JAVA FOUNDATIONS 1Z0-811

ORACLE ACADEMY

2 DE SEPTIEMBRE DE 2025

https://academy.oracle.com/

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

Contenido

→

1. Introduction

1.1. Technological Requirements:

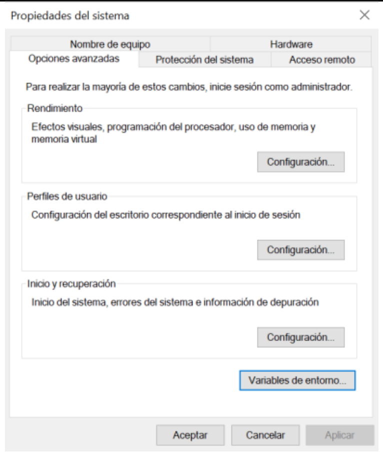
Java JDK	https://www.oracle.com/java/technologies/downloads/	jdk-8u202-windows-x64.exe
VS Code	https://code.visualstudio.com/Download	VSCodeSetup-x64-1.103.2.exe

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


Propiedades del sistema

Para realizar la mayoría de estos cambios, inicie sesión como administrador.

Rendimiento

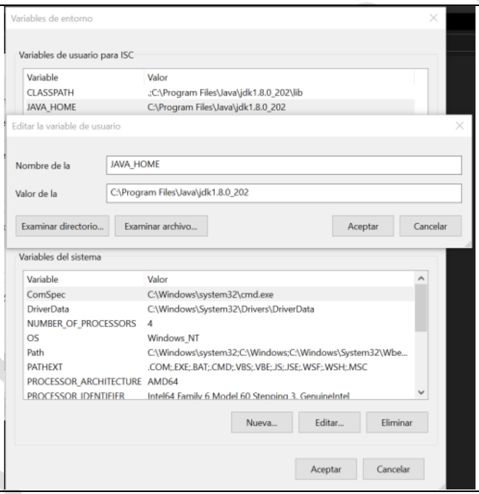
Efectos visuales, programación del procesador, uso de memoria y memoria virtual

Perfiles de usuario

Configuración del escritorio correspondiente al inicio de sesión

Inicio y recuperación

Inicio del sistema, errores del sistema e información de depuración


Variables de entorno

Variables de usuario para ISC

Variable	Valor
CLASSPATH	.;C:\Program Files\Java\jdk1.8.0_202\lib
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_202

Editar la variable de usuario

Nombre de la: JAVA_HOME

Valor de la: C:\Program Files\Java\jdk1.8.0_202

Variables del sistema

Variable	Valor
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbe...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 60 Stepping 3, GenuineIntel

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!
```

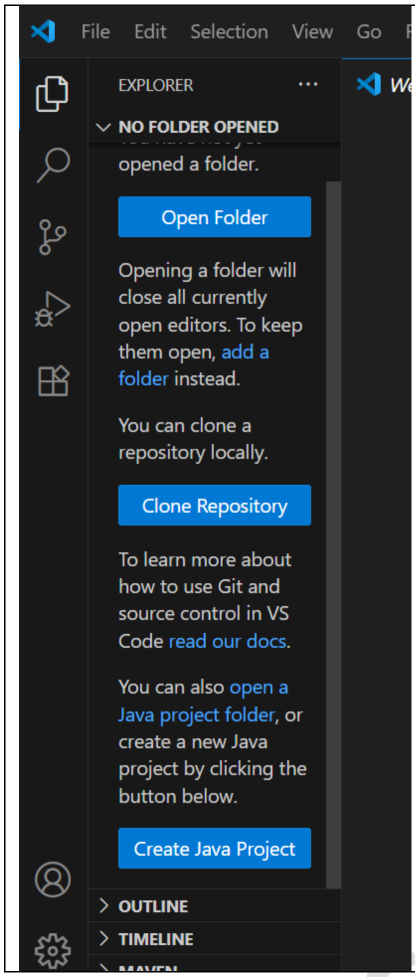
```java
public class Hola {

    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:



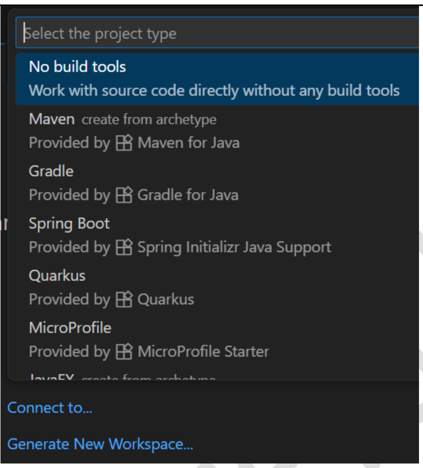
opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can clone a repository locally.

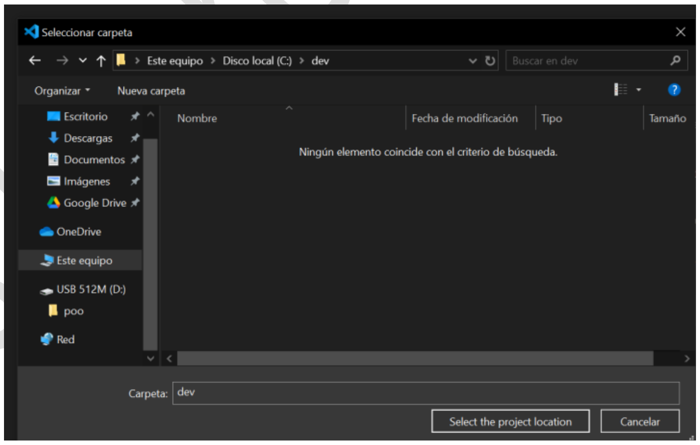
To learn more about how to use Git and source control in VS Code read our docs.

You can also open a Java project folder, or create a new Java project by clicking the button below.



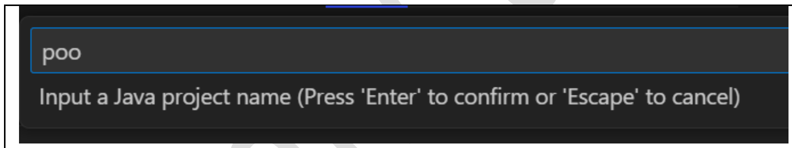
Select the project type

- No build tools — Work with source code directly without any build tools
- Maven create from archetype — Provided by Maven for Java
- Gradle — Provided by Gradle for Java
- Spring Boot — Provided by Spring Initializr Java Support
- Quarkus — Provided by Quarkus
- MicroProfile — Provided by MicroProfile Starter



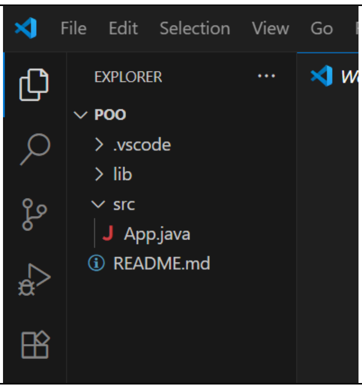
Ningún elemento coincide con el criterio de búsqueda.

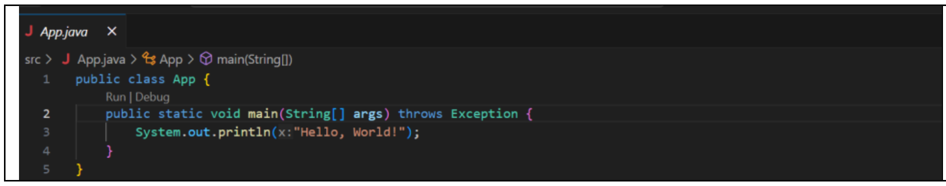
Carpeta: dev



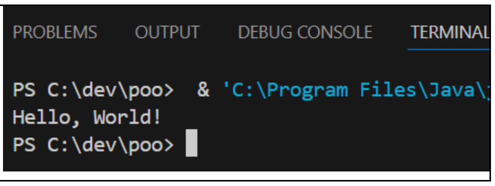
poo

Input a Java project name (Press 'Enter' to confirm or 'Escape' to cancel)





```java
public class App {
    public static void main(String[] args) throws Exception {
        System.out.println(x:"Hello, World!");
    }
}
```



```
PS C:\dev\poo>  & 'C:\Program Files\Java\
Hello, World!
PS C:\dev\poo>
```

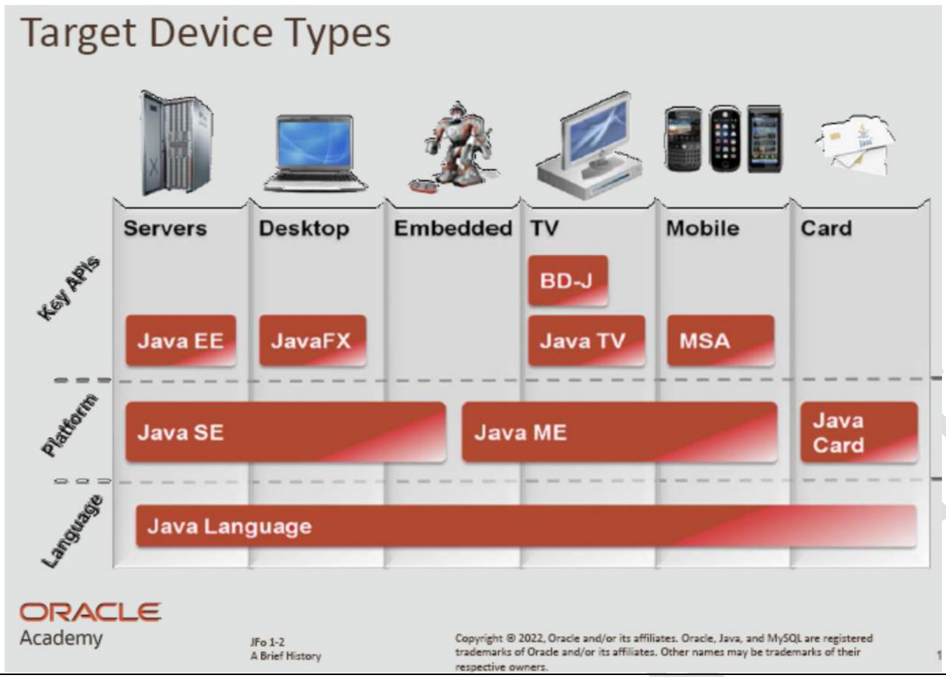
→

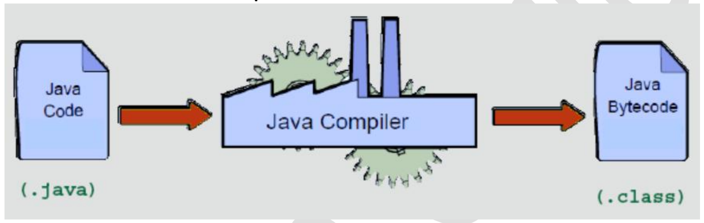
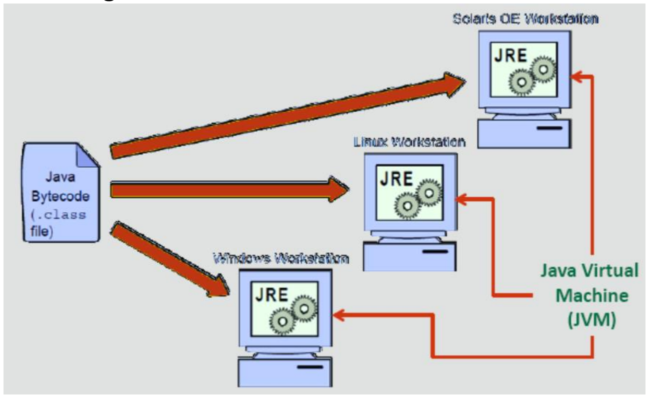
1.3. Setting Up Java

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

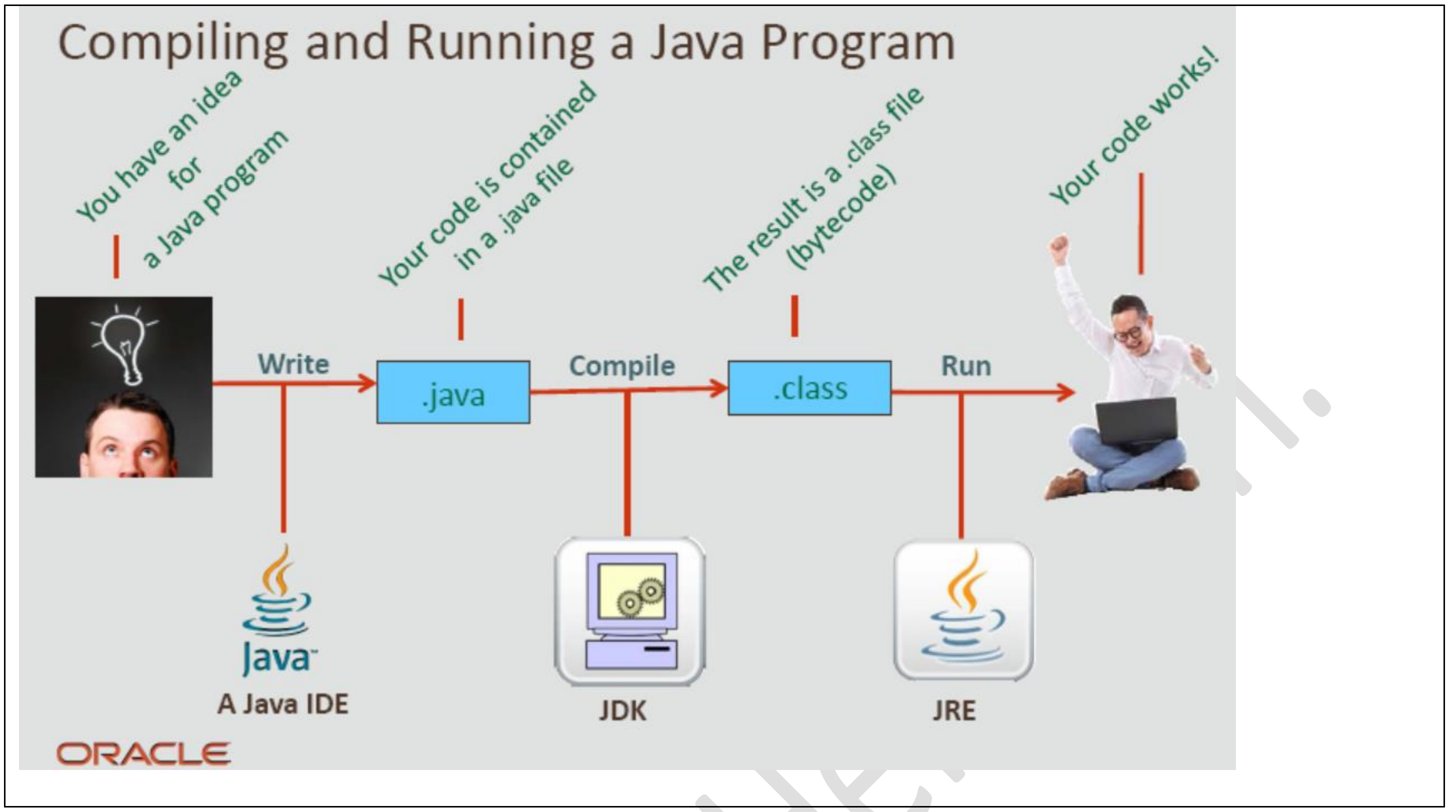
Oracle acquired Sun Microsystems in 2010, and released JDK 7 in 2011, and JDK 8 in 2014.

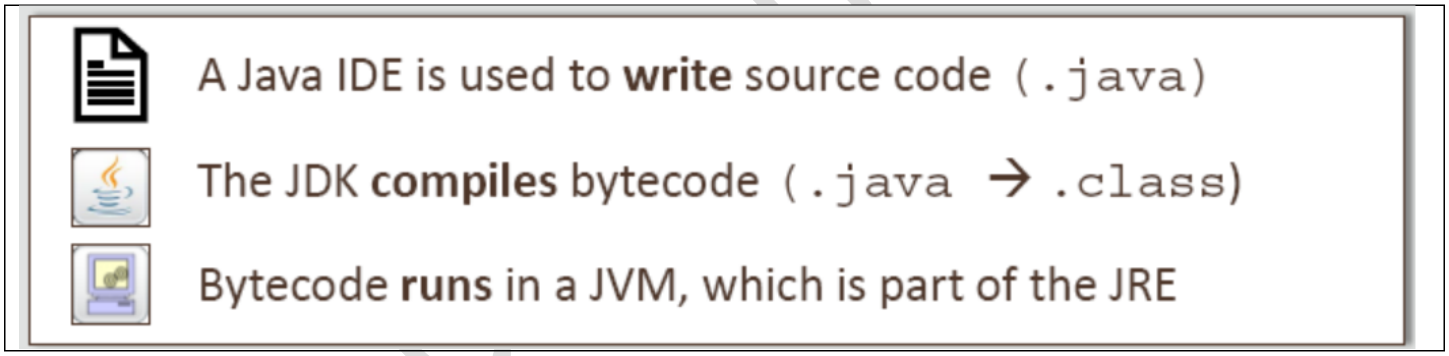
Jakarta EE Is used to create large enterprise, server-side, and client-side distributed applications


Target Device Types
Servers
Desktop
Embedded
TV
Mobile
Card
Key APIs
Java EE
JavaFX
BD-J
Java TV
MSA
Platform
Java SE
Java ME
Java Card
Language
Java Language
ORACLE Academy
JFo 1-2 A Brief History
Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Java is Platform-Independent	Java Programs Run in a JVM
 Java Code (.java) Java Compiler Java Bytecode (.class) •	 Solaris OE Workstation JRE Linux Workstation JRE Windows Workstation JRE Java Bytecode (.class file) Java Virtual Machine (JVM)

Java Runtime Environment (JRE)	Java Development Kit (JDK)
Includes: • The Java Virtual Machine (JVM) • Java class libraries Purpose: • Read bytecode (.class) • Run the same bytecode anywhere with a JVM	Includes: • JRE Java Compiler • Additional tools Purpose: Compile bytecode (.java .class)


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)`

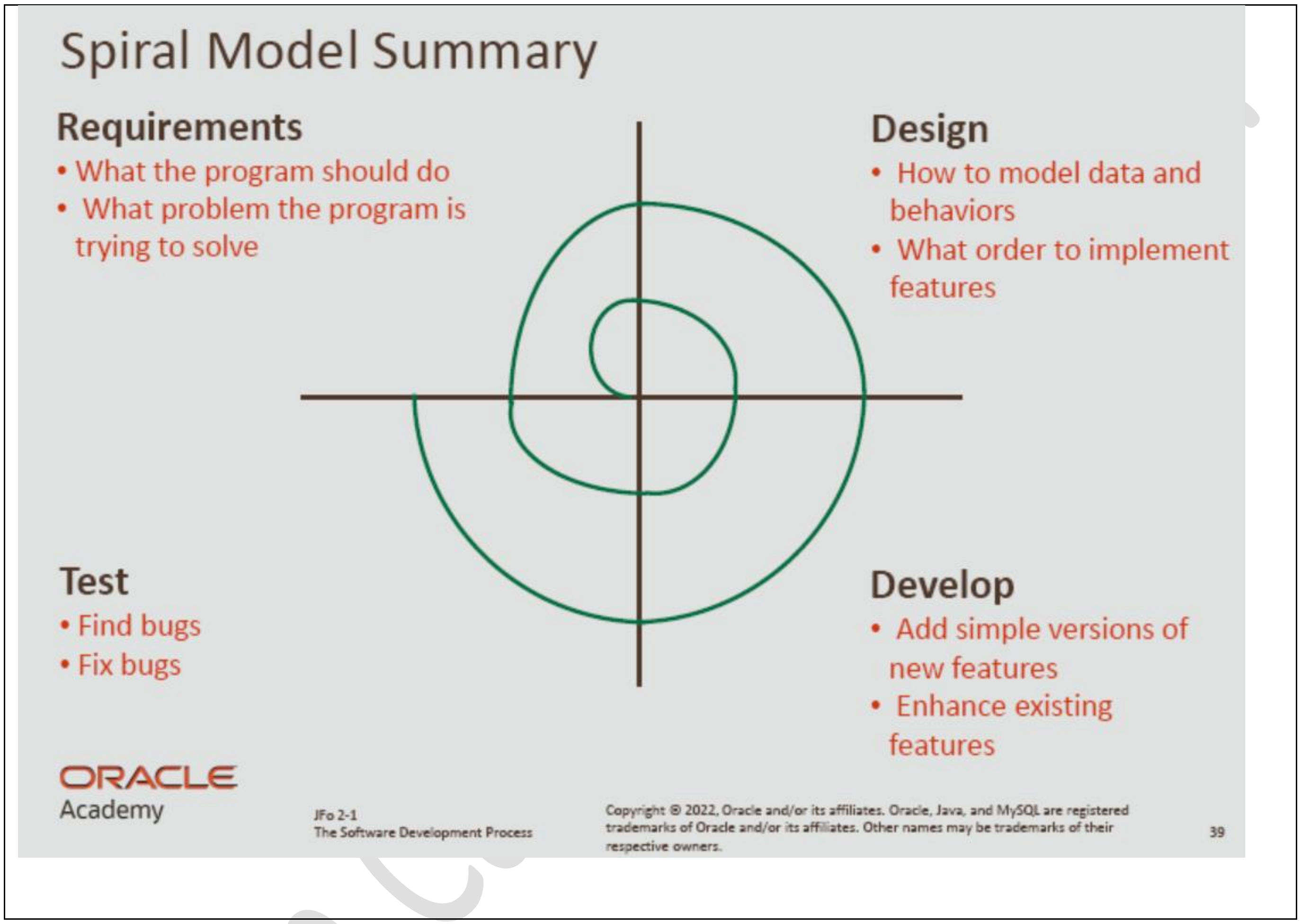
Bytecode **runs** in a JVM, which is part of the JRE

→

2. Java Basics

2.1. The Software Development Process

Spiral Model of Development



Spiral Model Summary

Requirements
- What the program should do
- What problem the program is trying to solve

Design
- How to model data and behaviors
- What order to implement features

Test
- Find bugs
- Fix bugs

Develop
- Add simple versions of new features
- Enhance existing features

ORACLE Academy

JFo 2-1
The Software Development Process

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

39

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

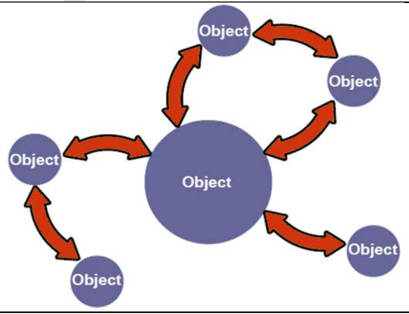
→

2.2. What is my Program Doing?

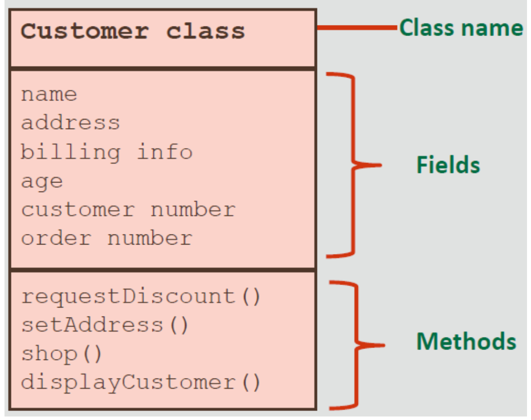
Code within curly braces is called a block of code Indentation before a line of code (4 spaces) Whitespace End statements with semicolons (;) // Single-line comments Multi-line comments /* Bievenidos a poo */	public static void NombreMetodo() { . . } NombreMetodo(); // llamar al método Debug To set a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

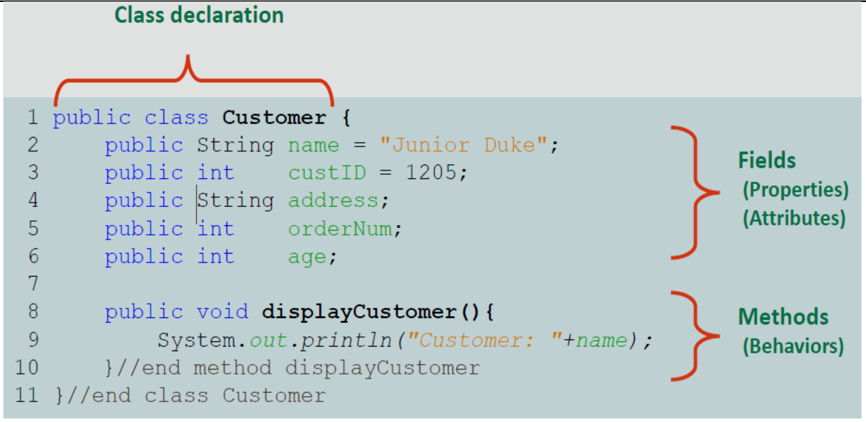
2.3. Introduction to Object-Oriented Programming Concepts

Procedural languages ...	Object-oriented languages…
• Read one line at a time • The C language is procedural	• 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	 Object Object Object Object Object Object Object

Modeling Properties and Behaviors





```
public class Customer {
    public String name = "Junior Duke";
    public int    custID = 1205;
    public String address;
    public int    orderNum;
    public int    age;

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

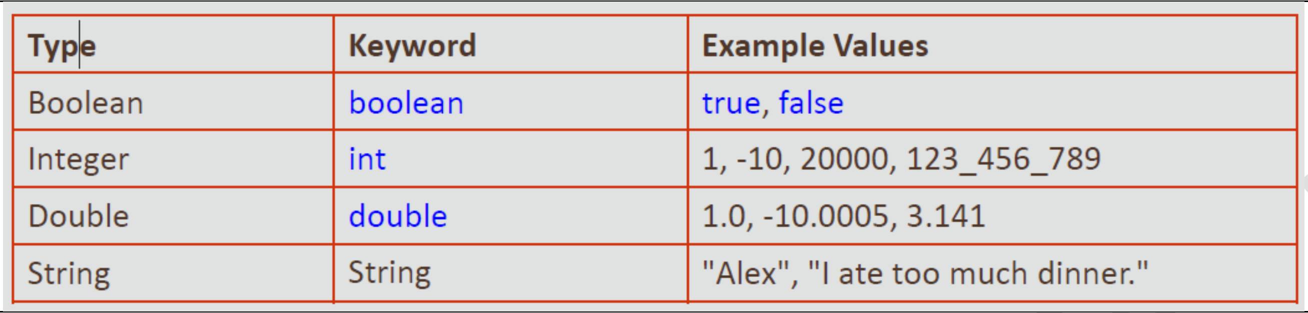
Quiz: JFo - Section 2 Questions 15

→

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	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."

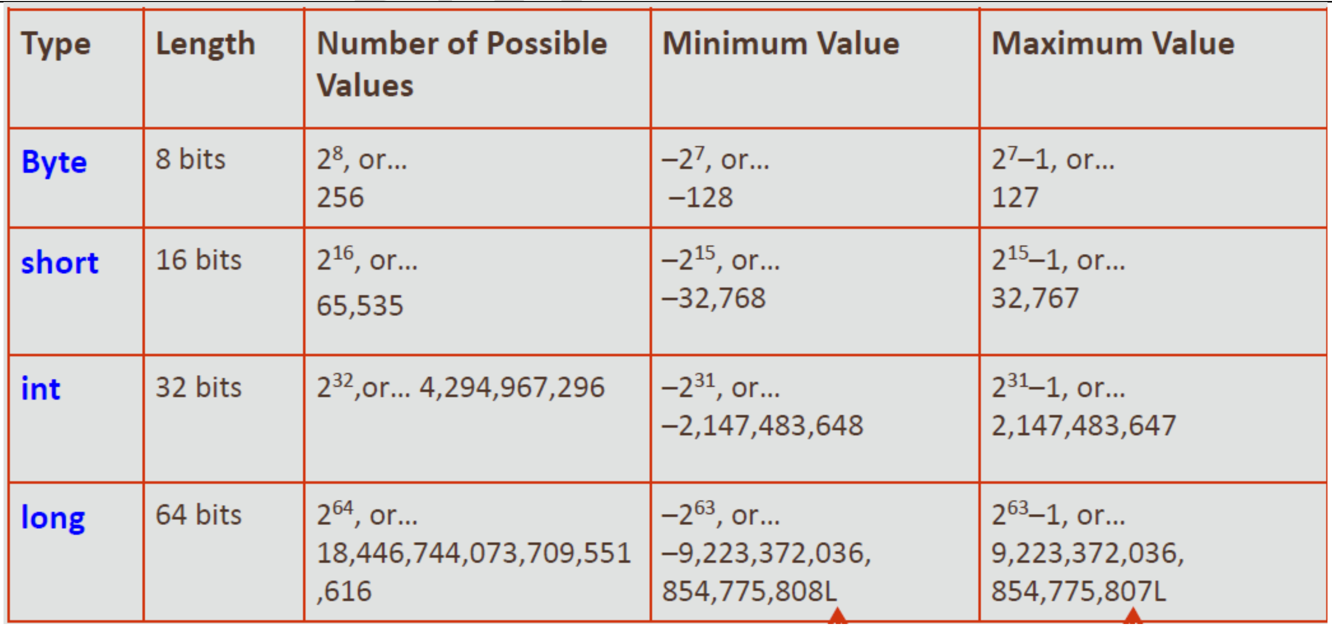
Variable Naming Conventions

- 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



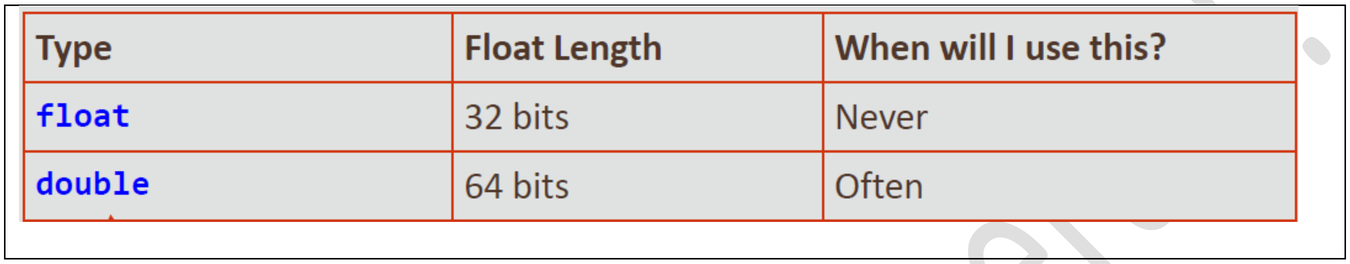
Type	Length	Number of Possible Values	Minimum Value	Maximum Value
Byte	8 bits	2^{8}, or... 256	-2^{7}, or... –128	$2^{7}-1$, or... 127
short	16 bits	2^{16}, or... 65,535	-2^{15}, or... –32,768	$2^{15}-1$, or... 32,767
int	32 bits	2^{32},or... 4,294,967,296	-2^{31}, or... –2,147,483,648	$2^{31}-1$, or... 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)

// 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;
```

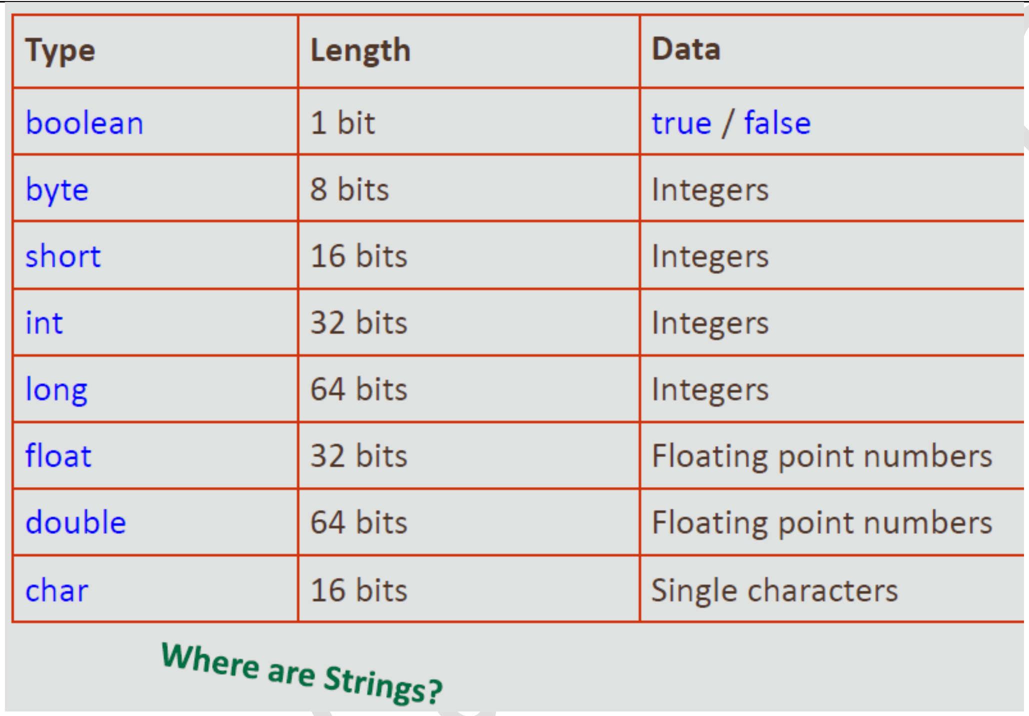
→

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

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
```

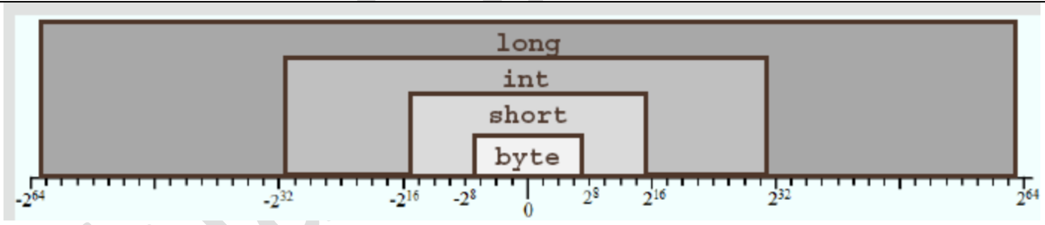
```
System.out.println("This is the first line."
                + "This is NOT the second line.");
```

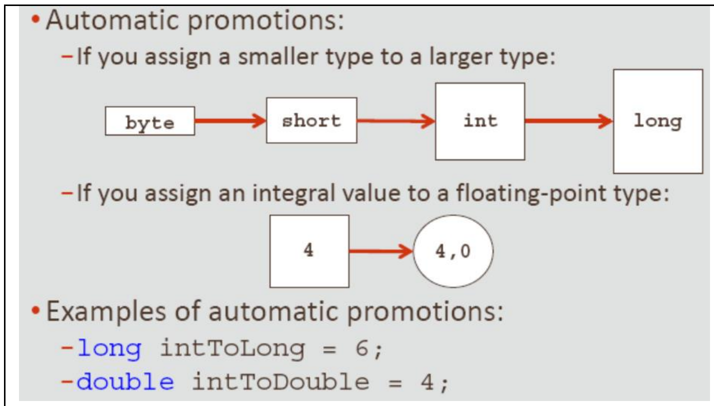
sout tab “Metodo abreviado”

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
```

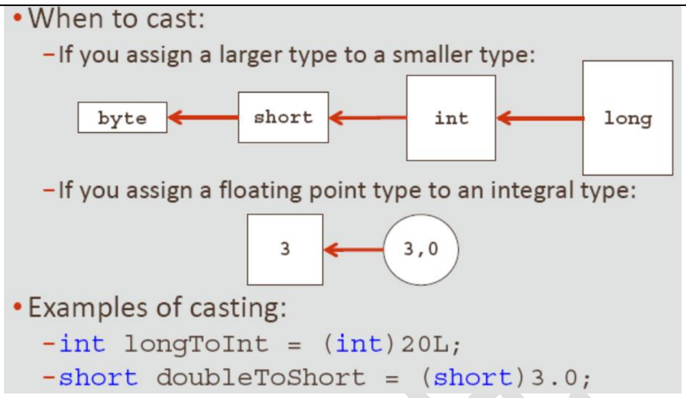




- Automatic promotions:
  - If you assign a smaller type to a larger type:

  - 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:

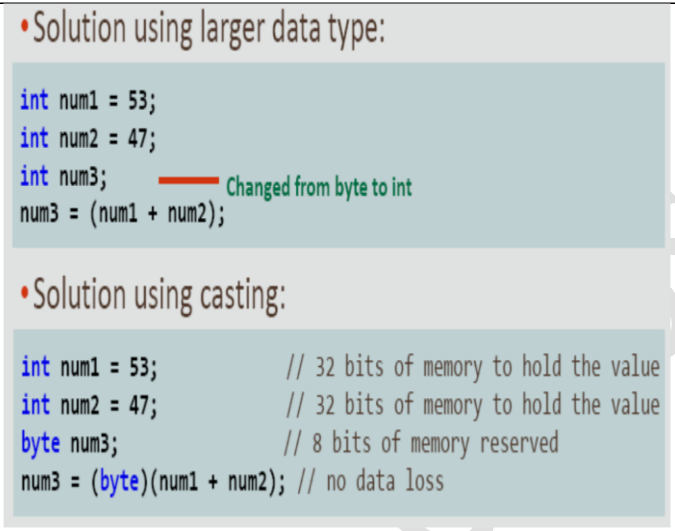
  - If you assign a floating point type to an integral type:

- 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

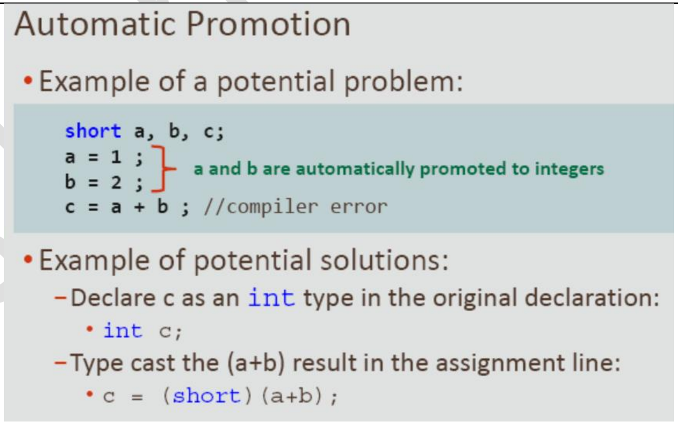


- Solution using larger data type:

```
int num1 = 53;
int num2 = 47;
int num3;          Changed from byte to int
num3 = (num1 + num2);
```

- 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 ;  } a and b are automatically promoted to integers
b = 2 ;  }
c = a + b ; //compiler error
```

- 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);`

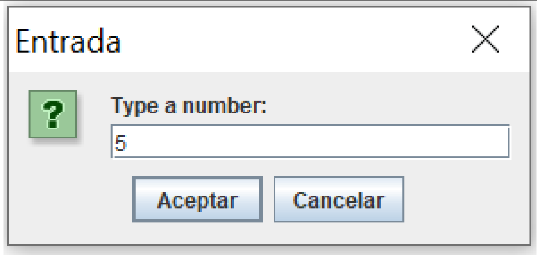
int x = 123_456_789;
int x = 123456789;

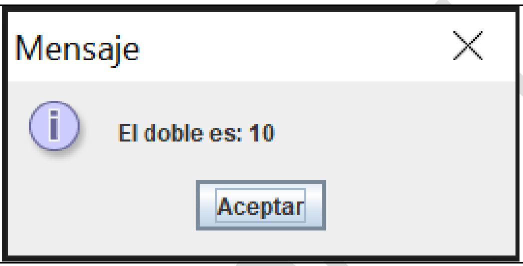
intintVar1 = Integer.parseInt("100");
doubledoubleVar2 = Double.parseDouble("2.72");

→

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);
```





El doble es: 10

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

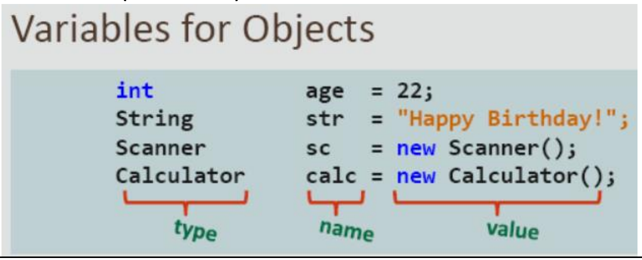
→

4. Java Methods and Library Classes

4.1. What Is a Method?

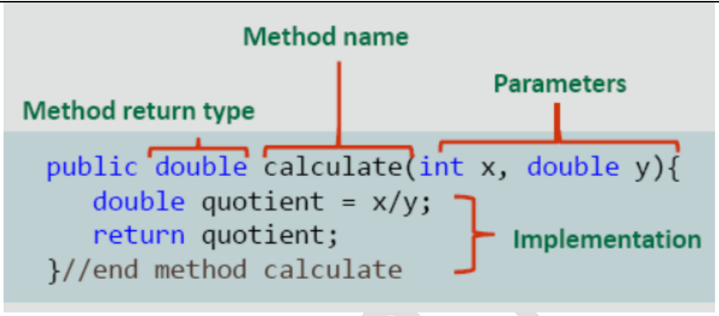
Instantiate an object

These classes outline objcts' …
Properties(fields)
Behaviors(methods)

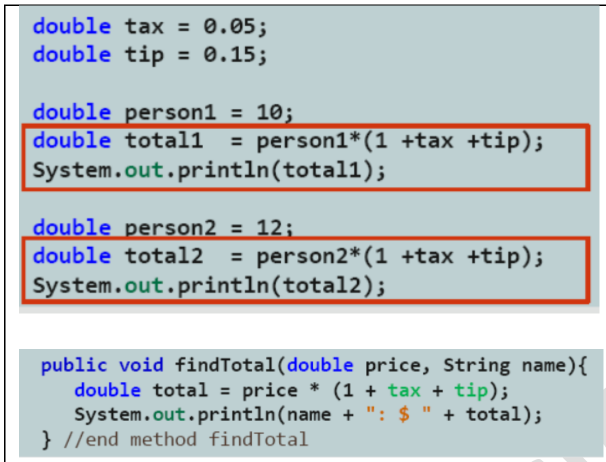


Variables for Objects

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

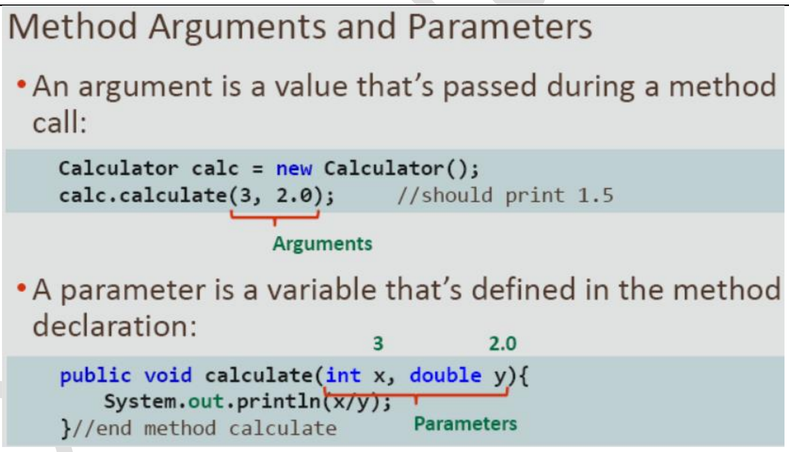


```
public double calculate(int x, double y){
    double quotient = x/y;
    return quotient;
}//end method calculate
```



```
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
```



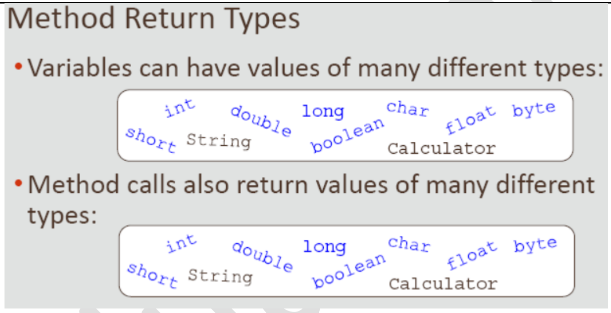
Method Arguments and Parameters

- 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
```

- 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
```



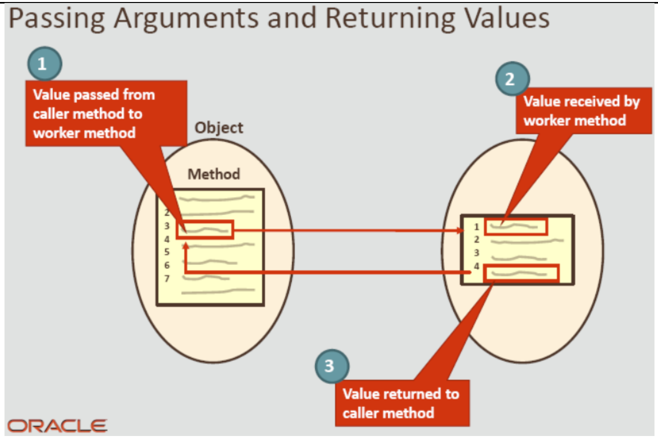
Method Return Types

- Variables can have values of many different types:

int double long char float byte short String boolean Calculator

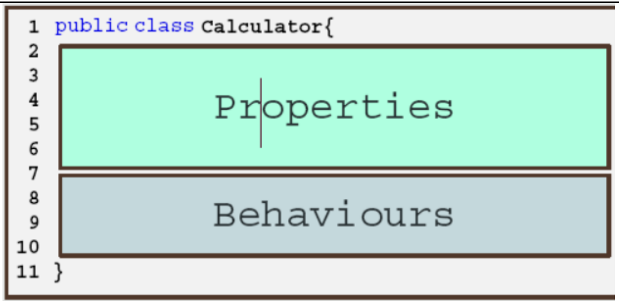
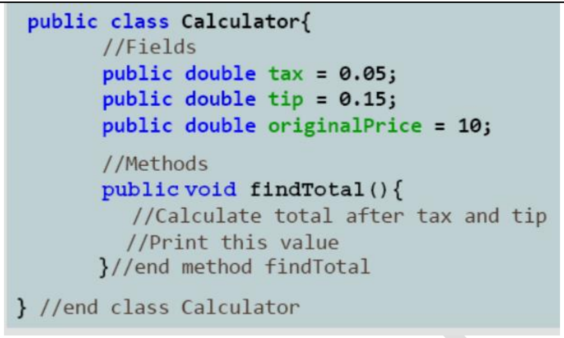
- Method calls also return values of many different types:

int double long char float byte short String boolean Calculator



Passing Arguments and Returning Values

1
Value passed from caller method to worker method
Object
Method
2
Value received by worker method
3
Value returned to caller method
ORACLE

 `public class Calculator{ }`	 `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();

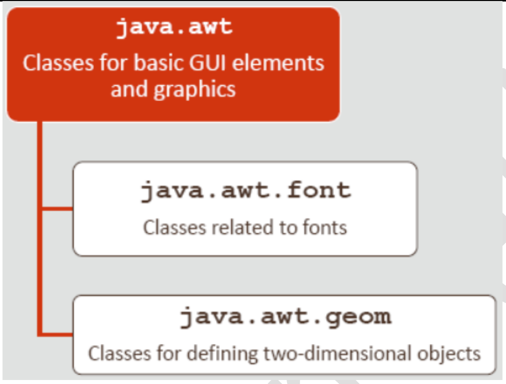
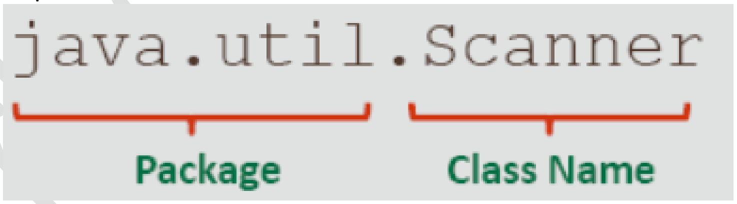
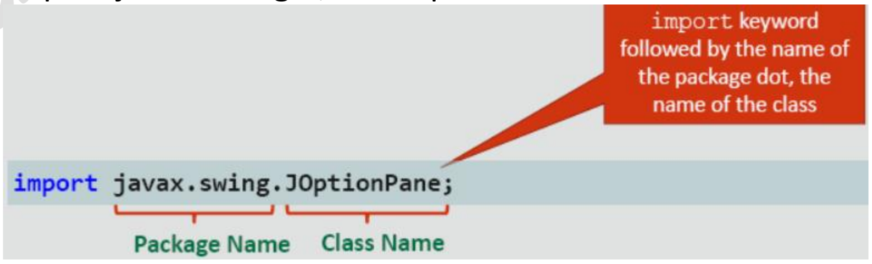
```
public class Calculator{
        //Fields
        public double tax = 0.05;
        public double tip = 0.15;
        public double originalPrice = 10;

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

} //end class Calculator
```

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
javax.swing	Provides classes to build GUI components
java.net	Provides classes for networking applications
java.time	Providesclasses for dates, times, instants, and durations

 java.awt
Classes for basic GUI elements and graphics
java.awt.font
Classes related to fonts
java.awt.geom
Classes for defining two-dimensional objects
	import  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

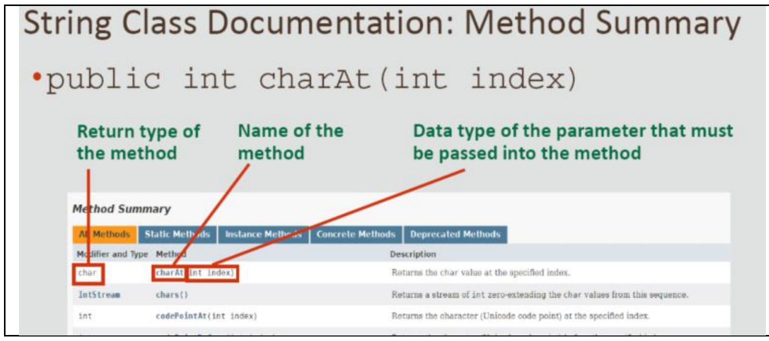
Quiz 2: JFo - Section 4 - L3-L5

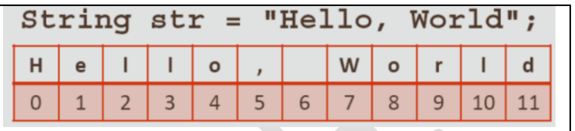
→

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
Name of the method
Data type of the parameter that must be passed into the method
Method Summary
All Methods
Static Methods
Instance Methods
Concrete Methods
Deprecated Methods
Modifier and Type
Method
Description
char
charAt(int index)
Returns the char value at the specified index.
IntStream
chars()
Returns a stream of int zero-extending the char values from this sequence.
int
codePointAt(int index)
Returns the character (Unicode code point) at the specified index.



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

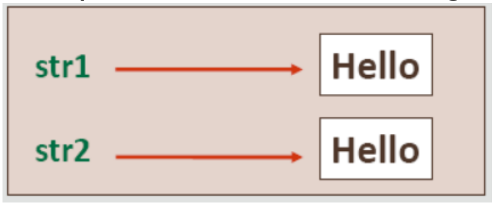
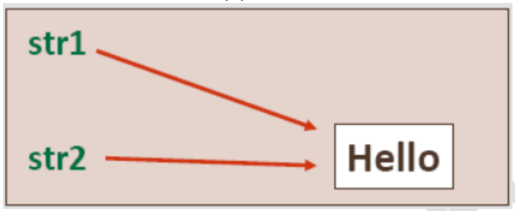
String str = new String(“Hello, World”);
Not commonly used and not recommended

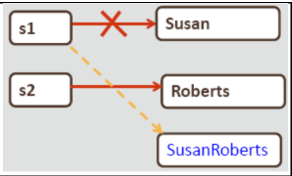
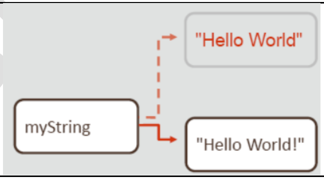
int length()	Returns the length of this string Example: LastName.length()
char charAt(int index)	Returns the char value at the specified index

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

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

→

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**
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)

→

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
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

→

5. Decision Statements

5.1. Boolean Expressions and if/else Constructs

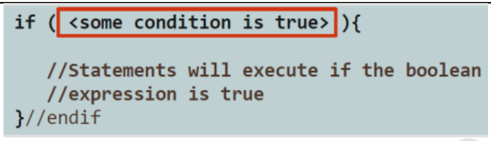
In Java the values for the boolean data type are true and false, instead of yes and no.

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

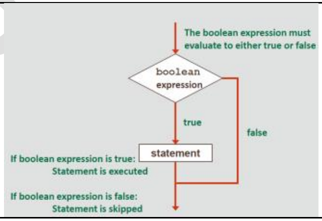
Relational Operators

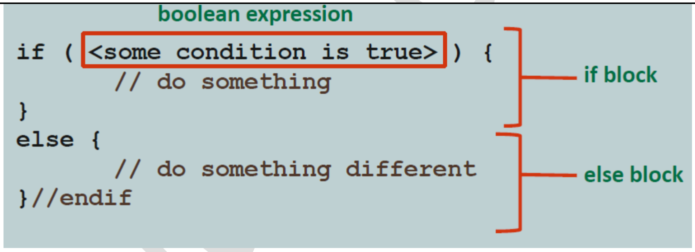
Condition	Operator	Example
Is equal to	==	int i=1; (i == 1)
Is not equal to	!=	int i=2; (i != 1)
Is less than	<	int i=0; (i < 1)
Is less than or equal to	<=	int i=1; (i <= 1)
Is greater than	>	int i=2; (i > 1)
Is greater than or equal to	>=	int i=1; (i >= 1)

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

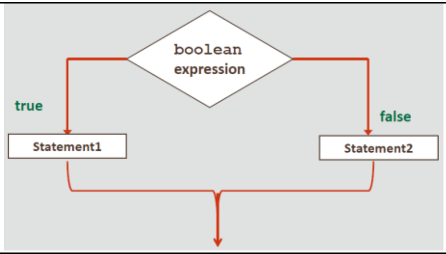


```
if ( <some condition is true> ){

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


The boolean expression must evaluate to either true or false
boolean expression
true
false
statement
If boolean expression is true: Statement is executed
If boolean expression is false: Statement is skipped



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



== 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?

5.2. Understanding Conditional Execution

Handling Multiple Conditions

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

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

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

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

Logic Operator	Meaning
&&	AND
||	OR
!	NOT

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

```java
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

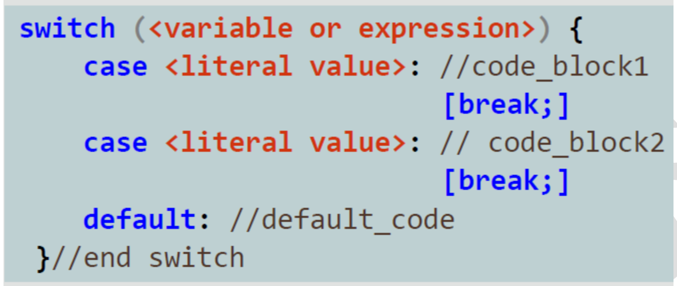
Operation	Operator	Example
If condition is true: assign result = value1 Otherwise: assign result = value2	?:	`result = condition ? value1 : value2` Example: `int x = 2, y = 5, z = 0;` `z = (y < x) ? x : y;`

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

```
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



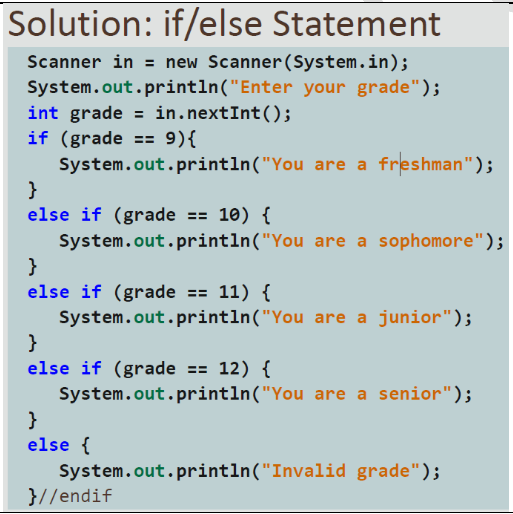
```
switch (<variable or expression>) {
    case <literal value>: //code_block1
                          [break;]
    case <literal value>: // code_block2
                          [break;]
    default: //default_code
 }//end switch
```

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

case : valor

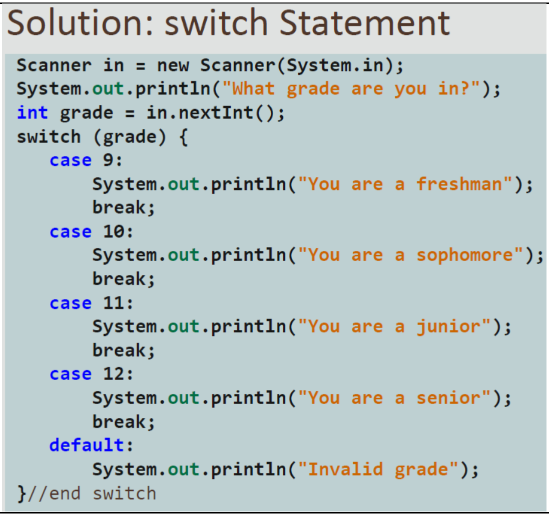
break : Salir del switch

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
```

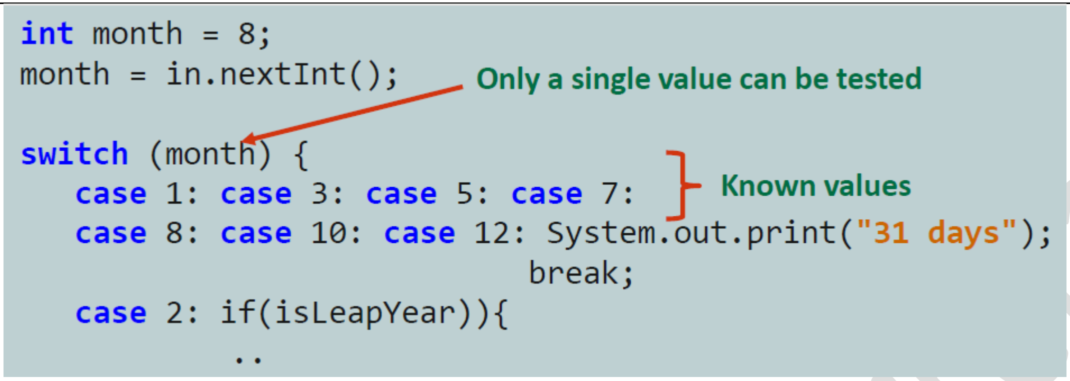


Solution: switch Statement

```
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
```

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)){
           ..
```

→

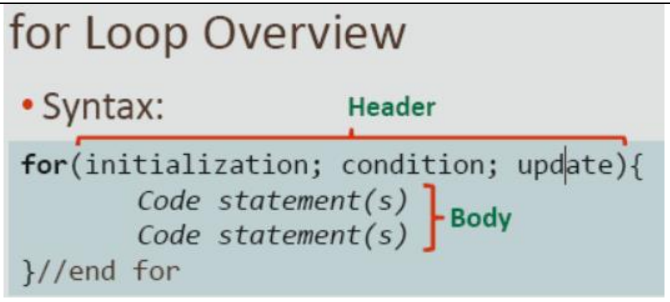
6. Loop Constructs

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:

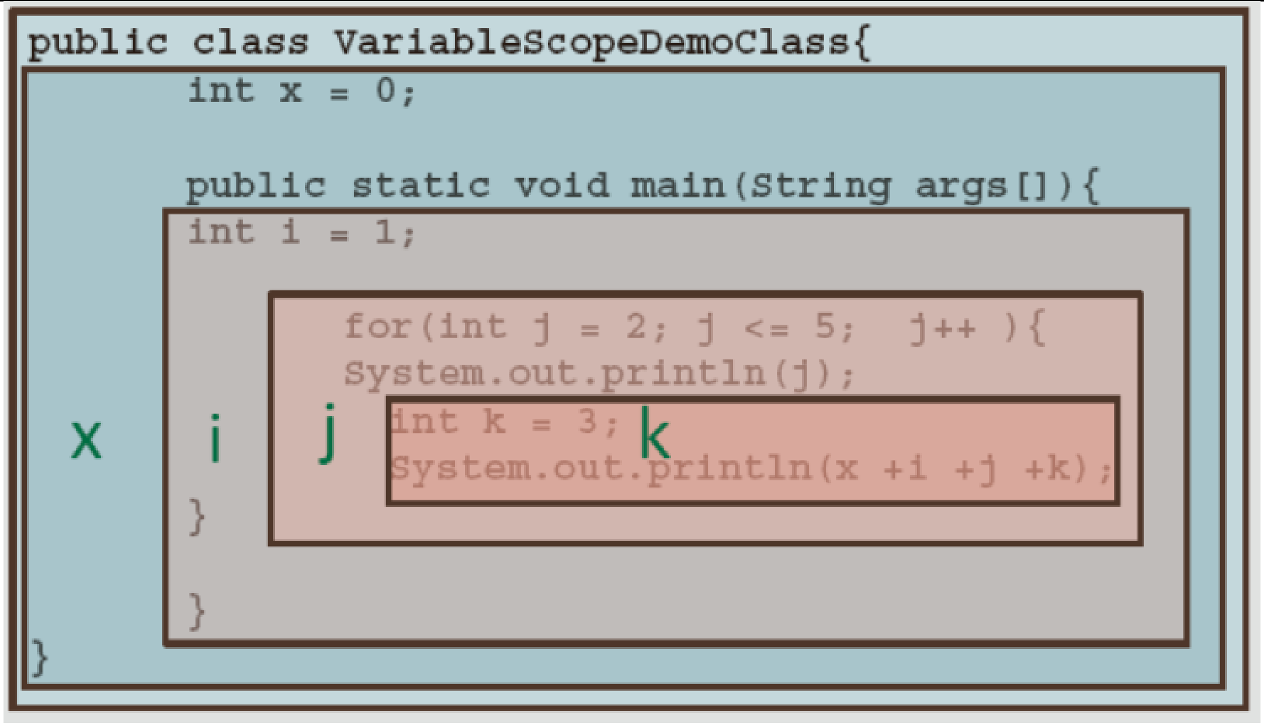
```
for(initialization; condition; update){
        Code statement(s)
        Code statement(s)
}//end for
```

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

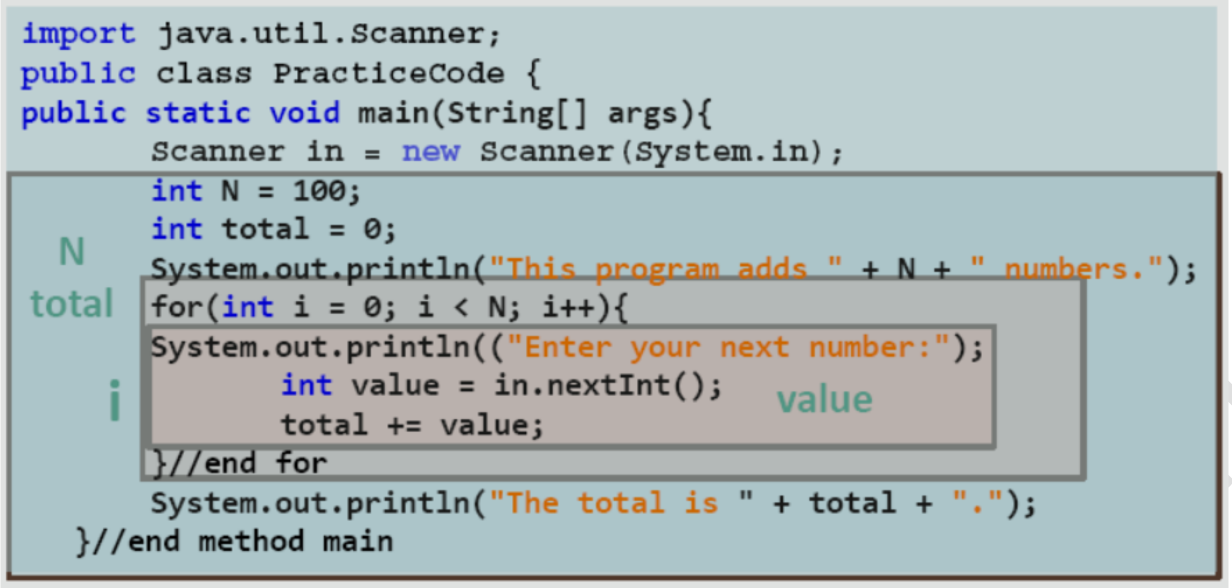
```
System.out.println("Countup to Song: ");
for (int i = 1;  i < 9;  i++) {
   System.out.println(i);
   // incremento implicito
} //end for
System.out.println("Mambo!");

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.

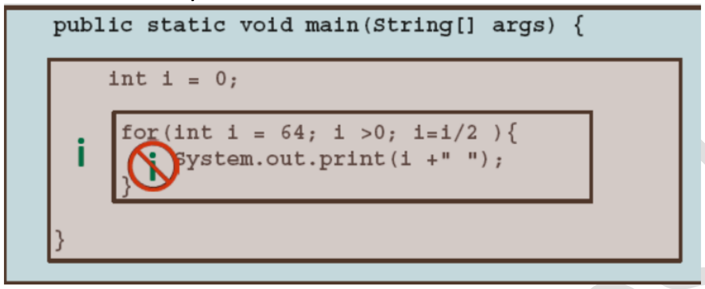


```
public class VariableScopeDemoClass{
        int x = 0;

        public static void main(String args[]){
        int i = 1;

              for(int j = 2; j <= 5;  j++ ){
              System.out.println(j);
                 int k = 3;
                 System.out.println(x +i +j +k);
        }

        }
}
```



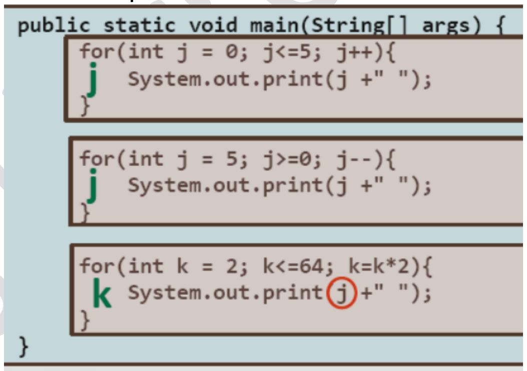
```java
import java.util.Scanner;
public class PracticeCode {
public static void main(String[] args){
        Scanner in = new Scanner(System.in);
        int N = 100;
        int total = 0;
        System.out.println("This program adds " + N + " numbers.");
        for(int i = 0; i < N; i++){
        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



```java
   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



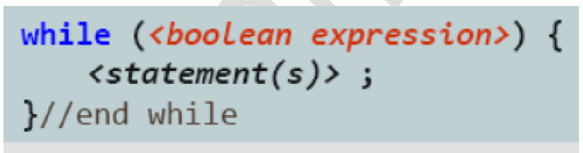
```java
 public static void main(String[] args) {
    for(int j = 0; j<=5; j++){
        System.out.print(j +" ");
    }
    for(int j = 5; j>=0; j--){
        System.out.print(j +" ");
    }
    for(int k = 2; k<=64; k=k*2){
        System.out.print(j +" ");
    }
 }
```

6.2. while and do-while loops

How Many Times to Repeat?

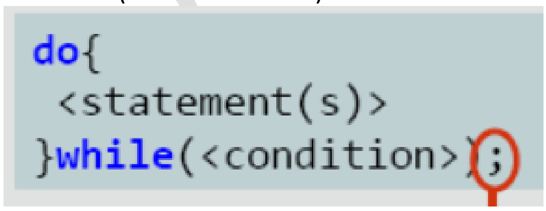
- In some situations, you don't know how many times to repeat something
- That is, you may need to repeat some code until a particular condition occurs

Pre-test



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

Post-test (mínimo una vez)



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

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

```java
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

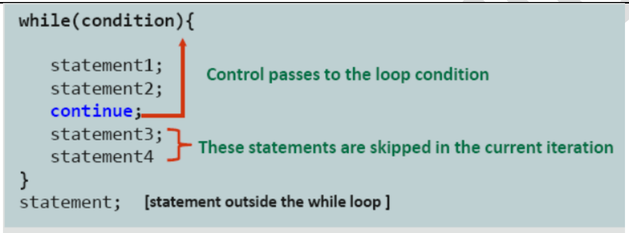
```
for (int i = 10; i >= 0; i--) {
   System.out.println(i);
}
System.out.println("Blast Off!");
```

```
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);
```

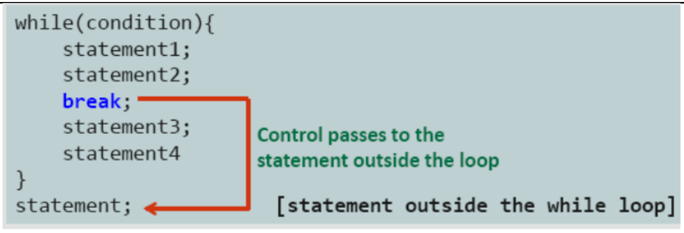
6.3. Using break and continue Statements

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;
```

Control passes to the loop condition
These statements are skipped in the current iteration
[statement outside the while loop]



```
while(condition){
    statement1;
    statement2;
    break;
    statement3;
    statement4
}
statement;
```

Control passes to the statement outside the loop
[statement outside the while loop]

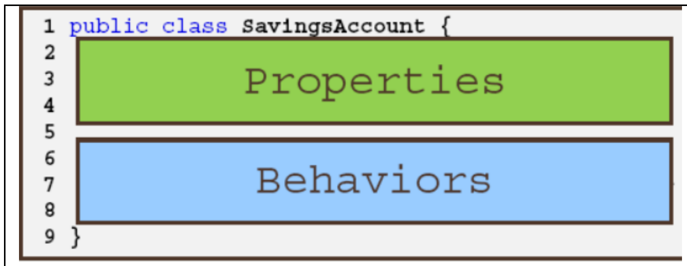
```
int i = 0;
while (i < 10) {
   if (i == 4) {
      break;
   }
   System.out.println(i+ "\t");
   i++;
}
System.out.println("\n. . .Fin");
```

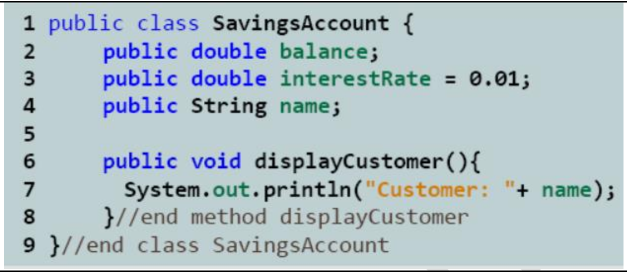
→

7. Creating Classes

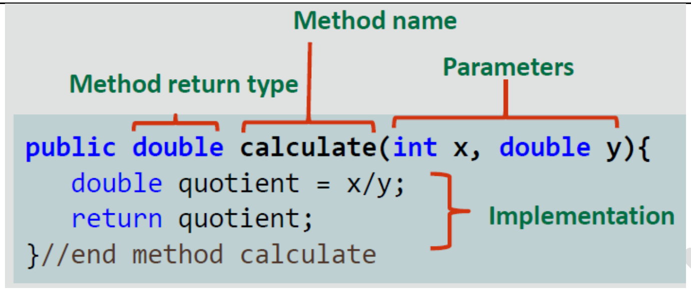
7.1. Creating a Class

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

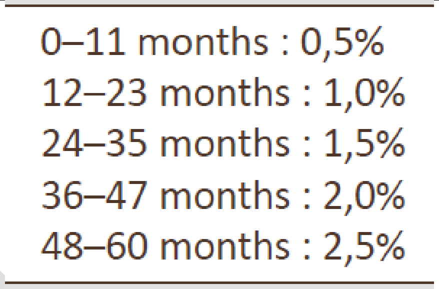




```
public class SavingsAccount {
    public double balance;
    public double interestRate = 0.01;
    public String name;

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



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



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;
}
```

→

```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";
```

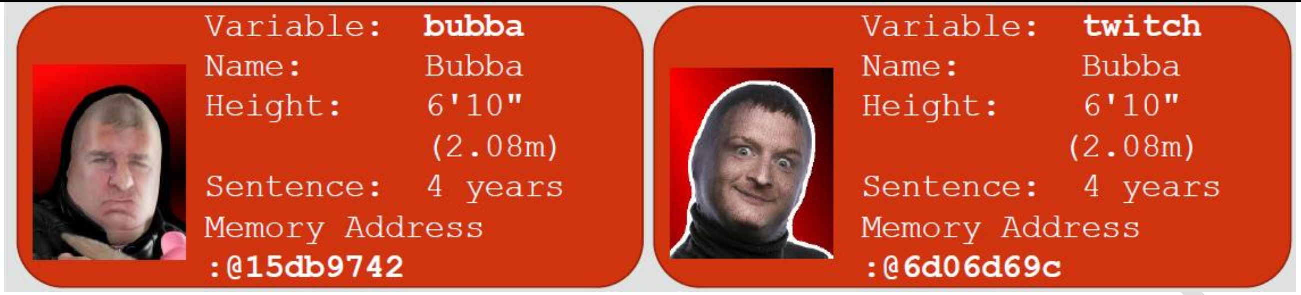
But you shouldn't do this
String s2 = new String("Test");

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;
}
```

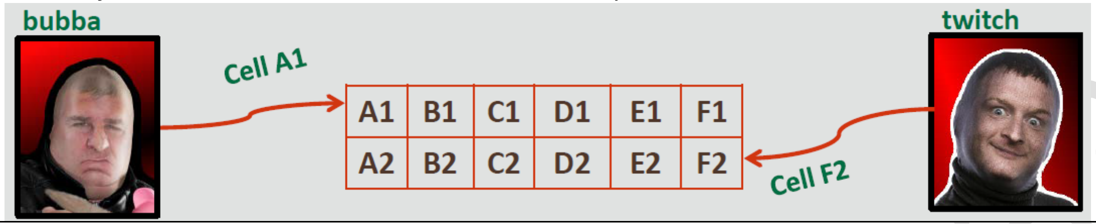
```java
public static void main(String[] args){
   Prisoner bubba  = new Prisoner();
   Prisoner twitch = new Prisoner();
   System.out.println(bubba); // Prisoner@15db9742
   System.out.println(twitch);// Prisoner@6d06d69c
                                // Memory addresses
}
```

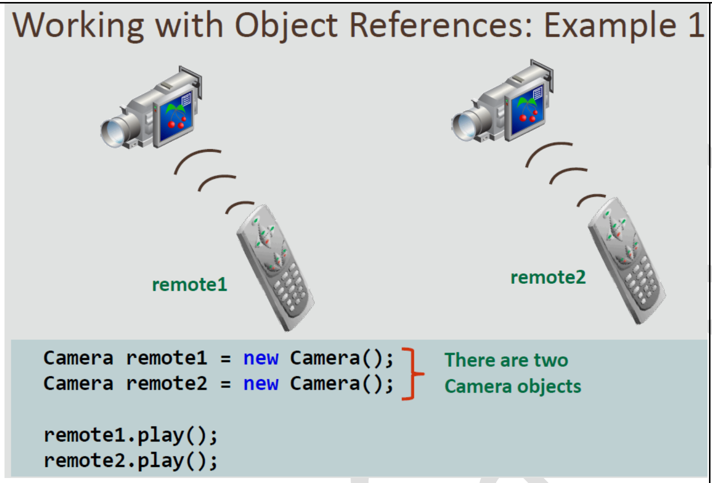


```
Variable:   bubba
Name:       Bubba
Height:     6'10"
            (2.08m)
Sentence:   4 years
Memory Address
:@15db9742
```

```
Variable:   twitch
Name:       Bubba
Height:     6'10"
            (2.08m)
Sentence:   4 years
Memory Address
:@6d06d69c
```

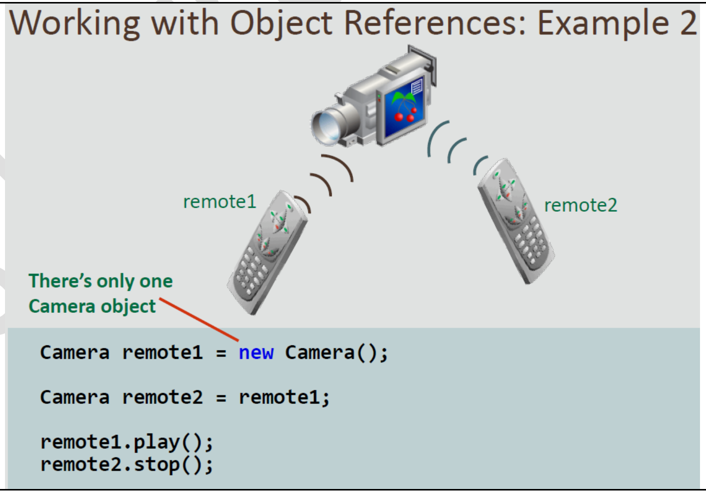
Each object will have a different location in memory





Working with Object References: Example 1

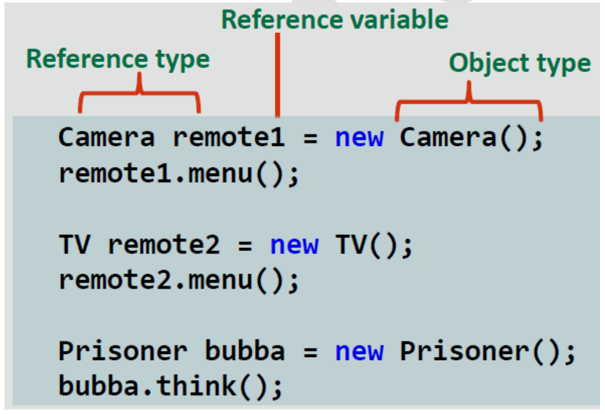
```
Camera remote1 = new Camera();
Camera remote2 = new Camera();

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



Working with Object References: Example 2

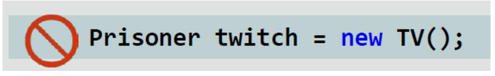
```
Camera remote1 = new Camera();

Camera remote2 = remote1;

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

References to Different Objects: Example



```
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();
```

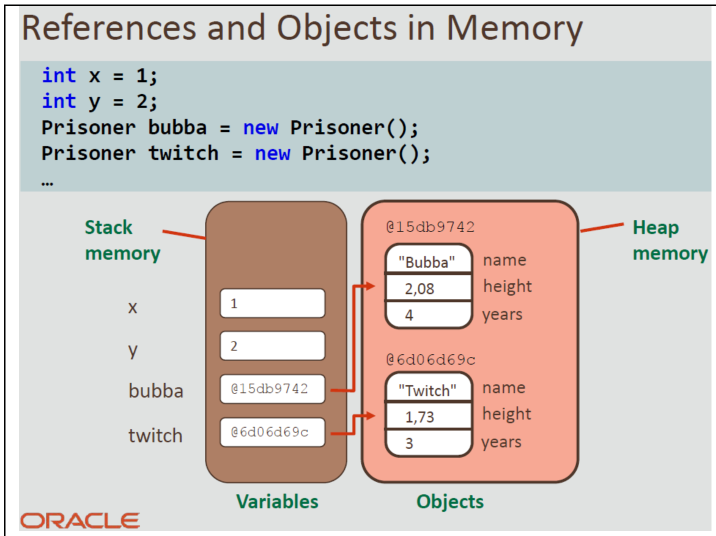
Caballo extends Animal { … }   // El caballo **ES UN** Animal
Animal potro = new Caballo();

Stack memory is used to store …

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

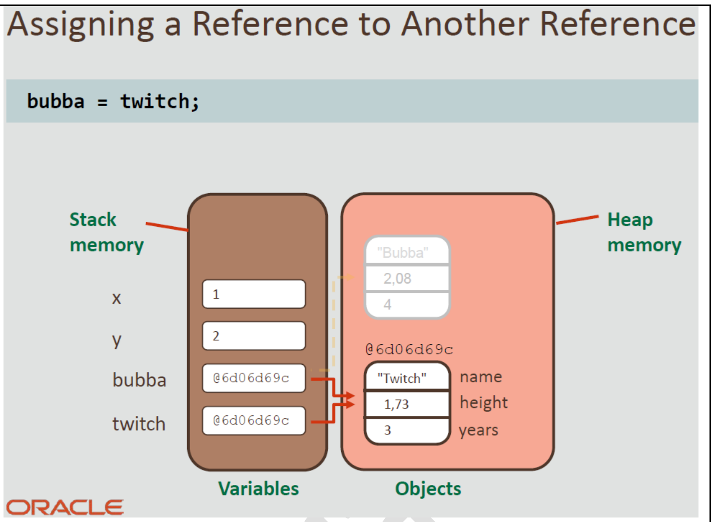
Heap memory is used to store …

- 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);   ??
```

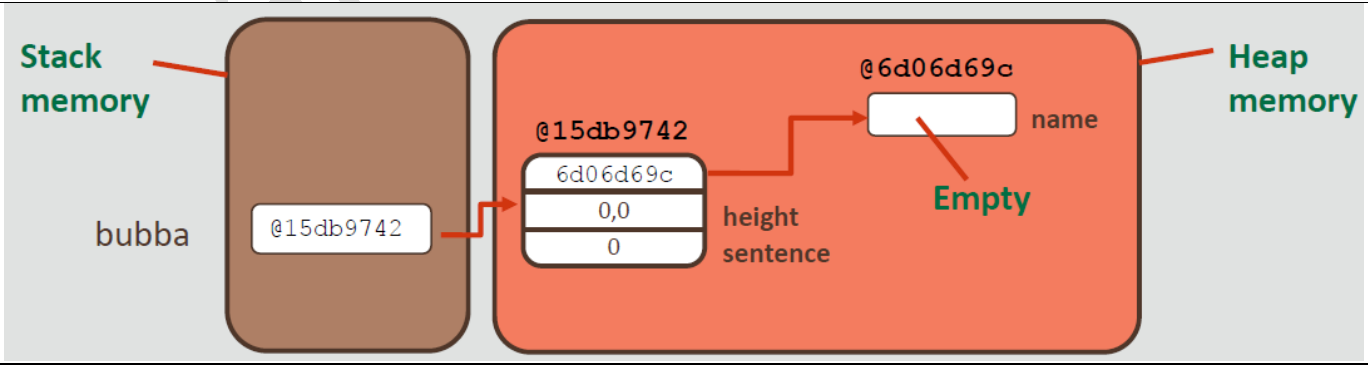
Si ninguna variable de referencia apunta a un objeto... Java borra automáticamente la memoria que ocupaba ese objeto. Esto se denomina recolección de basura (**Garbage Collection**). Los datos asociados a este objeto se pierden para siempre.

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

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

→

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);
    }
}
```

→

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

→

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

→