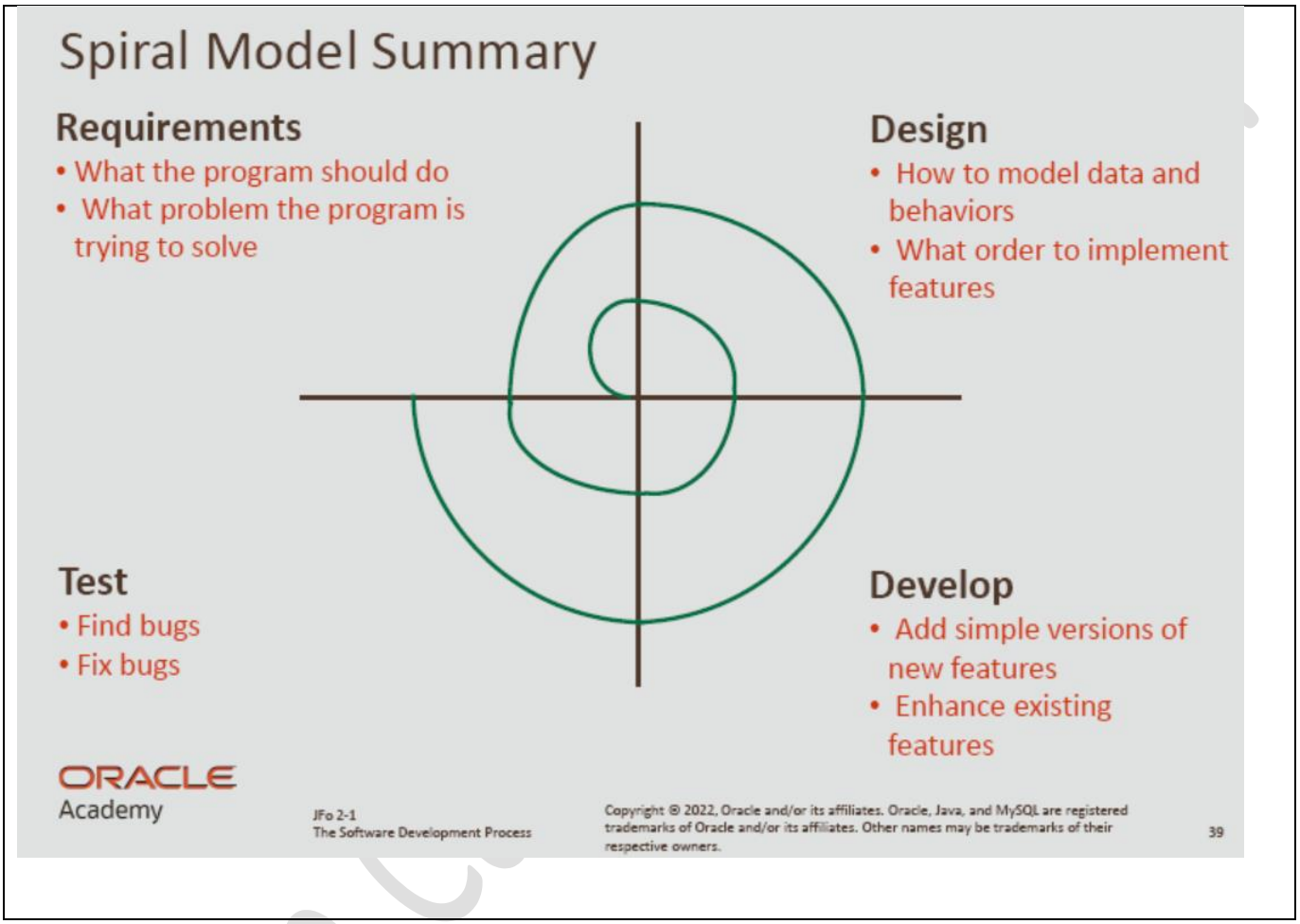

A Java IDE is used to write source code (.java)
The JDK compiles bytecode (.java → .class)
Bytecode runs in a JVM, which is part of the JRE

→

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

→

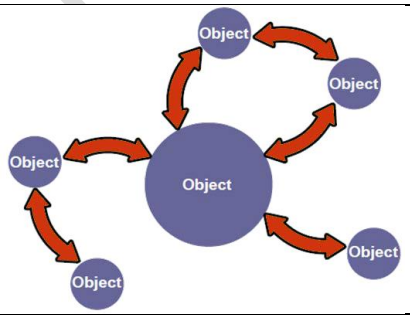
Juan C

2.2. What is my Program Doing?

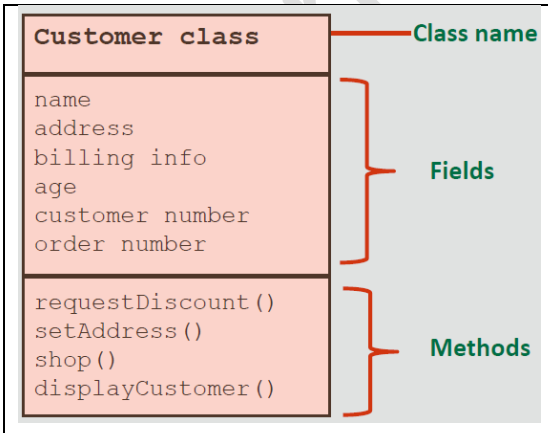
Code within curly braces is called a block of code Indentation before a line of code (4 spaces) Whitespace End statements with semicolons (;) // Single-line comments Multi-line comments /* Bievenidos a poo */	public static void NombreMetodo() { . . } NombreMetodo(); // llamar al método Debug To set a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

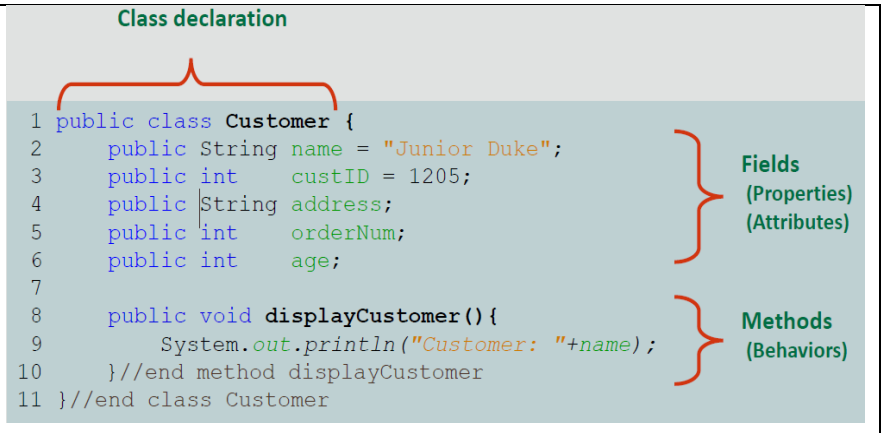
2.3. Introduction to Object-Oriented Programming Concepts

Procedural languages … • Read one line at a time • The C language is procedural	Object-oriented languages… • Read one line at a time • Model objects through code • Emphasize object interaction • Allow interaction without a prescribed order • Java and C++ are object-oriented languages

Object-Oriented Programming • Interaction of objects • No prescribed sequence	

Modeling Properties and Behaviors





Quiz: JFo - Section 2 Questions 15

→

Carlos Herrera H.

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	boolean	true, false
Integer	int	1, -10, 20000, 123_456_789
Double	double	1.0, -10.0005, 3.141
String	String	"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
Byte	8 bits	2^8, or… 256	-2^7, or… −128	2^7-1, or… 127
short	16 bits	2^{16}, or… 65,535	-2^{15}, or… −32,768	$2^{15}-1$, or… 32,767
int	32 bits	2^{32},or… 4,294,967,296	-2^{31}, or… −2,147,483,648	$2^{31}-1$, or… 2,147,483,647
long	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)

Carlos Herrera

// 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?
`float`	32 bits	Never
`double`	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;
```

→

Juan Carlos Herrera

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!"; // Asignación Hard-coding
	String Literal

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

Where are Strings?

String is capitalized

- Strings are an object, not a primitive
- Object types are capitalized by convention

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
\t	Insert a new tab
\b	Insert a backspace
\n	Insert a new line
\r	Insert a carriage return
\f	Insert a formfeed
\'	Insert a single quote character
\"	Insert a double quote character
\\	Insert a backslash character

System.out.print**ln**("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

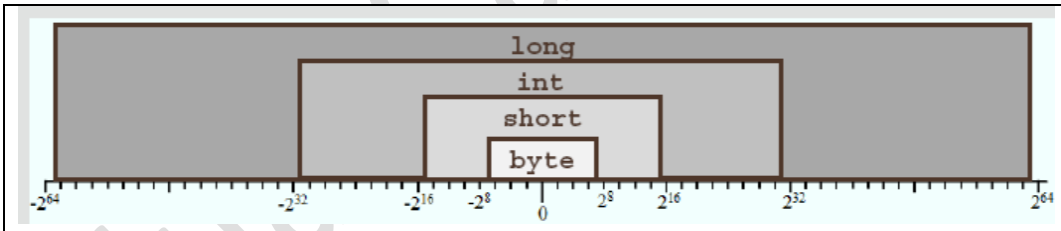
```
System.out.println("This is the first line."
                + "This is NOT the second line.");
```

sout tab "Metodo abreviado"

3.4. Converting Between Data Types

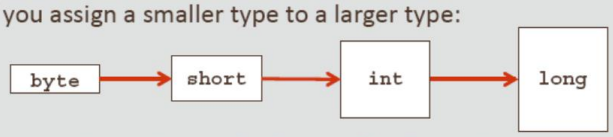
```
double x = 9 / 2; // Should be 4.5
System.out.println(x); // prints 4.0

double y = 4;
System.out.println(y); //prints 4.0
```

```
int    num1 = 7;
double num2 = 2;
double num3;
num3 = num1 / num2; // num3 is 3.5
```

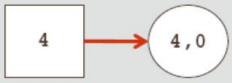


Juan Carlos Herrera H.

- Automatic promotions:
  - If you assign a smaller type to a larger type:



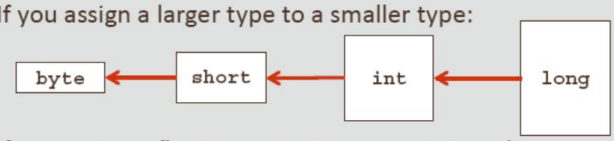
  - If you assign an integral value to a floating-point type:



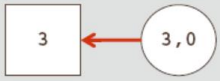
- Examples of automatic promotions:

```
-long intToLong = 6;
-double intToDouble = 4;
```

- When to cast:
  - If you assign a larger type to a smaller type:



  - If you assign a floating point type to an integral type:



- Examples of casting:

```
-int longToInt = (int)20L;
-short doubleToShort = (short)3.0;
```

double pi = 3.1416
int entero = (int) pi

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

- Solution using larger data type:

```
int num1 = 53;
int num2 = 47;
int num3;
num3 = (num1 + num2);
```

Changed from byte to int

- Solution using casting:

```
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
```

Automatic Promotion

- Example of a potential problem:

```
short a, b, c;
a = 1 ;
b = 2 ;
c = a + b ; //compiler error
```

a and b are automatically promoted to integers

- Example of potential solutions:
  - Declare c as an `int` type in the original declaration:
    - `int c;`
  - Type cast the (a+b) result in the assignment line:
    - `c = (short)(a+b);`

int x = 123_456_789;
int x = 123456789;

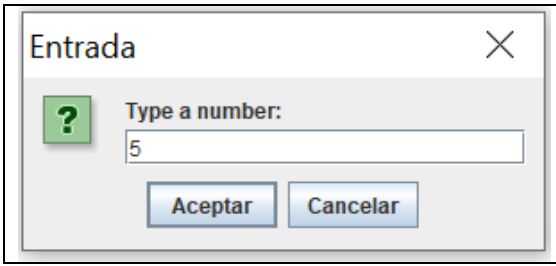
intintVar1 = Integer.parseInt("100");
doubledoubleVar2 = Double.parseDouble("2.72");

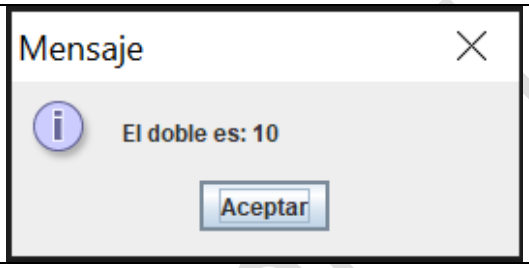
→

Juan Carlos Herrera H.

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. The Scanner class accepts input in tokens form.

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

→

Juan Carlos Herrera

4. Java Methods and Library Classes

4.1. What Is a Method?

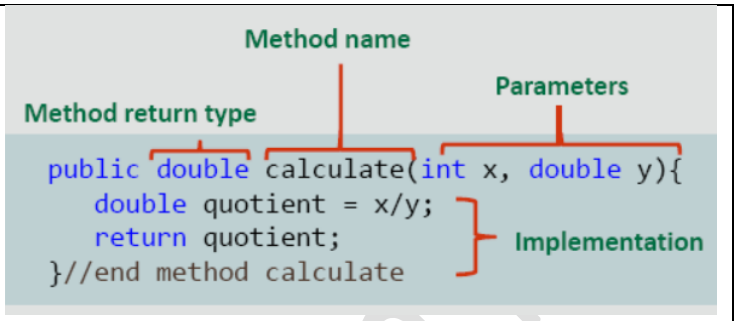
Instantiate an object

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();
```

type — name — value




```
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
```

3 — 2.0 — Parameters

Method Return Types

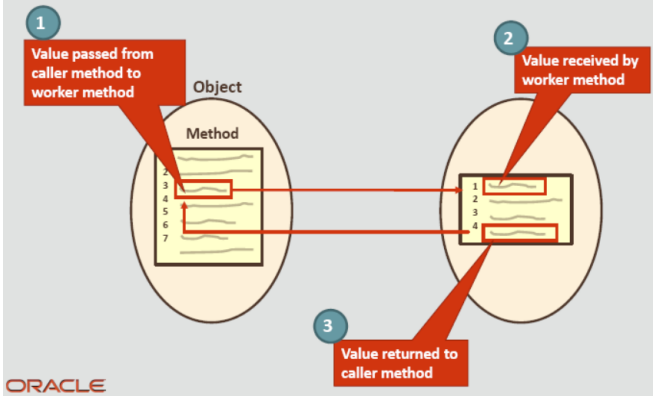
- Variables can have values of many different types:

int, double, long, char, float, byte, short, String, boolean, Calculator

- Method calls also return values of many different types:

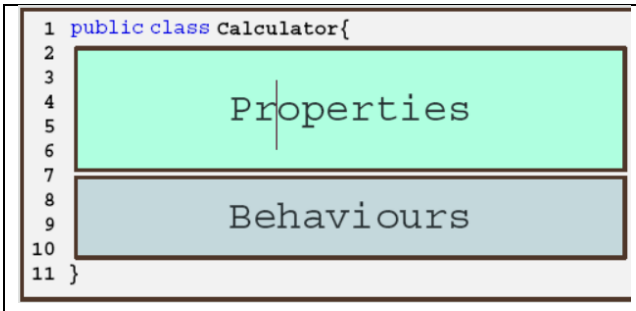
int, double, long, char, float, byte, short, String, boolean, Calculator

Passing Arguments and Returning Values



two operatoins are appropiate for the main method:

Creating instances of objects
Calling an instance object's field and methods.




```
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
```

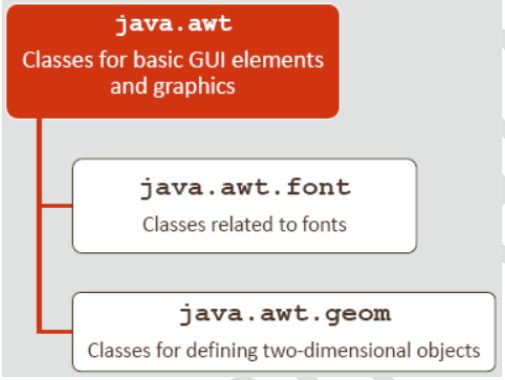
Calculator calc = **new** Calculator();

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

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

Package	Purpose
java.lang	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 contains Math class
javax.swing	Provides classes to build GUI components
java.net	Provides classes for networking applications
java.time	Provides classes for dates, times, instants, and durations



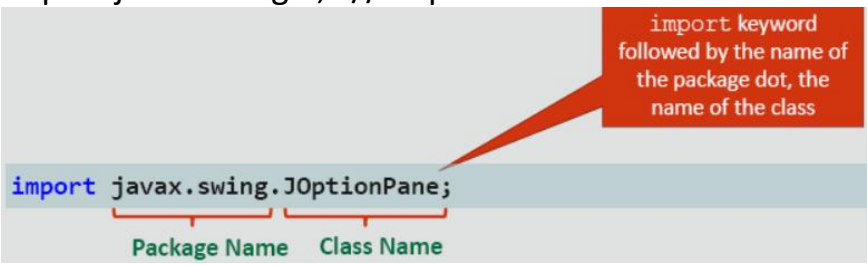
Which is a risk of using fully qualified class names when importing?
Code readability is reduced. Reduces readability and clutters code

e.g., java.util.ArrayList instead of simply ArrayList

Use fully qualified names only when necessary to resolve conflict

import

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



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

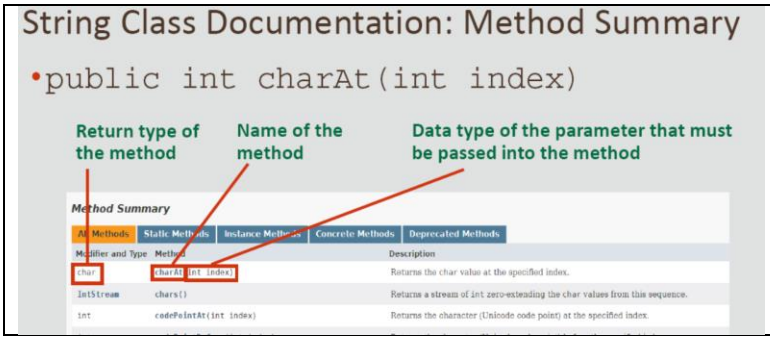
→

Juan Luis Herrera H.

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 str = "Hello, World";
```

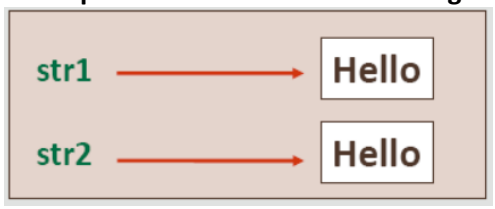
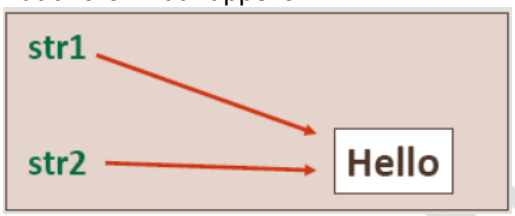
H	e	l	l	o	,		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10	11

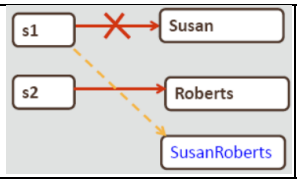
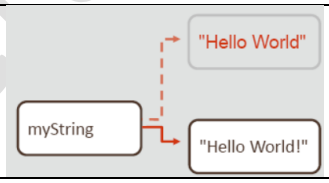
String str = new String("Hello, World");
Not commonly used and not recommended

int length()	Returns the length of this string Example: LastName.length()
char charAt(int index)	Returns the char value at the specified index
String concat(String str)	Concatenates the specified string to the end of this string. String producto = "coca"; producto.concat(" cola"); producto= producto.concat(" cola"); producto = producto +" cola";
boolean contains(CharSequence s)	Returns true if and only if this string contains the specified sequence of char values.
int indexOf(String str)	Returns the index within this string of the first occurrence of the specified substring
int indexOf(char c)	Returns the index value of the first occurrence of c
int indexOf(char c, int beginIdx)	Returns the index value of the first occurrence of c, starting from beginIdx to the end of the string
String substring(int beginIdx)	Returns the substring from beginIdx to the end of the string
String substring(in tbeginIdx, int endIdx)	Returns the substring from beginIdx up to, but not including endIdx
String replace(char oldChar, char newChar) String replace(CharSequence target, CharSequence replacement)	This method replaces **all** occurrences of matching characters in a string
replaceFirst(String pattern, String replacement)	replaces only the first occurrence of a matching character pattern in a string
int lastIndexOf(String str) int lastIndexOf(String str, int fromIndex) String trim() String toLowerCase() String toUpperCase()	Investigar que hacen las siguientes funciones cadena = "coca cola toma lo bueno" Realizar el programa que regrese el número de palabras de cadena

Juan Carlos Herrera H.

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

String str1 = "Hello"; **String str2 = "Hello";** **We expect this . . . . . but it's wrong** 	But this is what happens . . .  The Java runtime system knows that the two strings are identical and allocates the same memory location for the two objects.

String s1 ="Susan"; String s2 = "Roberts"; s1 = s1 + s2;		String myString = "Hello"; myString = myString.concat(" World"); myString = myString + "!"	

Comparing String

ASCII
'0' = 48
'1' = 49
'A' = 65
'a' = 97
 	
Practica: Imprimir el abecedario
	// Conversiones explícitas
	int ascii = (int) 'A';
	char character = (char) ascii;
	
	// Conversiones implícitas
	int ascii1 = 65;
	int ascii2 = 'A';
	char caracter1 = 65;
	char caracter2 = 'A';

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

→

Juan Carlos Herrera H.

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; range [min: max] inclusive
(int) (Math.random() * (max - min + 1)) + min;

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

→

Juan Carlos Herrera H.

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,** without creating an instance.
Common math functions like square root are taken care of in the language.

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
asin(value)	return radians
sqrt(value)	square root

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

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

BMI = Peso en libras / Altura en pulgadas^2 * 703 **IMC = Peso (kg) ÷ (Altura (m))^2**

Sen(30º) = 0.5 arcsen(0.5) = 30º $sen^{-1}(0.5) = 30º$ asin(0.5) = 30º

Quiz 1: JFo - Section 4 - L1-L2
Quiz 2: JFo - Section 4 - L3-L5

→

Juan Carlos Herrera H.

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.

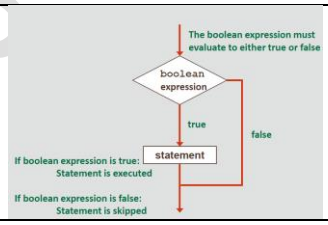
boolean bandera = true;
int x = 4;
boolean isFive = x == 5;

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)

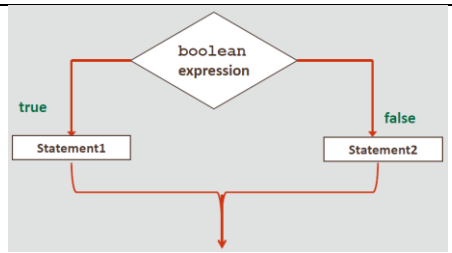
Conditional statements in Java are:
if statement
if/else statement
switch statement

```
if ( <some condition is true> ){

   //Statements will execute if the boolean
   //expression is true
}//endif
```




```
boolean expression
if ( <some condition is true> ) {
      // do something
}
else {
      // do something different
}//endif
```

if block
else block



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

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

Juan Carlos Herrera H.

5.2. Understanding Conditional Execution

Handling Multiple Conditions

```
int grade = 90;
int numberDaysAbsent = 0;

if (grade >= 88) {
    if (numberDaysAbsent == 0) {
        System.out.println("qualify");
    } // endif
} // endif
```

```
int grade = 90;
int numberDaysAbsent = 0;

if ((grade >= 88) && (numberDaysAbsent == 0)) {
   System.out.println("qualify");
} // endif
```

Logic Operator	Meaning
&&	AND
||	OR
!	NOT

```
boolean bandera = true;
if (bandera) {
    System.out.println("qualify");
} else {
    System.out.println("fail");
}
```

```
boolean bandera = true;
if (!bandera) {
    System.out.println("fail");
} else {
    System.out.println("qualify");
}
```

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

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

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

Ternary Conditional Operator

Operation	Operator	Example

If condition is true:
assign result = value1
Otherwise:
assign result = value2
	?:	`result = condition ? value1 : value2`
		Example:
		`int x = 2, y = 5, z = 0;`
		`z = (y < x) ? x : y;`

```
int numberOfGoals = 1;
System.out.println("I scored " + numberOfGoals + " "
        + (numberOfGoals == 1 ? "goal" : "goals"));
```

Juan Carlos Herrera H.

```
if (<condition1>){
    //code_block1
} else if (<condition2>){
    // code_block2
} else if (<condition3>){
    // code_block3
} else {
    // default_code
} // endif
```

```
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
```

5.3. switch Statement

```
switch (<variable or expression>) {
    case <literal value>: //code_block1
                          [break;]
    case <literal value>: // code_block2
                          [break;]
    default: //default_code
 }//end switch
```

switch : variable int, short, byte, char, or String

case : valor

break : Salir del switch

default : No encuentra relacion

Solution: if/else Statement

```
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
```

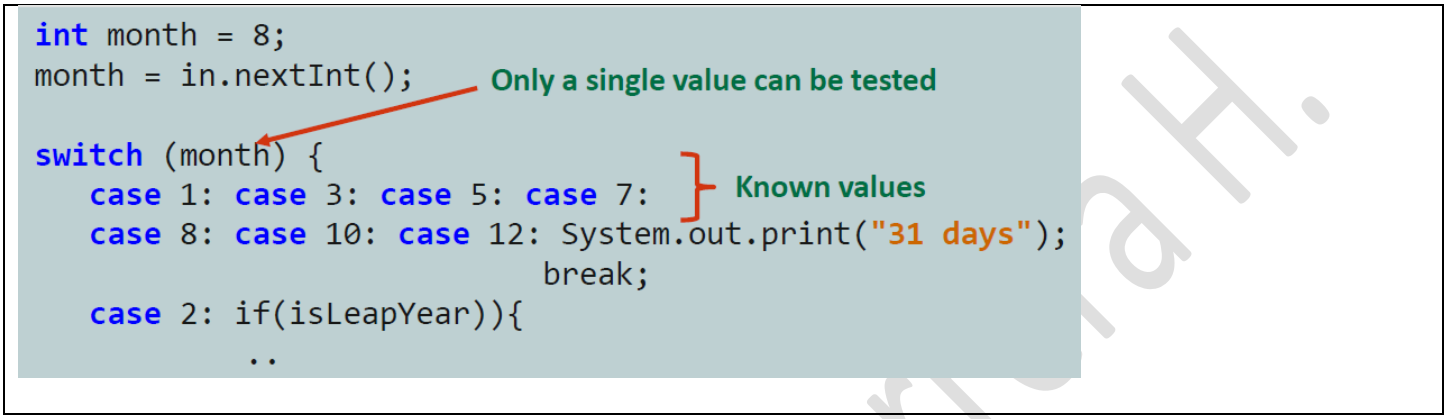
Solution: switch Statement

```
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
```

Carlos Herrera H.

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.



about the default statement:

When the input does not match any of the cases, the default statement is executed.
The default statement is optional in switch statement.

Quiz: JFo - Section 5

→

Juan Carlos Herrera H.

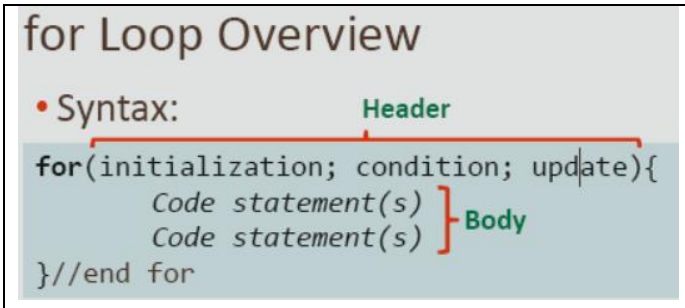
6. Loop Constructs

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.

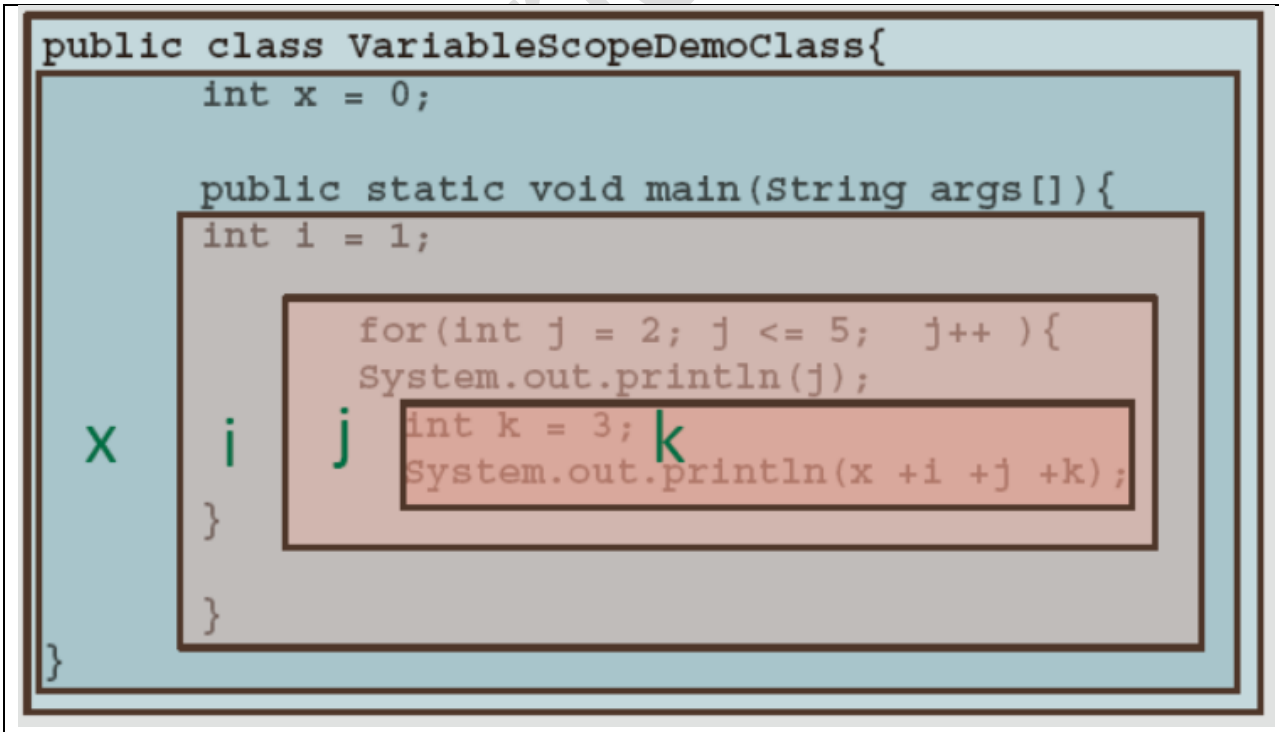



```
for ( ; ; ){
   System.out.println("Al infinito
                       y mas allá");
}
```

```
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.



Los Herrera H.

```
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
```

N
total
i
value

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(j+" ");
    }
}
```

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!");
```

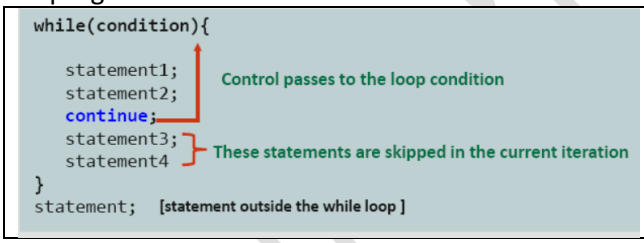
```
int i = 10;
while (i >= 0) {
   System.out.println(i);
   i--;
}
System.out.println("Blast Off!");
```

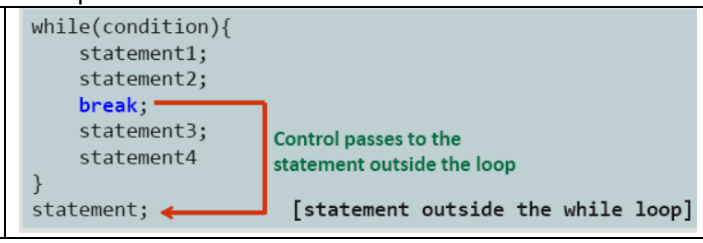
```
Scanner console = new Scanner(System.in);
int sum = 0;

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
Use a **break** statement to exit a loop
Se pueden usar en cualquier ciclo: for, while, do while

When a break statement is executed inside a loop, the loop-statement is terminated immediately, the execution of the program will continue with the statement following the loop-statement.






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

Quiz: JFo - Section 6

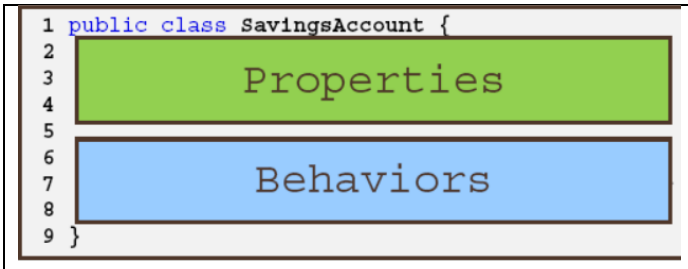
→

Juan Carlos Herrera H.

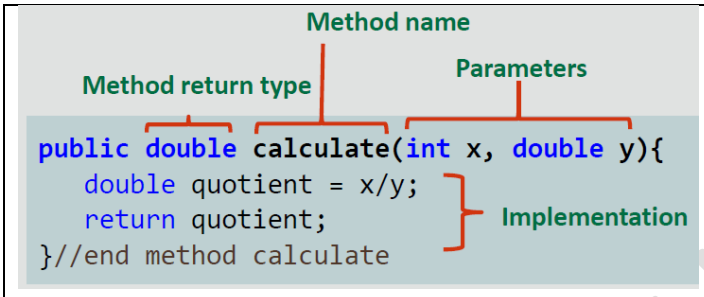
7. Creating Classes

7.1. Creating a Class

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




```
public class SavingsAccount {
    public double balance;
    public double interestRate = 0.01;
    public String name;

    public void displayCustomer(){
      System.out.println("Customer: "+ name);
    }//end method displayCustomer
}//end class SavingsAccount
```



0–11 months : 0,5%
12–23 months : 1,0%
24–35 months : 1,5%
36–47 months : 2,0%
48–60 months : 2,5%

En base a t(tiempo) obtener el rate(porcentaje)

```
Public void setTermAndRate(int t){
   if(t>=0 && t<12)
      rate= 0.005;
   else if(t>=12 && t<24)
      rate= 0.010;
   else if(t>=24 && t<36)
      rate= 0.015;
   else if(t>=36 && t<48)
      rate= 0.020;
   else if(t>=48 && t<=60)
      rate= 0.025;
   else {
      System.out.println("Invalid Term");
      t = 0;
   }
   term= t;
}
```

→

Juan Carlos Hernandez

```java
public class Cuenta {
    int numeroID;  // Numero de Tarjeta
    String titular;
    double saldo;

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double monto) {
        saldo += monto;
    }

    public void retirar(double monto) {
        if (monto <= saldo) {
            saldo = saldo - monto;
        } else {
            System.out.println(
                 "Sin suficiente saldo");
        }
    }
```

```java
public static void main(String[] args) {
    Cuenta cuenta1 = new Cuenta();
    cuenta1.numeroID = 1;
    cuenta1.titular = "Jesus";
    cuenta1.saldo = 1000;  // warning
    cuenta1.depositar(500);

    //Cuenta cuenta2 = new Cuenta(2, "Maria", 2000);
    //Cuenta cuenta3 = new Cuenta(3, "Jose",  3000);

    System.out.println(cuenta1.getSaldo());

    Cuenta[] cuentas = new Cuenta[3];
    cuentas[0] = cuenta1;
```

7.2. Instantiating Objects

Understand object references.
Understand the difference between stack and heap memory
Understand how Strings are special objects

```java
int x;
int y;

x = y;

x = 1;
y = 2;

System.out.println(x);
System.out.println(x == y);
```

¿Que imprime?

Strings Are Special Objects.

Strings should be instantiated without new
This is more memory-efficient

```java
String s1 = "Test";
```

But you shouldn't do this
String s2 = new String("Test");

String s3 = "Test";

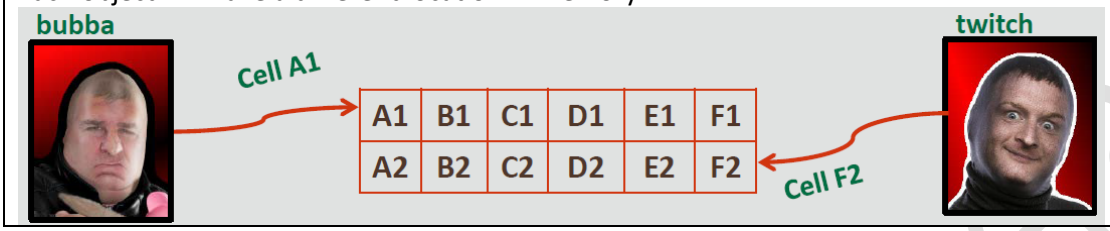
System.out.println(s1 == s2); ??
System.out.println(s1 == s3); ??

```java
public class Prisoner {
   public String name;
   public double height;
   public int sentence;
   LocalDate birthDate;
   LocalDate entryDate;
}
```

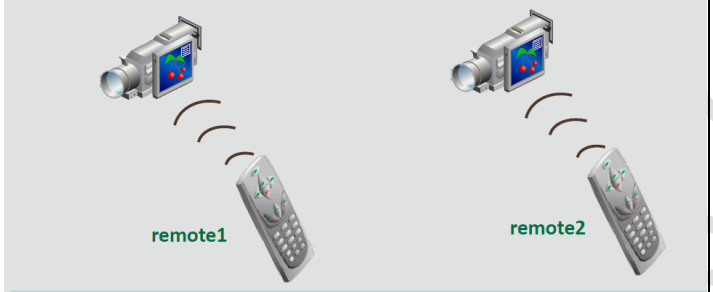
```java
public static void main(String[] args){
   Prisoner bubba  = new Prisoner();
   Prisoner twitch = new Prisoner();
   System.out.println(bubba); // Prisoner@15db9742
   System.out.println(twitch);// Prisoner@6d06d69c
                              // Memory addresses
}
```

Juan Carlos Herrera H.

Each object will have a different location in memory



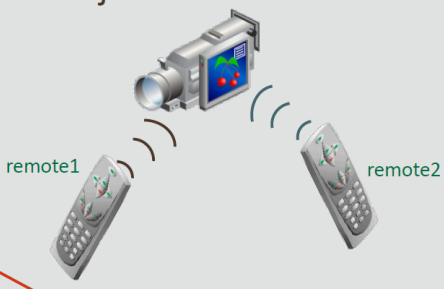
Working with Object References: Example 1




```
Camera remote1 = new Camera();
Camera remote2 = new Camera();

remote1.play();
remote2.play();
```

There are two Camera objects

Working with Object References: Example 2




```
Camera remote1 = new Camera();

Camera remote2 = remote1;

remote1.play();
remote2.stop();
```

References to Different Objects: Example

Reference variable

Reference type — Object type

```
Camera remote1 = new Camera();
remote1.menu();

TV remote2 = new TV();
remote2.menu();

Prisoner bubba = new Prisoner();
bubba.think();
```

References to Different Objects: Example

```
Prisoner twitch = new TV();
```

Caballo extends Animal { … } // El caballo **ES UN** Animal
Animal potro = new Caballo();

Stack memory is used to store …

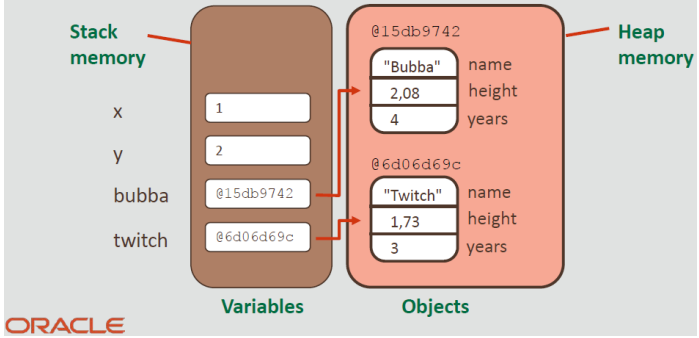
- Local variables
- Primitives
- References to locations in the heap memory

Heap memory is used to store …

- Objects

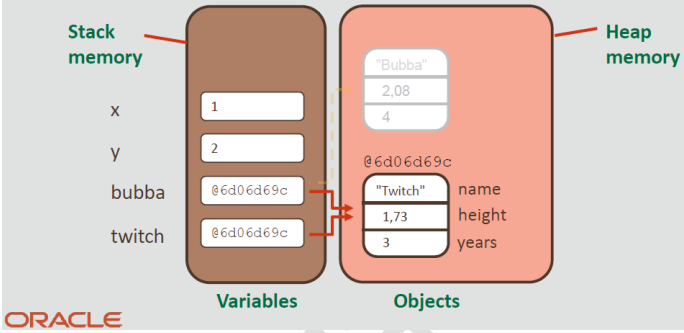
References and Objects in Memory

```
int x = 1;
int y = 2;
Prisoner bubba = new Prisoner();
Prisoner twitch = new Prisoner();
…
```



Assigning a Reference to Another Reference

```
bubba = twitch;
```




```
bubba.name= "Bubba";
twitch.name= "Twitch";
System.out.println(bubba.name);        ??
System.out.println(twich.name);        ??
System.out.println(bubba == twitch);   ??
```

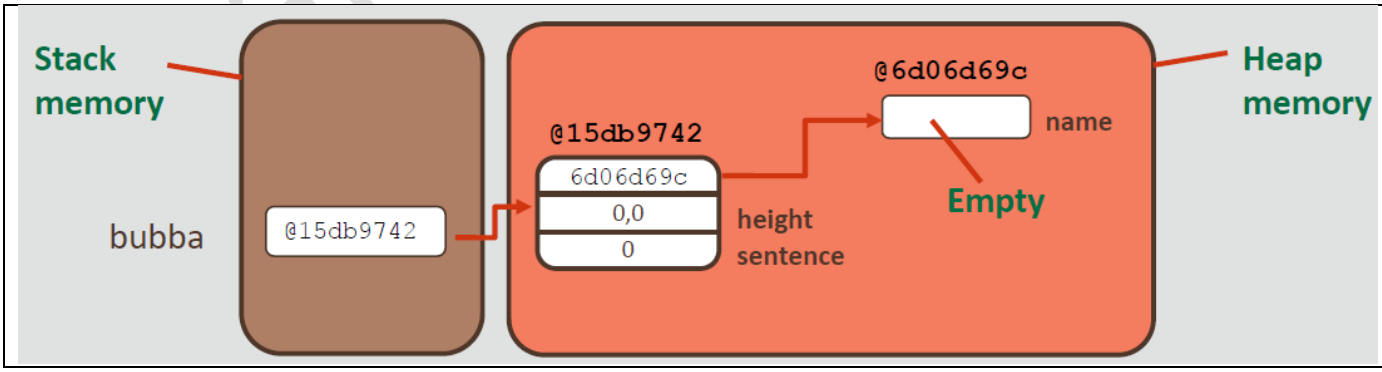
Si ninguna variable de referencia apunta a un objeto... Java borra automáticamente la memoria que ocupaba ese objeto. Esto se denomina recolección de basura (**Garbage Collection**). Los datos asociados a este objeto se pierden para siempre.

7.3. Constructors

Data Type	Default Value
boolean	false
int	0
double	0.0
String	null
AnyObject type	null

String test = null;
System.out.println(test.length()); // NullPointerException

```
public class Prisoner {
   public String name;
   public double height;
   public int sentence;
   LocalDate birthDate;
   LocalDate entryDate;

   public static void main(String[] args) {
      Prisoner bubba  = new Prisoner(); //vacio
      System.out.println(bubba);
      bubba.name = "Bubba";
   }
}
```



// Constructor
public Prisoner(String n, double h, int sentence) {
  name = n;
  height = h;
  this.sentence = sentence;
}
// They have no return type (not even void)
// They're named the same as the class
	Prisoner bubba = new Prisoner("Bubba", 2.08, 4);
	Prisoner twitch = new Prisoner("Twitch", 1.73, 3);

Summary of Constructors

- Are special methods in a class
- Named the same as the class
- Have no return type (not even void)
- Called only once during object instantiation
- May accept arguments
- Used to set initial values of fields
- If you don't write your own constructor, Java provides a default zero-argument constructor
- When you write your own constructor, the default constructor is no longer available

Overloading Constructors:

You can write more than one constructor in a class
- This is known as overloading a constructor
- A class may have an unlimited number of constructors

But they differ in any of the following ways:
- Number of parameters
- Types of parameters
- Ordering of parameters

Java developers do not need to know an object's location in memory.

Developers work with **references** to objects, not direct memory addresses.

Unlike languages like C or C++, Java **does not expose pointers** or allow direct memory manipulation, which helps prevent common bugs like buffer overflows or memory leaks

Quiz 1: JFo - Section 7 - L1-L3

→

Juan Carlos Herrera H.

7.4. Overloading Methods

Overloading Methods Summary:

- Have the same name
- Have different signatures:
    - The number of parameters
    - The types of parameters
    - The order of parameters

```
public class Calculator {
    // overloaded methods

    public int sum(int num1, int num2) {
        return num1 + num2;
    }

    // public double sum(int x, int y) {
    //     return x + y;
    // }

    public double sum(double num1, double num2) {
        return num1 + num2;
    }

    public double sum(int num1, double num2) {
        return num1 + num2;
    }

    public double sum(double num1, double num2, double num3) {
        // return num1 + num2 + num3;
        return sum(num1, num2) + num3;
    }

    public static void main(String[] args) {
        double result;
        Calculator calc = new Calculator();
        result = calc.sum(5, 10);
    }
}
```

→

Juan Carlos Herrera H.

7.5. Object Interaction and Encapsulation

Use the private modifier to define class variables.
Understand the purpose of getter methods.
Understand the purpose of setter methods.

The main method can access every objects' fields and methods. Interactions Between Objects: One object must know a reference to the other object.

Object references must be shared:
One object may contain another object as a field
One object's method may accept another object as an argument

A way to describe a Prisoner is by their Cell number. Cell class: String name, boolean isOpen, int securityCode;
- Should a Prisoner have a Cell property?
- Should a Cell have a Prisoner property?
- Should a Guard have a Cell property?
- Should a Guad have a Prisioner property?

Tenemos 3 Objetos: Celda, Prisionero, Guardia. ¿Qué propiedades debe tener cada objeto?

Access Modifier Details
- public: Visible to any class
  - It's the least secure
  - Methods are typically public
- Package: Visible to the current package
  - There's no keyword for this level of access
- private: Visible only to the current class
  - It's the most secure
  - Fields are typically private

public > protected > default > private.

Introducing Getter Methods	**Introducing Setter Methods**
Getters are also called accessors Getters are public Getters usually accept no arguments Getters return the value of a particular variable A prisoner should at least know their cell name.	Setters are also called mutators Setters are usually public Setters usually accept arguments Setters are void type methods A guard should be able to open a door, but a prisoner should not

Summary of Encapsulation
- Encapsulation offers techniques for limiting the visibility of a class
- Access and visibility should be limited as much as possible
- Most fields should be private
- Provide getter methods to return the value of fields
- Provide setter methods to safely modify fields

→

Juan Carlos Herrera H.

7.6. static Variables and Methods

static variables are variables that are shared by all instances of a class.
Static variables belong to the class, not to any individual instance.
Static variables can be accessed, even if no objects have been instantiated.
static final variables are variables that are shared by all instances of the class, but the value cannot change.

An object reference is used to access an object's fields and methods.

```
Prisoner p01 = new Prisoner()
p01.name       // Accessing a field
p01.display()  // Calling a method
```

We never need to instantiate a Math object.
Math fields and methods are accessed by directly referencing the Math class.
Static fields and methods are accessed by directly referencing the class

```
            // Nothing instantiated
Math.PI     // Accessing a static field
Math.sin(0) // Calling a static method
```

Unique instance variables exist for every instance of an object.

Creating Static Variables

- A variable becomes static when its declaration includes the **static** keyword
- Initialize static variables as they're declared, otherwise is null or 0.
- Static variables can appear in constructors, methods, or outside their class

```
class RedBumper {
    // Fields
    public static int orientation= 45; // Static variable
    public int rotation; // Instance variable

    // Constructor
    public RedBumper(int rotation) {
        this.rotation = rotation;
    }

    // Methods
    public void display() {
        System.out.println("Orientacion: " + orientation); // Access static var
        System.out.println("Rotacion: " + rotation); // Access instance var
    }
}

public class TestRedBumper {
    public static void main(String[] args) {
        int x = RedBumper.orientation; // Access static variable
        RedBumper rb01 = new RedBumper(90); // Instance
        int y = rb01.rotation; // Access instance variable
    }
}
```

→

Juan Carlos Herrera H.

Introducing Static Methods

Static variables are accessible from nonstatic methods.
Methods can also be made static.
Static variables are accessible from static methods.
Static or instance methods may call a static method
Java doesn't allow static methods to contain instance variables or instance methods

`public static int getPrisonerCount() {`
`  return prisonerCount;`
`}`
	`int total = Prisoner.getPrisonerCount()`

The names of final variables …

- Are capitalized by convention
- Use an underscore (_) to separate words

```
private static int prisonerCount = 0;
public static final int MAX_PRISONER_COUNT = 100;
System.out.println(Math.PI);
```

Static methods can be invoked through an instance of a class,
pero no es una buena practica.

In Java, **setters are typically void return type methods.**
An object can access another object's public methods
An object can access another object's public fields.
Un metodo estatico no puede llamar una variable de Instancia

Quiz 2: JFo - Section 7 - L4-L6

→

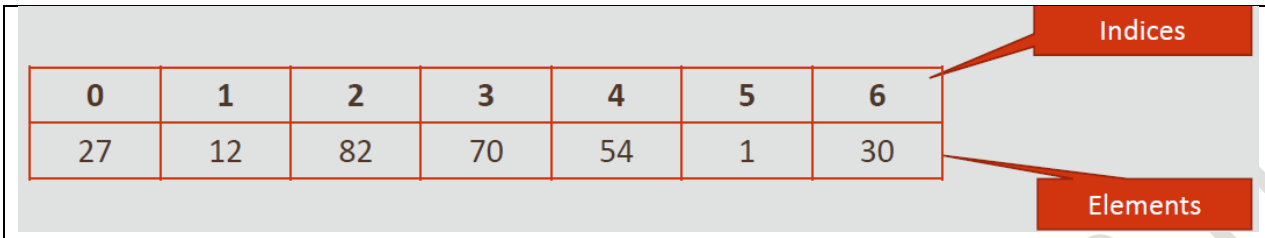
Juan Carlos Herrera H.

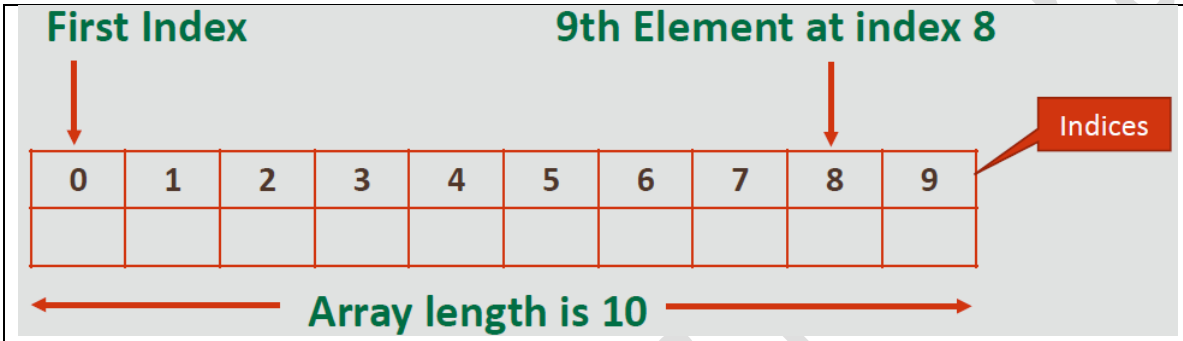
8. Arrays and Exceptions

8.1. One-dimensional Arrays

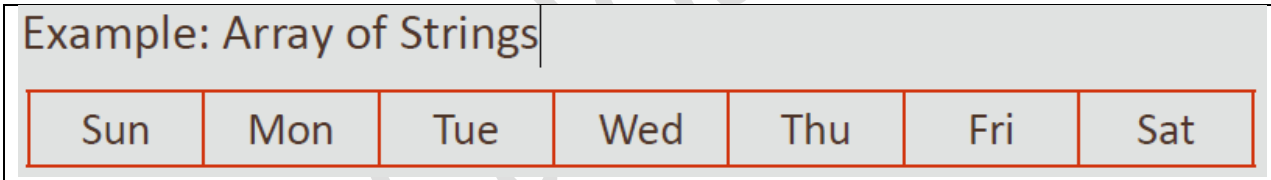
In Java, an array is an indexed container that holds a set of values of a single type

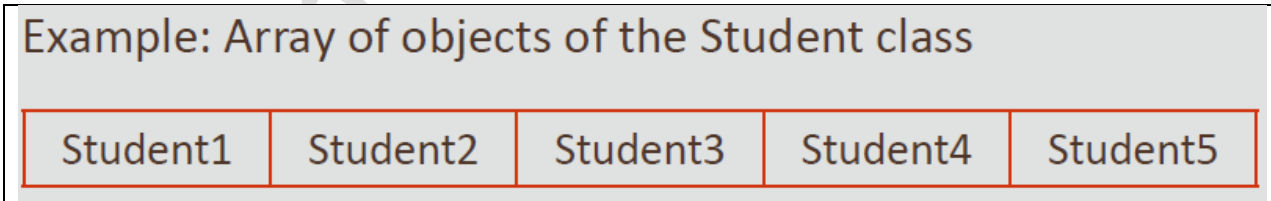
Arrays can be of any data type, but all elements have to share the same type





A 10-element array has 0 to 9 indices






```
1. int[] prime;
2. int prime[];
```

You can declare an array in two ways

```
int[] myIntArray = new int[100];

int[] myIntArray;
myIntArray = new int[100];
En este momento casa casilla tiene un 0
```

- Use the following syntax:

```
data_type[] variable_name = new data_type[size];
variable_name[index] = value; //repeat for each element
```

Arrays can't grow or shrink after initialization

Valores Desconocidos

```
int[] ages = new int[3];
ages[0] = 19;
ages[1] = 42;
ages[2] = 92;
```

```
String[] names = new String[3];
names[0] =  "Mary";
names[1] =  "Bob";
names[2] =  "Carlos";
```

Variable Name | Index | Value

Valores Conocidos

```
int[] ages = {25, 27, 48};
String[] names = {"Mary","Bob","Carlos"};
names[1] = "Patricio";  // cambiar el valor
System.out.println("My name is " + names[0]);
```

0	1	2
Mary	**Patricio**	Carlos
names[0]	names[1]	names[2]

You can access the size of any array

int count = names.length // 3

How Do You Print the Values of a names Array?

```
for (int idx = 0; idx < names.length; idx++){
    System.out.println(names[idx]);
}//end for
```

boolean expression

Counter used as the index of the array

Using a **for-each**

Iteration-Variable

Type | Array Name

```
for(String name: names){
    System.out.println(name);
}//end for
```

If an array index is out of bounds, the JVM throws an ArrayIndexOutOfBoundsException
The program is terminated if this exception isn't handled

1. Obtener el valor mínimo y el promedio del siguiente arreglo

```
int[] array = { -20, 19, 1, 5, -1, 27, 19, 5 } ;
```

2. Quita los elementos negativos del arreglo.
   Vuelve a obtener el mínimo y el promedio del arreglo.

→

8.2. ArrayLists

Objectives: Manipulate an ArrayList by using its methods.
Use wrapper classes and Autoboxing to add primitive data types to an ArrayList

ArrayList supports dynamic arrays that can grow as needed. ArrayList are known like Collection.
Has a set of useful methods for managing its elements: add(), get(), remove(), indexOf(), and many others
An ArrayList can contain only objects, not primitives. Examples: String, Person, Prisoner, Car, so on.

Some ArrayList Methods

add(value)	Appends the value to the end of the list
add(index, value)	Inserts the given value just before the given index, shifting subsequent values to the right
clear()	Removes all elements of the list
indexOf(value)	Returns the first index where the given value is found in the list (-1 if not found)
get(index)	Returns the value at the given index
remove(index)	Removes the value at the given index, shifting subsequent values to the left
set(index, value)	Replaces the value at the given index with a given value
size()	Returns the number of elements in the list
toString()	Returns a string representation of the list, such as "[3, 42, -7, 15]"

You can traverse an ArrayList in the following ways:

- Using the for-each loop
- Using an Iterator (forward): hasNext(),next(),remove()
- Using a ListIterator (both directions): hasNext(), hasPrevious(), next(), previous(). Remove() don't exist.

```
for (String i: names) {
  System.out.println("Name is "+i);
}
```

```
Iterator<String> iterator = names.iterator();
while (iterator.hasNext()) {
  System.out.println("Name is "+iterator.next());
}
```

```
ListIterator<String> litr = names.listIterator();

System.out.println("Traversing list forwards: ");
while (litr.hasNext()) {
  System.out.println("Name is "+ litr.next());
}

System.out.println("Traversing list backwards:");
while (litr.hasPrevious()) {
  System.out.println("Name is " + litr.previous());
}
```

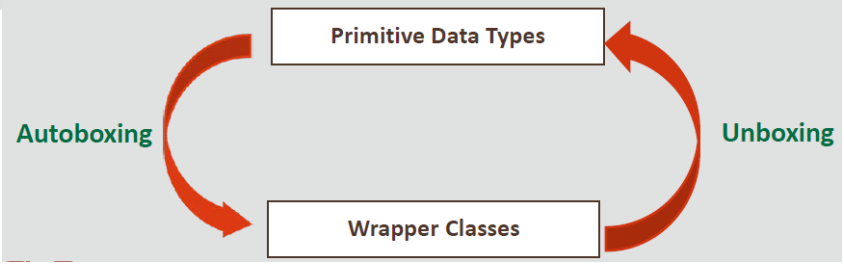
→

Juan Carlos Herrera H.

```
import java.util.ArrayList;

class Persona {
    String nombre;
    int edad;

    Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}

public class Colecciones {

    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<>();
        ArrayList<Persona> personas = new ArrayList<>();
        names.add("Jesus");
        names.add("Maria");
        names.add("Jose");
        names.add("Melchor");
        String nombre = names.get(3);
        System.out.println("El nombre en la posicion 3 es: " + nombre);
        names.remove(3);

        System.out.println("Contenido de la lista:");
        for (String elemento : names) {
            System.out.println(elemento);
        }
        personas.add(new Persona("Chuy", 30));
    }
}
```

You can still use ArrayList with primitive types by using special classes called wrapper classes
ArrayList<**Integer**> list = new ArrayList<>();

Primitive Type	Wrapper Type
byte	Byte
Short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean




```
Double score= 18.58; // Autoboxing
double goal = score; // Unboxing
```

Example: Remueva los números negativos: `int[] array = { -20, 19, 1, 5, -1, 27, 19, 5 } ;`
→

Juan Carlos Herrera H.

8.3. Exception Handling

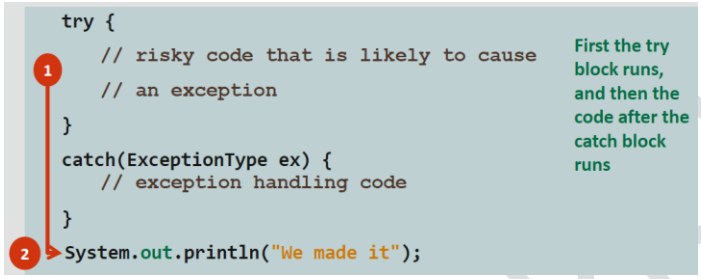
An exception is an error that occurs during the execution of a program(run-time) that disrupts the normal flow of the Java program.

Execution of the program is terminated. Java "throws" an exception. The program crashes.
A stack trace, with the details of the exception, is printed in the console
Example: int a = 10 / 0;

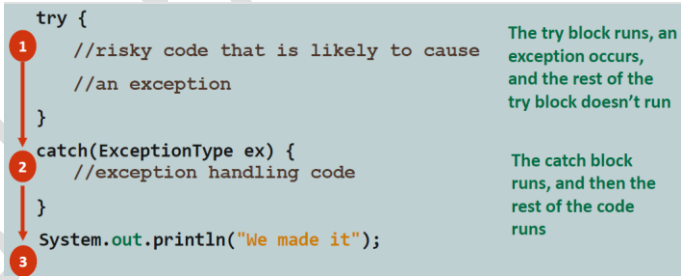
```
int divisor = 0;
if (divisor == 0) {
    System.out.println("Can't be zero!");
} else {
    System.out.println( 5 / divisor);
}
System.out.println(". . .Hecho!");
```

```
int divisor = 0;
try {
    System.out.println( 5 / divisor);
} catch (Exception e) {
    System.out.println(e.getMessage());
}
System.out.println(". . .Hecho!");
```

If the try block succeeds, no exception occurs



If the try block fails, an exception occurs



Java exceptions fall into two categories:

- Checked Exceptions:
  - Compiler checks and deals with exceptions
  - If the exceptions aren't handled in the program, it gives a compilation error
  - Examples: `FileNotFoundException`, `IOException`
- Unchecked Exceptions:
  - Compiler does not check and deal with exceptions
  - Examples: `ArrayIndexOutOfBoundsException`, `NullPointerException`, `ArithmeticException`

Juan Carlos Herrera H.

Examples of Exceptions:

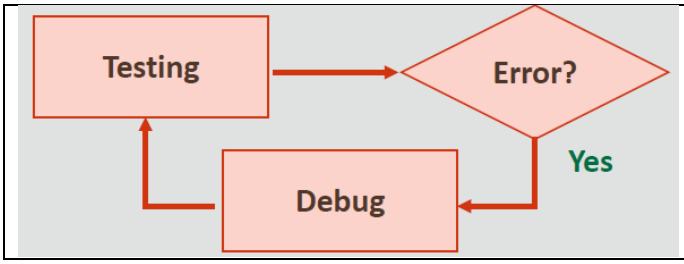
- java.lang.ArrayIndexOutOfBoundsException
  - Attempt to access a nonexistent array index
- java.lang.NullPointerException
  - Attempt to use an object reference that wasn't instantiated
- java.io.IOException
  - Failed or interrupted I/O operations

```
int[] intArray = new int[5];
intArray[5] = 27;

String name=null;
System.out.print("Longitud: " + name.length());


try { // Checked
    File testFile = new File("//testFile.txt");
    testFile.createNewFile();
    System.out.println("testFile exists: " + testFile.exists());
} catch (IOException e) { // Tratar de ser especifico en la excepcion
    System.out.println(e);
}

→
```

Juan Carlos Herrera H.

8.4. Debugging Concepts and Techniques

Three Types of Errors:	Debugging Techniques
Compilation errors: Syntax error • Logic errors: Produces incorrect result • Runtime errors: Exception handling to detect errors ArrayIndexOutOfBounds, NullPointerException, ArithmeticException and so on	• Using print statements • Using an IDE debugger



Testing:
- To determine if a code contains errors

Debugging:
- To identify an error and fix it

Print statements:
- Print the value of a variable.
- Print within an if statement to see if it was true or false.
- Print to see if it went through a certain process.

The IDE Debugger
- Set breakpoints
- To view the contents of variables
- Set a breakpoint when the variable reaches a certain value. Expresion: n ==10

Find the bug

```
    public static int sumatoria(int n) {
        int sum = 10;
        while (n > 1) {
            sum = sum + n;
            n--;
        }
        return sum;
    }

    public static void main(String[] args) {
        System.out.println("The sum of the integers 1 to 10 is " + sumatoria(10));
    }
```

Problema:
Basado en coordenadas, un carro se encuentra en el punto(2,1) y desea llegar al punto(6,4). ¿Cuál es la distancia que debe recorrer para llegar a su destino?
Caso 1: Considere que se encuentra en el desierto o en una zona llana.
Sabemos que la distancia más corta entre dos puntos es la línea recta.
Caso 2: Considere que se encuentra en una ciudad urbana, donde todas las calles están cuadriculadas.
Sabemos que todas las calles o cuadras tienen un ángulo de 90 grados.

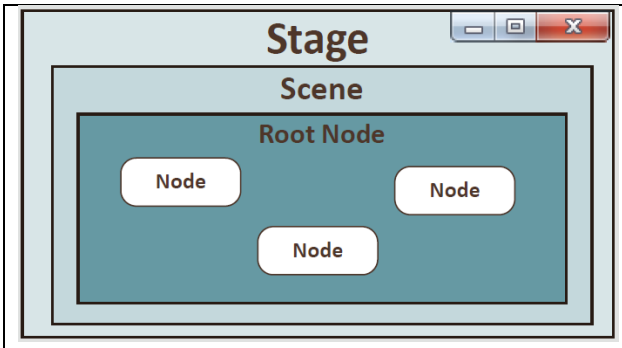
Quiz: JFo - Section 8

→

Juan Carlos Herrera H.

9. JavaFX

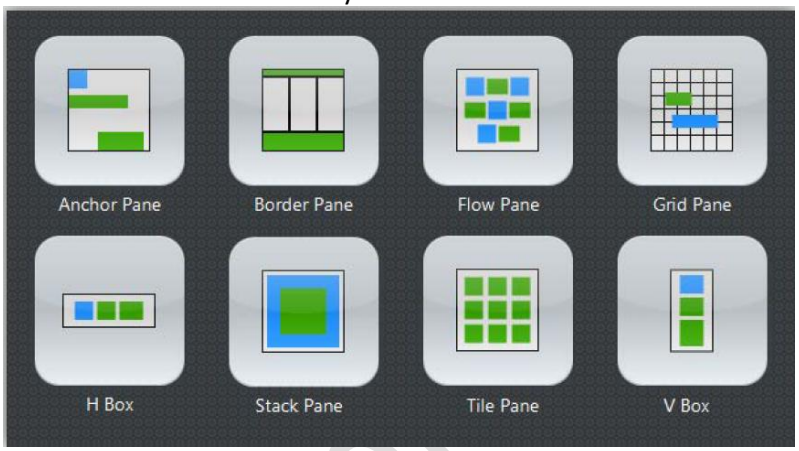
9.1. Introduction to Java FX



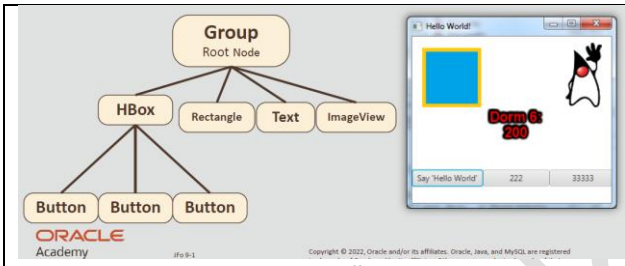
root puede ser:
StackPane root = new StackPane();
HBox root = new HBox();
TilePane root = new TilePane();

root.getChildren().addAll(btn1, btn2);

Each Pane determines the layout of Nodes



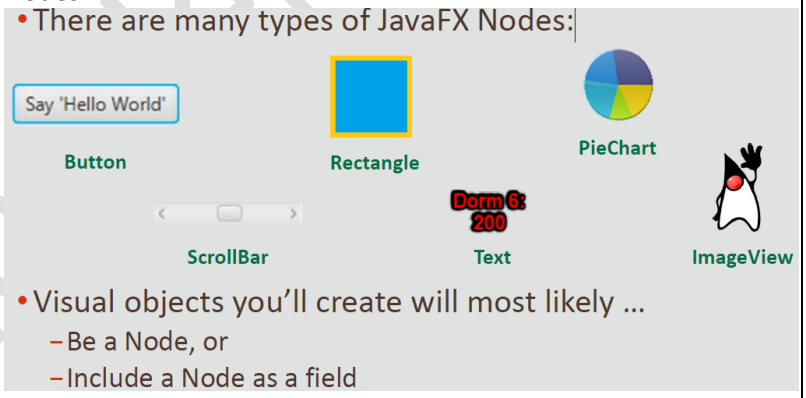
Panes are also Nodes



HBox rootHBox = new HBox();
rootHBox.getChildren().add(btn);

A Group allows you to place Nodes anywhere
Group root = new Group(); //
root.getChildren().addAll(rootHBox, btn2);

Nodes



Any node can be added to the Root Node

Buttons for example, are like any other object
They can be instantiated
They contain fields
They contain methods

Button btn = new Button();

Buttons for example, are Nodes:
btn.setText()
btn.getText()
btn.setLayoutX() //set x position
btn.setLayoutY() //set y position
btn.isPressed() //is it pressed?

→

Juan Carlos Torres

A Root Node (which contains: Buttons, Text, and so on)

A Scene (which contains the Root Node). The Scene is the container for all content in the JavaFX Scene Graph

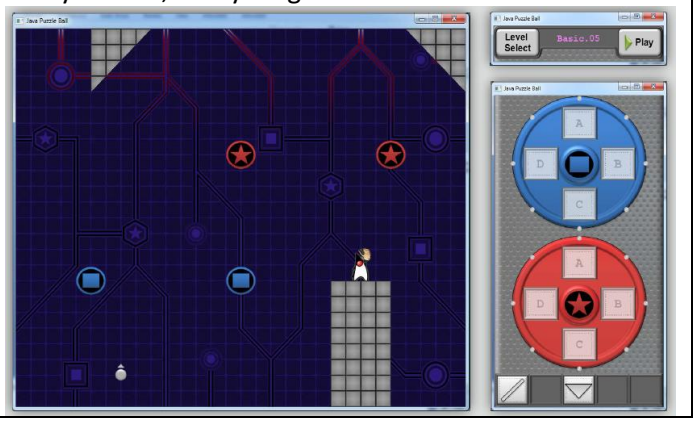
A Stage (which contains the Scene). Think of the Stage as the application window

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class JavaFXMainEx extends Application {
    @Override
    public void start(Stage primaryStage) {  // main method for JavaFX
        Button btn = new Button();
        btn.setText("Say 'Hello World'");

        btn.setOnAction(event -> System.out.println("Hello World"));//lambda expression

        // De adentro hacia afuera:      Root   ->    Scene  ->    Stage

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 250, Color.BLACK);//root Node, width, height

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {  // It launches the JavaFX application
        launch(args);
    }
}
```

Many Scenes, One Stage

It's possible to swap any scene into a single Stage

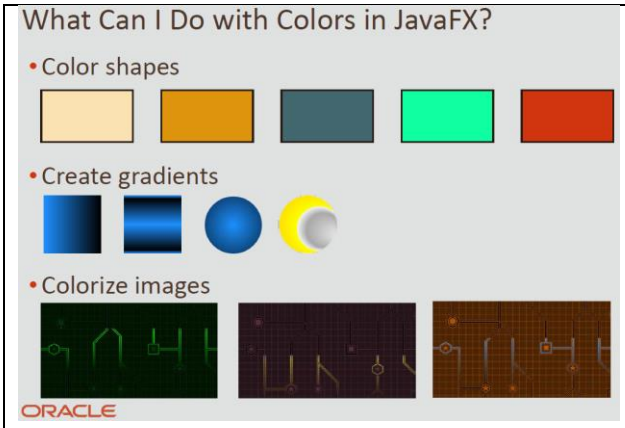
Many Scenes, Many Stages

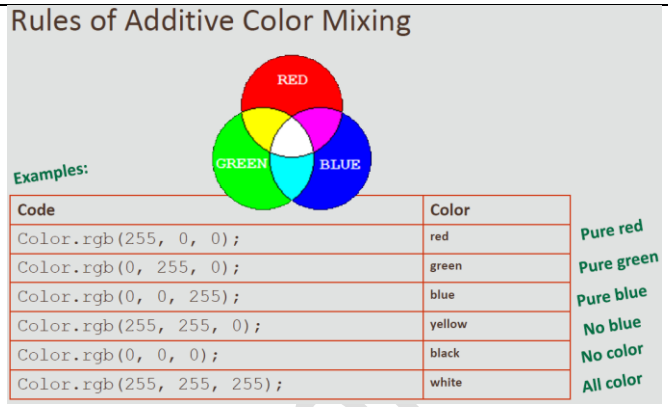


→

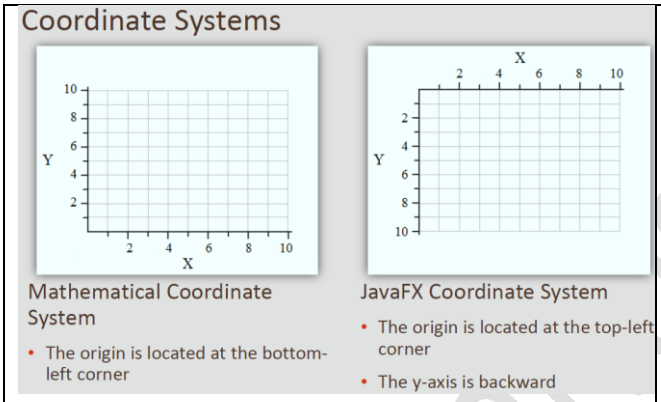
9.2. Colors and Shapes

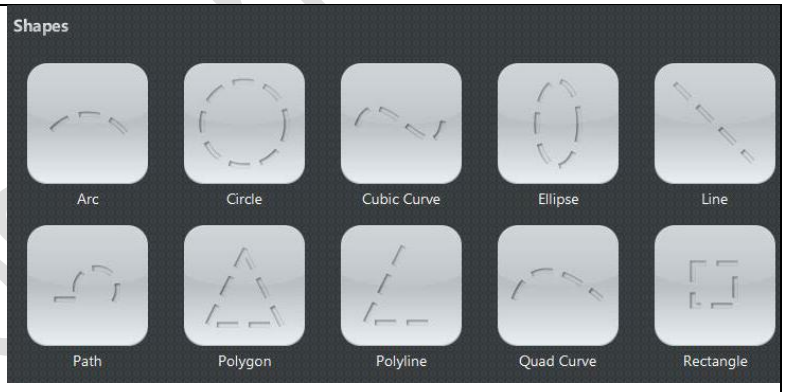
https://openjfx.io/ Download Software
https://www.jfx-ensemble.com/ Examples
https://openjfx.io/javadoc/17/index.html JavaFX API Documentation





Color color = Color.rgb(255, 255, 20, 1.0); // Opacity [0-1]






```
Import javafx.scene.shape.Rectangle;

Rectangle rect = new Rectangle(20, 20, 100, 200); // x, y, width, height
rect.setX(10);
rect.setFill(Color.AQUAMARINE);

Circle circle = new Circle(50,100, 200); // x, y, radius
circle.setFill(gradient1);
```

```
setX(double d)
setY(double d)
setWidth(double d)
setHeight(double d)
setFill(Paint paint)
setStroke(Paint paint)
setStrokeWidth(double d)
```

```
// create simple linear gradient
LinearGradient gradient1 = new LinearGradient(0, 0, 1, 0, true,
      CycleMethod.NO_CYCLE, new Stop[] {
            new Stop(0, Color.DODGERBLUE),
            new Stop(1, Color.BLACK)
      });
```

→

9.3. Graphics, Audio and MouseEvents

Understand Lambda expressions in GUI applications

An Image is an object that describes the location of a graphics file (.png, .jpg, .gif …)
An ImageView is the actual Node

Image image; String imagePath = "Images/Fan1.png"; Image = new Image(getClass().getResource(imagePath).toString); ImageView imageView = new ImageView(image); imageView.setImage(image2);	 Fan1.png image1	 Fan2.png image2

You have the option to preserve the aspect ratio of an ImageView
An ImageView's width and height scale together
This avoids distortion
imageView.setPreserveRatio(true);
imageView.setFitWidth(25);
	
	It might look awful!

Ordering Nodes the Right Way

• The order that Nodes are added to the Root Node determines the order that they are displayed
• Nodes added early are buried under nodes added later
root.getChildren().addAll(gloveImageView, dukeImageView);
 	
• To fix this you could …
Change the order that Nodes are added to the Root Node
Bring an ImageView to the front or back
gloveImageView.toFront(); //Either one of these
dukeImageView.toBack(); //will solve the problem
		

```
AudioClip audio = new AudioClip(getClass().getResource("../vis/Note5.wav").toString());
audio.play();
```

Realizar el siguiente ejercicio

Juan Carlos Herrera H.

Mouse and Keyboard Events	**Event Listening**

Nodes can detect mouse and keyboard events
Helpful methods to make this happen include:
setOnMouseClicked()
setOnMouseEntered()
setOnMouseExited()
setOnMouseMoved()
 	
setOnMousePressed()
setOnMouseDragged()
setOnMouseReleased()
	btn.setOnAction(new EventHandler<ActionEvent>() {
	  @Override
	  public void handle(ActionEvent event) {
	  System.out.println("Hello World!");
	  }
	});
	
	Lambda Expressions
	btn.setOnAction (/*Lambda Expression*/);
	btn.setOnAction(event -> System.out.println("Hello World!"));

```
imageView.setOnMousePressed(event -> System.out.println(("Clicked at: " + event.getX() + ", " + event.getY() ) );

imageView.setOnMouseClicked(event -> {
   audio.play();
   // formatear a dos decimales
   lbl2.setText(                    // getX, getSceneX, getScreenX
         "Clicked at: " + String.format("%.2f", event.getX()) + ", " + String.format("%.2f", event.getY()) );
});
```

Quiz: JFo - Section 9

Juan Carlos Herrera H.