



Politécnico  
Internacional



# *PROGRAMACIÓN ORIENTADA A OBJETOS*



*Harol Hernán Torres Neuta  
Magister en Educación en Tecnología  
Licenciado en Informática*



*“Cualquier tecnología suficientemente avanzada es equivalente a la magia..”*

*Arthur C. Clarke*



# *ACUERDOS PEDAGÓGICOS*



# ACUERDOS PEDAGÓGICOS

## PROGRAMACION ORIENTADA A OBJETOS - SOFN1-2TS

Sesion	Mes	Dia	Inicia	Finaliza	Parcial Corte I	Parcial Corte II	Parcial Corte III
1	Julio	20	18:00	21:00			
2	Julio	27	18:00	21:00			
3	Agosto	3	18:00	21:00			
4	Agosto	10	18:00	21:00	x		
5	Agosto	17	18:00	21:00			
6	Agosto	24	18:00	21:00			
7	Agosto	31	18:00	21:00			
8	Septiembre	7	18:00	21:00		x	
9	Septiembre	14	18:00	21:00			
10	Septiembre	21	18:00	21:00			x
Miercoles							



# ACUERDOS PEDAGÓGICOS

## ACUERDOS

### 1. PUNTUALIDAD Y ASISTENCIA:

1.1 *Hora de inicio: 6:20 PM MARTES - 07:50 PM JUEVES*

1.2 *Día y hora encuentros sincrónicos:* martes de 6 a 9 pm y Jueves de 7.5 a 9 pm.

1.3 *Trabajo autónomo:* El estudiante es responsable por el trabajo autónomo.

1.4 *Foro de dudas académicas:* En este foro el estudiante presentará las inquietudes que tenga con el desarrollo de la asignatura. El docente responderá dentro de las **12** horas hábiles siguientes. En ningún caso el tiempo de respuesta del docente puede ser superior a 24 horas después de la solicitud del estudiante. Los estudiantes revisarán las preguntas consignadas por sus compañeros, para identificar si la inquietud que tiene ya fue planteada por un compañero y respondida por el docente. En caso que en el foro no haya una pregunta igual, registrará su pregunta allí. Esto para hacer más eficiente el tiempo de respuesta.

1.5 *Vídeo conferencia:* La video conferencia es el encuentro sincrónico (personal, en línea). En cada sesión se realizarán dos video conferencias, las cuales se aclaran en el desarrollo de la cada sesión.



# ACUERDOS PEDAGÓGICOS

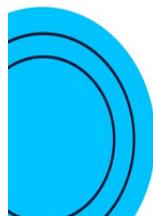
## 6. CÓDIGO DE CONDUCTA

- 6.1 Se trabajará en un clima de respeto, orden y cordialidad entre alumnos y docente. Se considera falta de respeto una agresión verbal de cualquier tipo.  
La docente se compromete a cumplir los horarios acordados tanto para encuentros sincrónicos, como para retroalimentaciones.
- 6.2 Las entregas de trabajos se han pautadas con anticipación.
- 6.3 Es necesario que en las video conferencias los estudiantes mantengan los micrófonos y cámaras apagados. La participación debe ser por el chat, excepto que la docente les indique que pueden participar por audio o vídeo.
- 6.4 El cometer faltas en las actividades, como plagio (adueñarse de propiedad intelectual sin referenciar al autor), realización de tareas a otros compañeros, duplicar actividades elaboradas por otros compañeros, entre otros, implica un 0.0 en la nota, al igual que en la nota cualitativa del profesor y se iniciará un proceso disciplinario.
- 6.5 Este acuerdo pedagógico se socializa en la primera sesión. Es susceptible de modificaciones según lo acordado entre estudiantes y docente. Una vez quede definitivo en sesión 1, la docente lo publicará en el foro “Noticias”, en el cual debe escribir cada estudiante indicando su conformidad o no, durante los siguientes 3 días hábiles. Para los estudiantes que no participen en el foro durante este tiempo, se entenderá que lo aceptan.
- 6.6 La comunicación por WhatsApp será solo de lunes a viernes hasta las 18:00 pm solo a través de mensajes.
- 6.7 La comunicación no se realizará a través de llamadas si no es autorizada por el docente.
- 6.8 Las entregas de trabajos se deben realizar dentro de las fechas sugeridas.
- 6.9 Los parciales se realizarán en las fechas sugeridas, no se abrirá plataforma después.



○ ○ ○ ○ ○ ○ ○

Transformamos  
**la evaluación**  
para mejorar



○ ○ ○ ○ ○ ○ ○  
○ ○ ○ ○ ○ ○ ○  
○ ○ ○ ○ ○ ○ ○  
~ ~ ~ ~ ~ ~ ~

A partir de este ciclo los **porcentajes en las evaluaciones de cada corte** han cambiado de la siguiente forma:

1<sup>er</sup> corte **15%**  
2<sup>do</sup> corte **35%**  
3<sup>er</sup> corte **50%**



Si tienes alguna inquietud  
**consulta con tu mentor**



**Primer corte (Parcial 1):** semana 4. Las notas obtenidas a esta fecha más la nota de la evaluación parcial tienen un peso del 35% de la nota final.

**Segundo corte (Parcial 2):** semana 8. Las notas obtenidas entre las semanas 5 y 8 más la nota de la evaluación parcial tienen un peso del 35% en la nota final.

**Tercer corte (Final):** semana 10. Las notas obtenidas durante las semanas 9 y 10 más la nota de la evaluación parcial tienen un peso del 30% en la nota final, cumpliendo así con el 100% de la nota de la asignatura.



# CONTENIDOS



<b>ASIGNATURA:</b>	PROGRAMACIÓN ORIENTADA A OBJETOS
<b>CICLO</b>	II
<b>CODIGO</b>	0554
<b>CREDITOS</b>	3
<b>INTENSIDAD HORARIA</b>	4.5 horas semanales
<b>PRERREQUISITOS</b>	Diseño de Algoritmos
<b>CORREQUISITOS</b>	Ninguno
<b>COMPETENCIA DE ENTRADA</b>	El estudiante resuelve problemas empleando diagramas de flujo y lenguajes de programación (Python) teniendo en cuenta los procedimientos y los métodos más comunes de la lógica matemática, con el fin de proponer soluciones concretas a una situación problema.
<b>COMPETENCIA DE SALIDA</b>	El estudiante desarrolla aplicaciones de consola y de escritorio en el lenguaje Python/Java haciendo uso de arquitectura MVC, programación orientada a objetos y técnicas de desarrollo de software para cumplir con los requerimientos de los clientes.
<b>PERFIL DOCENTE</b>	Ingeniero de sistemas o electrónicos con experiencia en desarrollo de software.
<b>VERSIÓN</b>	01
<b>FECHA DE IMPLEMENTACIÓN</b>	2019-2T



COMPETENCIA DE SALIDA	
El estudiante desarrolla aplicaciones de consola y de escritorio en el lenguaje Python/Java haciendo uso de arquitectura MVC, programación orientada a objetos y técnicas de desarrollo de software para cumplir con los requerimientos de los clientes.	
INDICADORES DE DESEMPEÑO	
<p>1.1 Propone una solución concreta a una situación problema a partir de un diagrama de flujo, utilizando los procedimientos y los métodos más comunes de la lógica matemática</p> <p>1.2 Propone una solución concreta a una situación problema a partir de lenguaje Python/Java, utilizando los procedimientos y los métodos más comunes de la lógica matemática.</p> <p>1.3 Argumenta el diagrama de flujo y el código propuesto para la solución de una situación problema.</p>	<ul style="list-style-type: none"><li>• clases<ul style="list-style-type: none"><li>✓ introducción a la programación orientada a objetos</li><li>✓ definición de clase, método, atributos, instancias y objetos</li><li>✓ atributos de objetos</li><li>✓ método init</li><li>✓ definición de objetos</li><li>✓ atributos privados y públicos</li><li>✓ definición de métodos</li><li>✓ funciones get y set</li><li>✓ representaciones de objetos (<code>_eq_, _add_, _sub_</code>)</li><li>✓ polimorfismo</li><li>✓ herencia</li></ul></li><li>• sistemas gráficos<ul style="list-style-type: none"><li>✓ tkinter - ttk<ul style="list-style-type: none"><li>○ ventanas</li><li>○ geometría</li><li>○ grilla</li><li>○ etiquetas</li><li>○ botones</li><li>○ entradas</li><li>○ frames</li><li>○ adicionales</li></ul></li><li>✓ Javax -JavaFX<ul style="list-style-type: none"><li>○ ventanas</li></ul></li></ul></li></ul>

	<ul style="list-style-type: none"><li>○ geometría</li><li>○ grilla</li><li>○ etiquetas</li><li>○ botones</li><li>○ entradas</li><li>○ frames</li><li>• adicionales</li></ul>
REQUERIMIENTOS DE INFRAESTRUCTURA O MEDIOS EDUCATIVOS	<ul style="list-style-type: none"><li>• Equipos con acceso a internet, proyector, IDE de programación (Netbeans, Eclipse, Jcreator), java virtual machine y acceso a internet.</li><li>• IDE(Thonny, eclipse, Netbeans, Vscode, Geany) openJDK, Python, proyector, Git bash.</li></ul>



# PRESENTACIÓN DE ESTUDIANTES

1. Nombre
2. A que se dedica
3. ¿Qué es programación orientada a objetos? ¿Qué es una BD?
4. ¿Ha programado en Java?
5. ¿Conoce la serie Stranger Things de NetFlix?.



<https://padlet.com/harol003/yc637crn2k99px23>



*Todos deberían aprender a programar*

<https://www.youtube.com/watch?v=Y1HHBXDL9bg>

"Everybody in this country should  
learn how to program a computer..."

"Todo el mundo en este país debería aprender a programar un



# *RECORDEMOS ALGUNOS CONCEPTOS*



*Un lenguaje de programación tiene distintas maneras de aproximarnos a la resolución de los problemas y diferentes estilos de programación.*

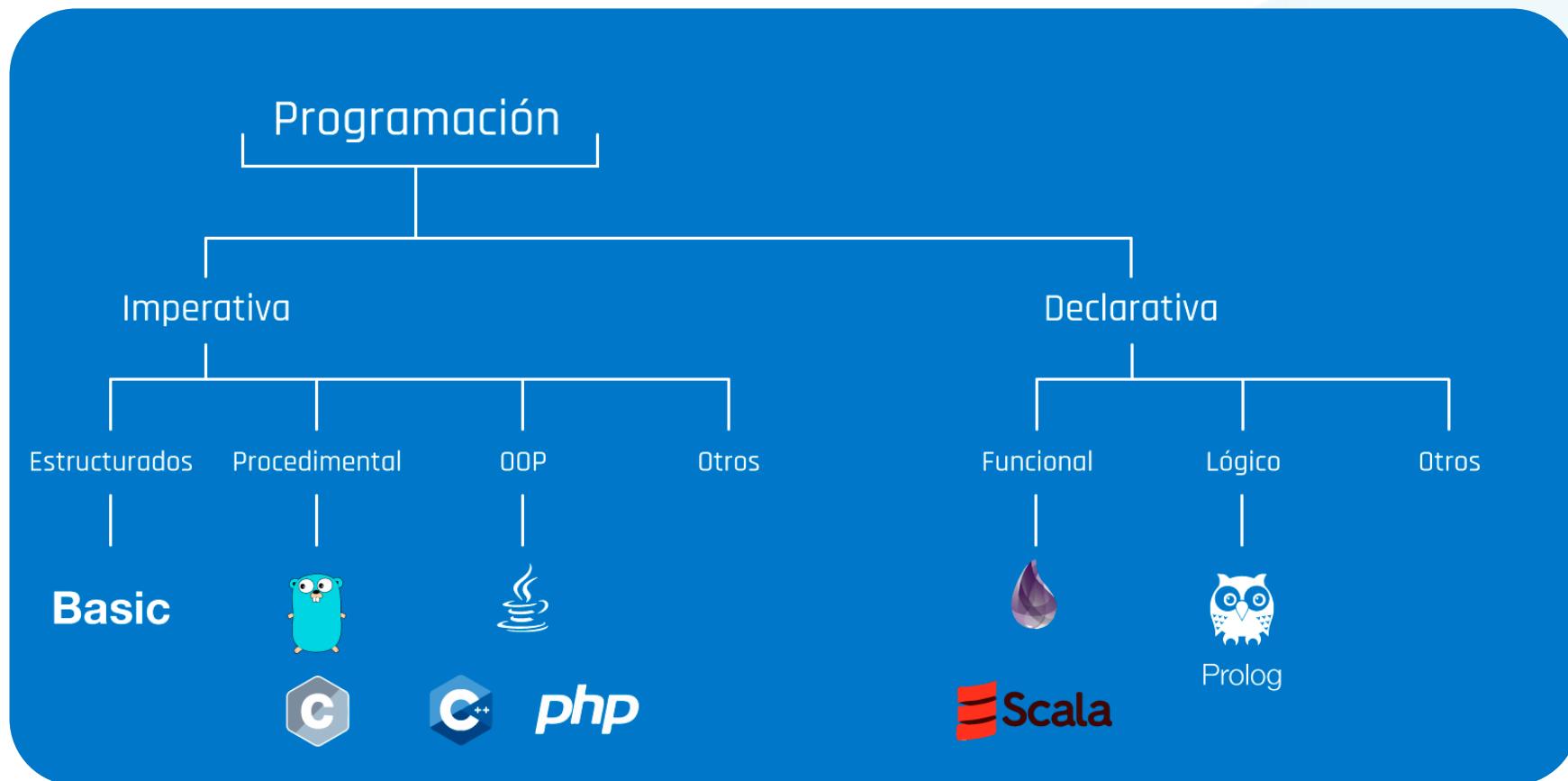
*Por tanto, se clasifican en lenguajes de programación:*

- *Imperativos*
- *Orientados a Objetos*
- *Funcionales*
- *Lógicos*

## PARADIGMAS DE PROGRAMACIÓN



# PARADIGMAS DE PROGRAMACIÓN





## ¡PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS!

The infographic is titled "¡PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS!" and features four main pillars represented by classical columns:

- ABSTRACCIÓN**: Es el proceso de **definir los atributos y los métodos** de una clase.
- ENCAPSULAMIENTO**: Protege la información de manipulaciones no autorizadas.
- POLIMORFISMO**: Da la misma orden a varios **objetos** para que respondan de diferentes maneras.
- HERENCIA**: Las **clases hijo heredan atributos y métodos** de las clases padre.

According to the paradigm, object-oriented programming is based on these 4 pillars. **Estos definen la simplicidad y la funcionalidad del código.**

¿Cómo utilizas la POO actualmente?  
 [ed.team/cursos/poo](https://ed.team/cursos/poo)

EDteam

# PARADIGMAS DE PROGRAMACIÓN



# TIPOS DE LENGUAJE

The infographic is divided into three main sections: **COMPILED**, **INTERPRETADO**, and **INTERMEDIO**.

- COMPILED:** Shows icons for C# and C++. It explains that it converts code to binaries that the operating system reads. An illustration shows a laptop with code being converted into binary (01) and then into a running application.
- INTERPRETADO:** Shows icons for JS and Python. It explains that it requires a program to read the code instruction by instruction in real-time and execute it. An illustration shows a keyboard connected to a browser icon.
- INTERMEDIO:** Shows icons for Scala and Java. It explains that it compiles source code into an intermediate language, which is then executed in a virtual machine. An illustration shows a laptop with code being converted into a virtual machine icon.

Domina todos lenguajes de programación estudiando en EDteam

[ed.team/cursos](https://ed.team/cursos)

# TIPOS DE LENGUAJE





- *¿Qué es un algoritmo?*
- *¿Qué es el Pseudocódigo?*
- *Técnicas para la formulación de algoritmos.*
  - *Pseudocódigo.*
  - *Diagramas de Flujo.*
  - *Diagramas Estructurados.*
- *Prueba de escritorio.*

## RECORDEMOS





# ¿QUÉ ES UN ALGORITMO?

Es la secuencia de pasos que resuelve un problema y es la base de la programación.

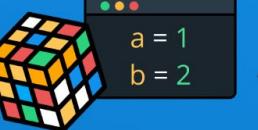
Prof. Alexys Lozada



## PARTES DE UN ALGORITMO

### ENTRADA

Son los datos que se le dan al algoritmo.



### PROCESO

Operaciones que se hacen con los datos.



### SALIDA

Resultado final que se obtiene de las operaciones, en este caso será 3.



## CARACTERÍSTICAS



### PRECISO

Tiene que resolver el problema sin errores.



### DEFINIDO

Si ejecutas el algoritmo varias veces, los datos de salida serán iguales en cada repetición.



### FINITO

Debe tener un inicio y un final.

### LEGIBLE

Cualquier persona que vea el algoritmo debe ser capaz de comprenderlo.

Aprende como funcionan los algoritmos en:

👉 [ed.team/cursos/algoritmos](https://ed.team/cursos/algoritmos)



# RECORDEMOS ALGORITMOS



INSTRUCCIÓN	entrada	menor	suma	Pantalla
leer entrada	10			
menor = entrada		10		
suma :=0			0	
suma :=suma + entrada			10	
leer entrada	7			
menor = entrada		7		
suma :=suma + entrada			17	
leer entrada	9			
suma :=suma + entrada			26	
leer entrada	0			
Escribir "valor menor:"			Valor Menor	
Escribir menor			7	
Escribir "Suma:"			Suma:	
Escribir suma			26	

# PRUEBA DE ESCRITORIO





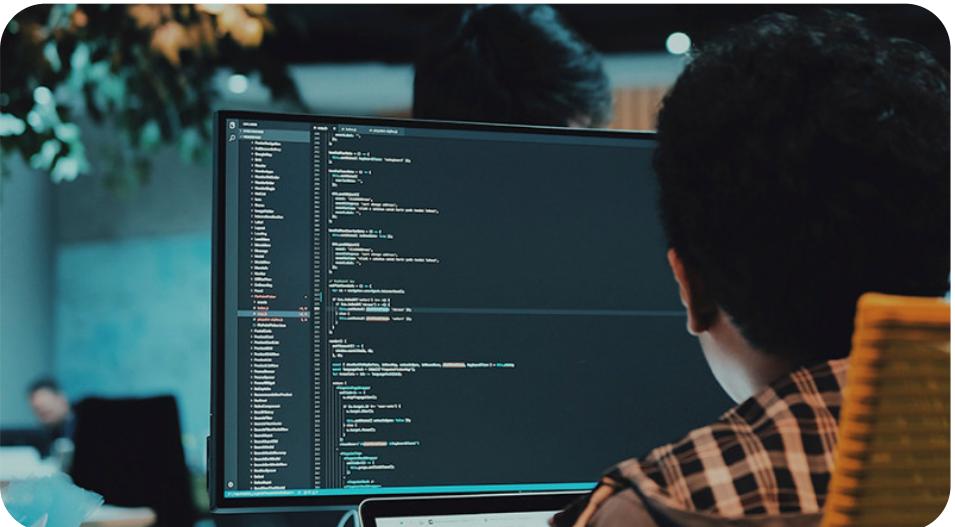
Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

# RECORDEMOS DIAGRAMAS DE FLUJO





# PSEUDOCÓDIGO



## Pseudocódigo

- 1.Inicio**
- 2.Inicializar variables: A = 0, B = 0
- 3.Solicitar la introducción de dos valores distintos
- 4.Leer los dos valores
- 5.Asignarlos a las variables A y B
- 6.**Si A = B Entonces** vuelve a 3 porque los valores deben ser distintos
- 7.**Si A>B Entonces**  
**Escribir A, "Es el mayor"**
- 8.**De lo contrario:** **Escribir B, "Es el mayor"**
- 9.**Fin\_Si**





*¿Y entonces como vamos a trabajar?*



# SINTAXIS BÁSICA DE JAVA

Todos los archivos pertenecen a un paquete

Importa los paquetes para el proyecto

Java usa clases para ejecutar el código

Se debe indicar el tipo de dato.

Modificadores de acceso: private, public, protected o por defecto ninguno.

El método principal en Java es el método main

La palabra reservada new crea un objeto del tipo de dato especificado.

```
package team.ed.course;  
import java.lang.*;  
  
public class Person {  
    private String name;  
  
    public static void main(String args[]){  
        Person friend = new Person();  
  
        friend.name = "Peter";  
  
        System.out.println("Hola " +  
                           friend.name);  
    }  
}
```

Se utilizan ; para cada sentencia

Se usan llaves para identificar el bloque de código

Aprende Java desde cero hasta nivel Jedi en:  
[ed.team/java](http://ed.team/java)

EDteam





## 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones creada para resolver una necesidad específica.**

**1 GOOGLE GUAVA** Mejora del flujo de trabajo destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

**2 RXJAVA** Crea aplicaciones para reaccionar a los flujos de datos en tiempo real.

**3 MPANDROIDCHART** Crea gráficos de líneas, barras, radares y burbujas para Android.

**4 FASTJSON** Convierte objetos Java en su representación JSON y viceversa.

**5 MOCKITO** Librería de código abierto para simular pruebas unitarias.

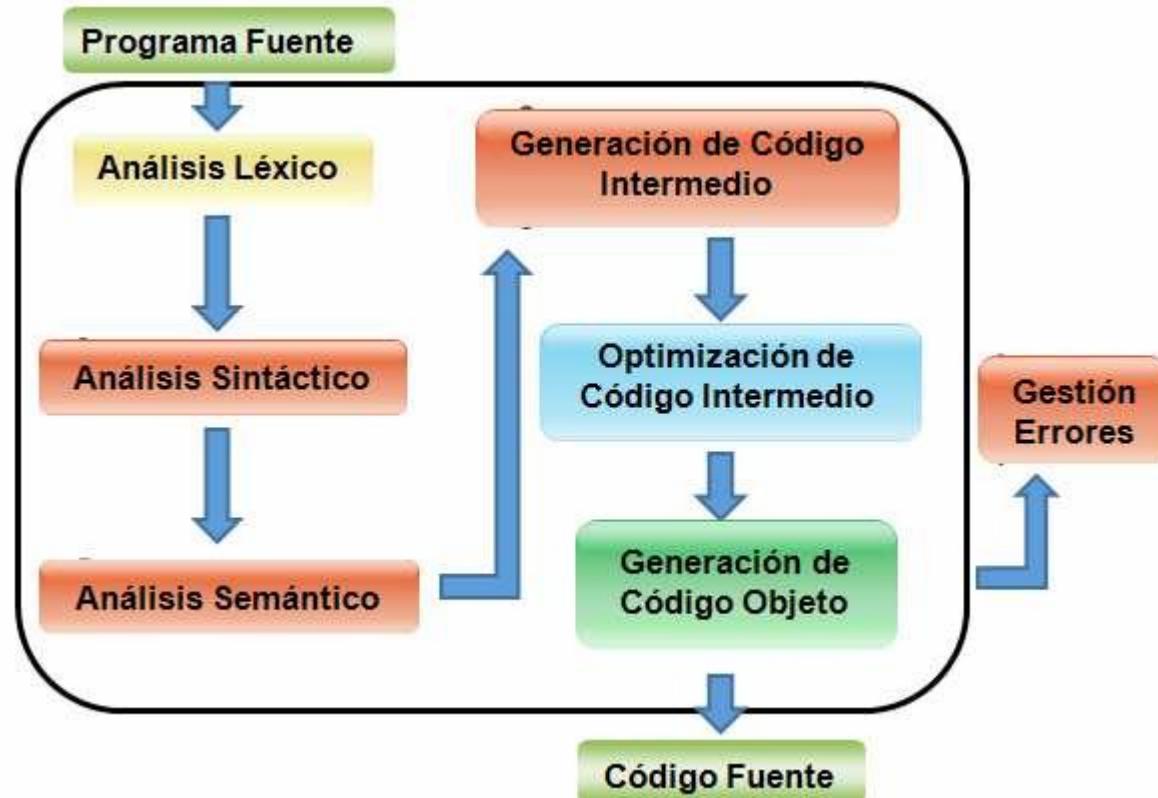
**6 JUNIT** Esta es la librería más usada para pruebas.

Aprende Java desde cero hasta nivel Jedi en: [ed.team/cursos/java](http://ed.team/cursos/java)

EDteam



## ETAPAS EN LA GENERACION DE UN PROGRAMA





<https://forms.office.com/r/wdXpAYYip>



*¿preguntas?*



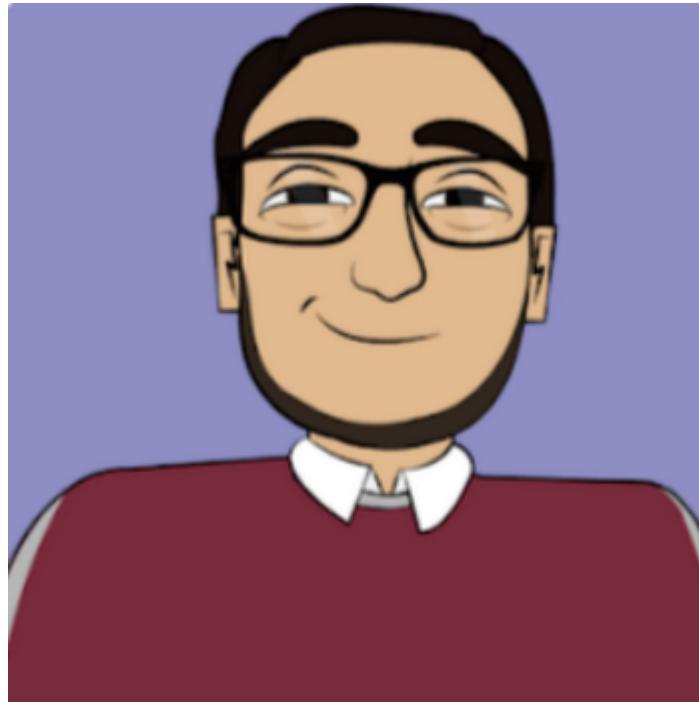
# CONCLUSIONES



Politécnico  
Internacional



# *PROGRAMACIÓN ORIENTADA A OBJETOS*



*Harol Hernán Torres Neuta  
Magister en Educación en Tecnología  
Licenciado en Informática*



*“La tecnología es una palabra que describe algo que no funciona todavía.”*



## *PRESENTACIÓN DE ESTUDIANTES*

1. *Nombre*
2. *A que se dedica*
3. *¿Qué es programación orientada a objetos? ¿Qué es una BD?*
4. *¿Ha programado en Java?*
5. *¿Conoce la serie Stranger Things de NetFlix?.*



<https://padlet.com/harol003/yc637crn2k99px23>



# *RECORDEMOS ALGUNOS CONCEPTOS*

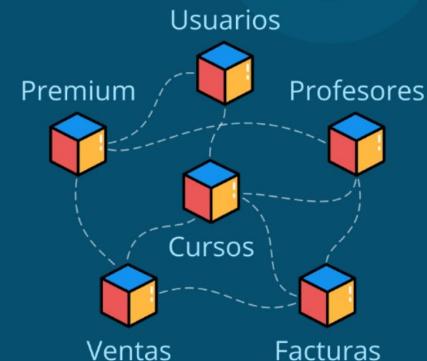


## ¿QUÉ ES LA PROGRAMACIÓN ORIENTADA A OBJETOS?

- Los objetos se crean a partir de una plantilla llamada **clase**. Cada objeto es una instancia de su clase.



- En una aplicación los objetos están separados **pero se comunican entre ellos**.



Es un **paradigma de programación** que organiza las funciones en entidades llamadas **objetos**.



**OBJETO**

- Los objetos tienen **datos (atributos)** y **funcionalidades (métodos)**.

**ATRIBUTOS**

Nombres  
Apellidos  
Correo  
Contraseña  
Premium

**MÉTODOS**

Editar perfil  
Iniciar sesión  
Cerrar sesión  
Cambiar contraseña  
Pasar a premium



Puedes programar con este paradigma **en la mayoría de lenguajes**.

Aprende a crear aplicaciones usando la POO en:

 [ed.team/cursos](https://ed.team/cursos)

 EDteam





## PARADIGMA

*Un paradigma se conceptualiza como un modelo o patrón de cualquier disciplina científica o de otro contexto. Puede concebirse como un conjunto de ideas, creencias, pensamientos, técnicas, métodos, entre otros elementos. Ese conjunto es adoptado por grupos de personas comprometidos e identificados con la misma para la resolución de un problema relacionada con el tema*

## CONCEPTOS BASICOS





## PARADIGMAS DE PROGRAMACIÓN

*Un paradigma de programación se concibe como una propuesta tecnológica que un grupo de personas adopta para la resolución de un problema claramente delimitado. Un paradigma de programación supone la formulación de técnicas, lenguajes, métodos o cualquier otro elemento que permita el cambio radical o parcial de otro paradigma anterior. Por ejemplo, el paradigma de la orientación a objetos vino a sustituir la programación tradicional basada en una estructura procedimental*

## CONCEPTOS BASICOS





## TIPOS DE PARADIGMAS

*A través de la historia de la computación han existido varios paradigmas de programación. Al día de hoy la programación orientada a objetos es la que se encuentra vigente y la que más compañías de software utilizan en el mundo. Los tipos de paradigmas se han gestado gracias a la investigación tecnológica, el aporte de usuarios y las experiencias vividas tanto en el desarrollo de software como también en la funcionalidad del mismo.*

## CONCEPTOS BASICOS





Figura 3.1 Tipos de paradigmas de programación.

## CONCEPTOS BASICOS





## ¡PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS!

The infographic is titled "¡PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS!" and features four main pillars represented by classical columns and 3D cubes:

- ABSTRACCIÓN**: Represented by a column icon and a checklist icon. Description: Es el proceso de **definir los atributos y los métodos** de una clase.
- ENCAPSULAMIENTO**: Represented by a column icon and a lock icon. Description: Protege la información de manipulaciones no autorizadas.
- POLIMORFISMO**: Represented by a column icon and icons of a computer monitor, an envelope, and two people. Description: Da la misma orden a varios objetos para que respondan de diferentes maneras.
- HERENCIAS**: Represented by a column icon and three 3D cubes. Description: Las clases hijo heredan atributos y métodos de las clases padre.

According to the paradigm, object-oriented programming, based on these 4 pillars, defines the simplicity and functionality of the code.

¿Cómo utilizas la POO actualmente?  
[ed.team/cursos/poo](http://ed.team/cursos/poo)

EDteam

## CONCEPTOS BASICOS





## OBJETOS

*Los objetos son entidades que comparten las mismas características. Se agrupan en clases funcionales y pueden ser utilizados mediante instanciación. Conceptualmente, un objeto posee dos contextos: un estado y un comportamiento. El estado se refiere a la configuración inicial de todos y cada uno de los atributos de un objeto. El comportamiento son los cambios que se producen en los estados iniciales de un objeto o la lógica que se ejecuta en un método de la clase instanciada.*

## CONCEPTOS BASICOS



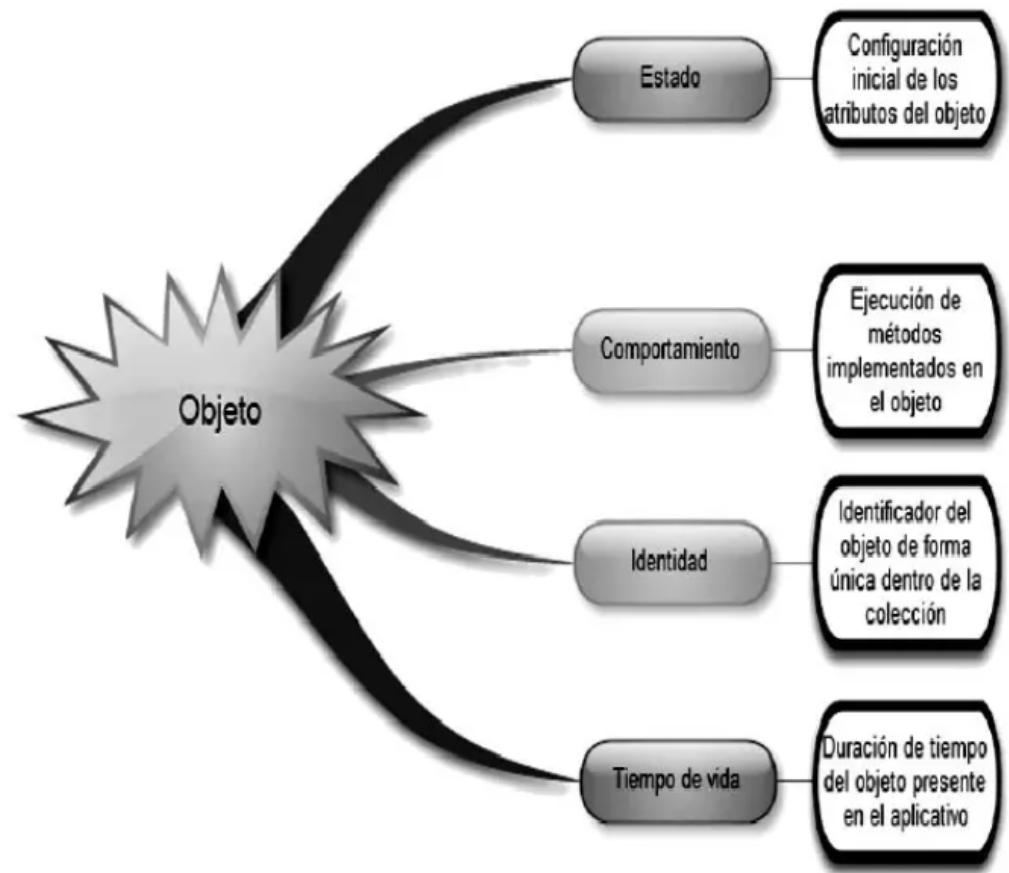


Figura 3.2 Características básicas de un objeto.

## CONCEPTOS BASICOS





### *Definición: CLASE*

*Una clase conjunta elementos que comparten las mismas propiedades y métodos. La instanciación de una clase crea lo que se denomina un objeto. Existen superclases y subclases. Con ello se crea lo que se llama jerarquía de clases, donde la clase padre hereda a las clases hijas.*

## **CONCEPTOS BASICOS**





### *Definición: HERENCIA*

*Es la facilidad mediante la cual una clase hereda las propiedades y métodos públicos de otra clase. La herencia puede ser simple o múltiple. Es simple cuando una clase hereda solamente de otra clase superior (se convierte una relación subclase - superclase) mientras que la herencia múltiple es cuando la clase hereda de 2 o más clases superiores.*

## **CONCEPTOS BASICOS**





### *Definición: OBJETO*

*Es una entidad instanciada de una clase provista de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos) que básicamente responden a eventos. El objeto puede instanciarse de una clase directamente o de una clase derivada la cual hereda de otra.*

## **CONCEPTOS BASICOS**





*Definición: **METODO***

*Constituye la lógica de un objeto.  
Algoritmo que se implementa para  
realizar una determinada operación  
sobre los atributos de la clase o de  
una clase derivada*

## CONCEPTOS BASICOS





## *Definición: EVENTO*

*Es el encargado de que un método se ejecute. Entre los eventos programáticos más comunes se tiene el clic que se produce sobre un botón, el keypress sobre un objeto texto o change que se produce sobre dropdown.*

## **CONCEPTOS BASICOS**





*Definición: MENSAJE*

*Constituye la comunicación que se establece entre objetos para intercambiar parámetros sean de entrada, salida o ambos. Mediante parámetros se establece un esquema de colaboración entre objetos*

## CONCEPTOS BASICOS





## *Definición: PROPIEDAD*

*Se le denomina también atributo o dato de la clase. Constituye el contenedor de un valor de un tipo de dato determinado. Un método puede variar ese contenido.*

## **CONCEPTOS BASICOS**





## *Definición: PROPIEDAD*

*Se le denomina también atributo o dato de la clase. Constituye el contenedor de un valor de un tipo de dato determinado. Un método puede variar ese contenido.*

## **CONCEPTOS BASICOS**





## *Definición: ABSTRACCION*

*Representa las características esenciales de un objeto, sin tener la preocupación de las demás características que no lo son. Se preocupa más de los detalles importantes de la implementación de un objeto, separando el comportamiento esencial.*

*Abstracción es describir una entidad del mundo real, por compleja que sea, y utilizar dicha descripción en un programa. Ejemplo de abstraccionismo es la de un motor. Este motor puede ser muy genérico y comportarse de tal forma cuando es implementado por un motor a explosión, eléctrico o a vapor.*

## **CONCEPTOS BASICOS**





## *Definición: ENCAPSULAMIENTO*

*Significa la conjunción de elementos que abstractamente se consideran pertenecientes a una entidad, con mayor cohesión entre los componentes de la misma. No se debe confundir con el principio de ocultamiento*

## **CONCEPTOS BASICOS**





*Definición:* **MODULARIDAD**

*Tiene como objetivo descomponer una aplicación en otras más pequeñas (módulos) que sea lo más acoplada posible a sí misma (independiente)*

## **CONCEPTOS BASICOS**





## *Definición: OCULTACION*

*Es el sistema de aislamiento generalmente de las propiedades de un objeto para proteger su integridad y el acceso desautorizado que pueda modificar su contenido. Sólo los métodos internos o pertenecientes a la clase pueden modificar dichas propiedades.*

## **CONCEPTOS BASICOS**





## Definición: **POLIFORMISMO**

*Se refiere a la capacidad de los objetos de comportarse de acuerdo a la funcionalidad requerida. Es decir, establecer diferentes comportamientos para los métodos, por ejemplo, del objeto, que implementan diversas funcionalidades de acuerdo con parámetros recibidos a través de los mismos. Por ejemplo, se puede realizar una suma de enteros que devuelva enteros y reciba parámetros enteros, o una suma de valores flotantes donde se reciban parámetros flotantes y devuelva un valor flotante. Ambos métodos pueden llamarse igual pero se diferencian por los parámetros que reciben.*

## CONCEPTOS BASICOS





## REGLAS DE ALCANCE

*Se entiende por alcance, ámbito o scope de una variable, la parte del programa donde la variable es accesible. Veremos los tipos que existen.*

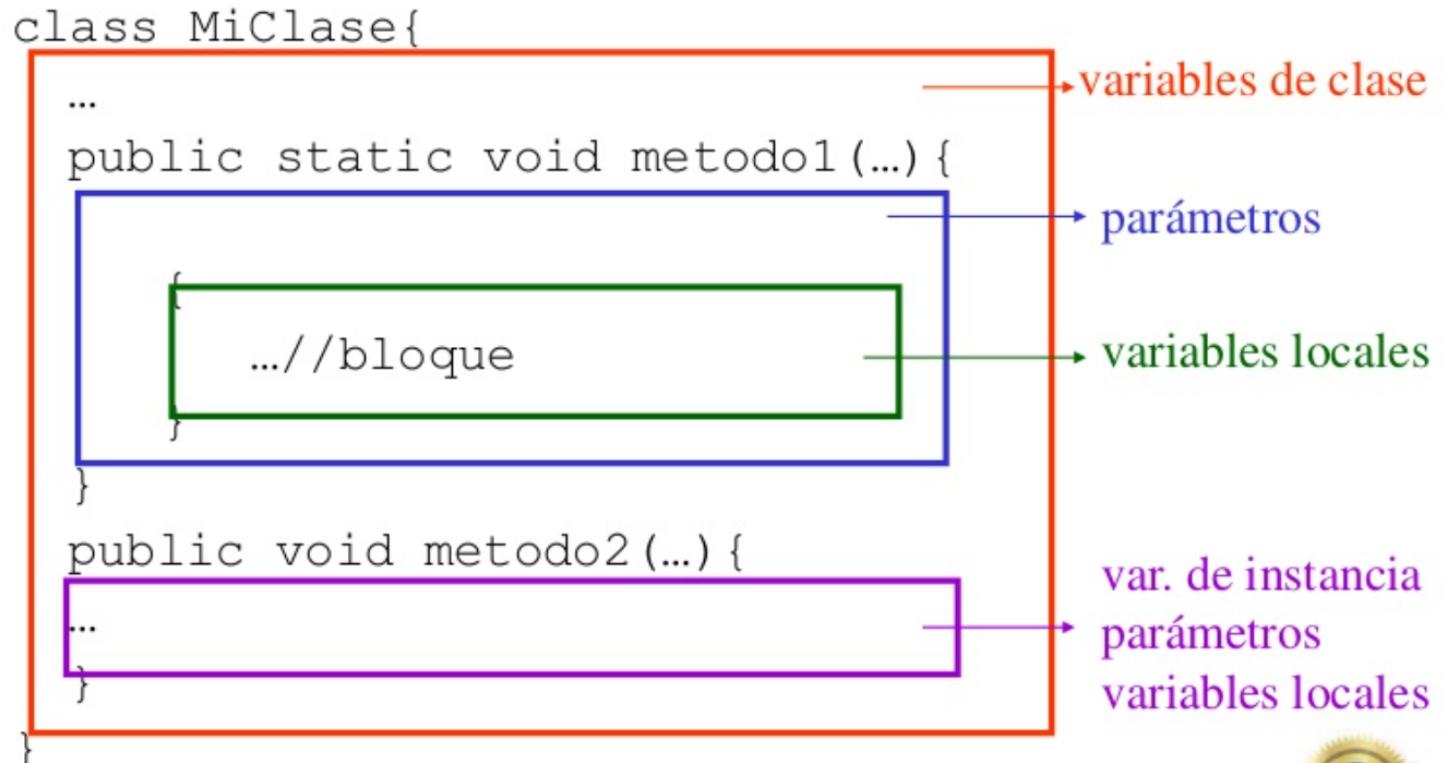
*Al igual que C/C ++, en Java, todos los identificadores tienen un ámbito léxico (o estático). Los programas de Java están organizados en forma de clases. Cada clase es parte de un paquete. Las reglas de ámbito de Java se pueden cubrir en las siguientes categorías.*

## REGLAS DE ALCANCE





# Reglas de alcance



El lenguaje de programación Java





## PARAMETROS

*En los lenguajes de programación suelen existir dos formas de pasar los parámetros a los métodos. Parámetros por valor (dónde se realiza una copia de las variables) o parámetros por referencia (dónde se pasa una referencia a la variable original).*

## PARAMETROS





## POR VALOR

*Lo primero que tenemos que ver es que para los datos primitivos en Java se realiza claramente una copia.*

```
public void metodo(int p) {  
    p=3;  
}
```

```
int p1=2;  
metodo(p1);
```

```
System.out.println(p1); //p1 = 2
```

## PARAMETROS





## POR REFERENCIA

*Entonces al tomar un objeto como parámetro. Lo que sucede al manejar los objetos en Java es que las variables mantienen una referencia al objeto, por lo tanto cuando pasamos un objeto como parámetro se está realizando una copia de la referencia. Así tenemos dos variables diferentes apuntando al mismo objeto.*

## PARAMETROS





# PARAMETROS

## POR REFERENCIA

Creamos una clase básica llamada MiClase:

```
public class MiClase {  
    public int valor;  
}
```

Y ahora un método que modifica ese valor:

```
public static void metodo_referencia(MiClase m) {  
    m.valor =3;  
}
```

Veamos como se pasa por valor, aunque parece que hay una referencia:

```
MiClase m1 = new MiClase();  
m1.valor = 2;  
System.out.println(m1.valor); // Devuelve 2  
metodo_referencia(m1);  
System.out.println(m1.valor); // Devuelve 3
```





## IN LINE

*Se conocen como funciones inline a las funciones que, al compilar, no son llamadas en el código objeto, sino insertadas en la sección del código donde se las llame.*

*Cuando escribimos el nombre de una función en un programa, por así decirlo "llamamos" a dicha función. Por lo tanto, nuestro programa ejecuta la función y sigue la ejecución del código. Esto sucede cuando la función se declara de una manera normal, pero cuando creamos la función y le especificamos que es de tipo inline, lo que sucede dentro de nuestro código es que, al ser compilado, la función se inserta como una copia en el mismo código fuente, en vez de llamar a la función*

## FUNCIONES





## SOBRECARGA DE FUNCIONES

*Sobrecarga es la capacidad de un lenguaje de programación, que permite nombrar con el mismo identificador diferentes variables u operaciones.*

*En programación orientada a objetos la sobrecarga se refiere a la posibilidad de tener dos o más funciones con el mismo nombre pero funcionalidad diferente. Es decir, dos o más funciones con el mismo nombre realizan acciones diferentes. El compilador usará una u otra dependiendo de los parámetros usados. A esto se llama también sobrecarga de funciones.*

## SOBRE CARGA DE FUNCIONES





## USO DE PLANTILLAS

*Las plantillas (en inglés: Templates) son una de las variadas utilidades de las cuales disponemos al trabajar con IDEs, en general una plantilla es un texto o código reutilizable que usualmente contendrá ciertas secciones de contenido variable las cuales se concretaran según ciertos valores o el contexto en el que se utiliza.*

*El objetivo de usar plantillas es no perder el tiempo escribiendo código, comentarios, etc. que ya pueden ser incluidos o generados automáticamente por el propio IDE.*

*Podemos diferenciar dos tipos de plantillas: las de archivo y las de código.*

## USO DE PLANTILLAS





## USO DE PLANTILLAS

```
NuevaClase.java X
1 /**
2 *
3 */
4 package mipaquete;
5
6 /**
7 * @author Dark[byte]
8 *
9 */
10 public class NuevaClase
11 {
12
13 }
```

*Archivo*

**USO DE PLANTILLAS**





## LIBRERIAS

*En Java y en varios lenguajes de programación más, existe el concepto de librerías. Una librería en Java se puede entender como un conjunto de clases, que poseen una serie de métodos y atributos. Lo realmente interesante de estas librerías para Java es que facilitan muchas operaciones. De una forma más completa, las librerías en Java nos permiten reutilizar código, es decir que podemos hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar nosotros mismos esas funcionalidades.*

## LIBRERIAS





# 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones** creada para resolver una necesidad específica.



## 1 GOOGLE GUAVA



### Mejora del flujo de trabajo

destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

## 3 MPANDROIDCHART

Crea **gráficos** de líneas, barras, radares y burbujas para Android.

## 5 MOCKITO

Librería de código abierto para **simular pruebas unitarias**.

## 2 RXJAVA

Crea aplicaciones para **reaccionar** a los flujos de datos en tiempo real.

## 4 FASTJSON

Convierte **objetos Java** en su representación JSON y viceversa.

## 6 JUNIT

Esta es la librería más **usada para pruebas**.

Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/cursos/java](http://ed.team/cursos/java)



## LIBRERIAS





*¿preguntas?*



# CONCLUSIONES



Politécnico  
Internacional



# *PROGRAMACION ORIENTADA A OBJETOS*



“Controlar la complejidad  
es la esencia de la programación.”

*Brian Kernigan*



## ESPACIOS DE FORMACIÓN

- 
- 1 Martes y Jueves
  - 2 Break

Según Plataforma  
Por acordar.

## A TENER EN CUENTA



1. Cerrar los micrófonos y desactivar las cámaras.



2. Solicitar la palabra mediante el chat.



3. Preguntar



# AGENDA

- *Conceptos Básicos de Java*
- *Innova Colombia*
- *Conclusiones*



# RAMAS DE LA PROGRAMACIÓN



## DESARROLLO WEB

La web es una interfaz universal, las aplicaciones que creas funcionan en cualquier dispositivo.

## INTELIGENCIA ARTIFICIAL

Diseña modelos para que las computadoras aprendan y tomen decisiones por sí solas (con Machine Learning).



## REALIDAD VIRTUAL Y AUMENTADA

Sumerge al usuario en entornos virtuales que simulen la realidad.



## VIDEOJUEGOS

Trabaja creando tu videojuego favorito para cualquier plataforma.



## SEGURIDAD INFORMÁTICA

Rompe la seguridad de un sistema o protégelo de ataques de terceros.



## DESARROLLO MÓVIL

Desarrolla aplicaciones exclusivas para Android o iOS.

Aprende a programar en cualquier lenguaje (primer curso gratis) en:  
 [ed.team/programacion](https://ed.team/programacion)



## ¿QUÉ ÁREA DE LA PROGRAMACIÓN ELEGIR?

- Para comenzar en el mundo de la programación **debes aprender a programar desde cero.**
- Luego puedes ir a **cualquier lenguaje de programación**, dependerá mucho de la rama en la que quieras especializarte, **estas son algunas que puedes seguir:**

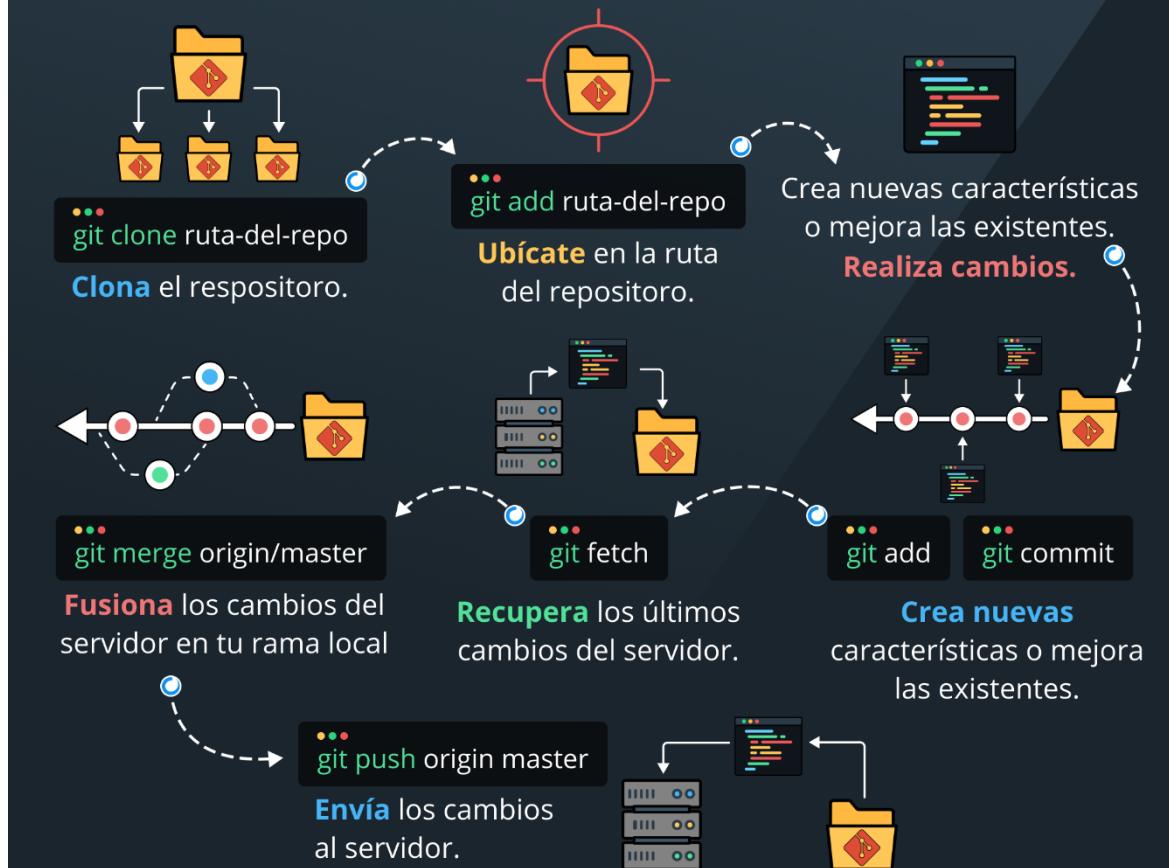


Comienza tu carrera como programador (primer curso gratis) en:

👉 [ed.team/programacion](https://ed.team/programacion)



## ¿CÓMO CONTRIBUIR EN UN PROYECTO CON GIT?



Aprende a controlar las versiones de tus proyectos:

👉 [ed.team/cursos/git](http://ed.team/cursos/git)



## TÉRMINOS DE git QUE DEBES CONOCER

### REPOSITORY/REPO

Base de datos donde se almacena el historial del código.

### COMMIT

Registro de uno o varios cambios hechos en el repositorio.

### STAGE

Lista de archivos que se usarán para un commit.

### FORK

Copia de un repositorio.

### BRANCH / RAMA

Entorno o espacio de trabajo independiente en Git.

### MASTER

Rama principal de un repositorio.

### CHECKOUT

Es la acción de moverse entre diferentes ramas.

### MERGE

Combina dos o más ramas en una.

Si eres programador es **obligatorio** aprender GIT. Domínalo en:

[ed.team/cursos/git](https://ed.team/cursos/git)

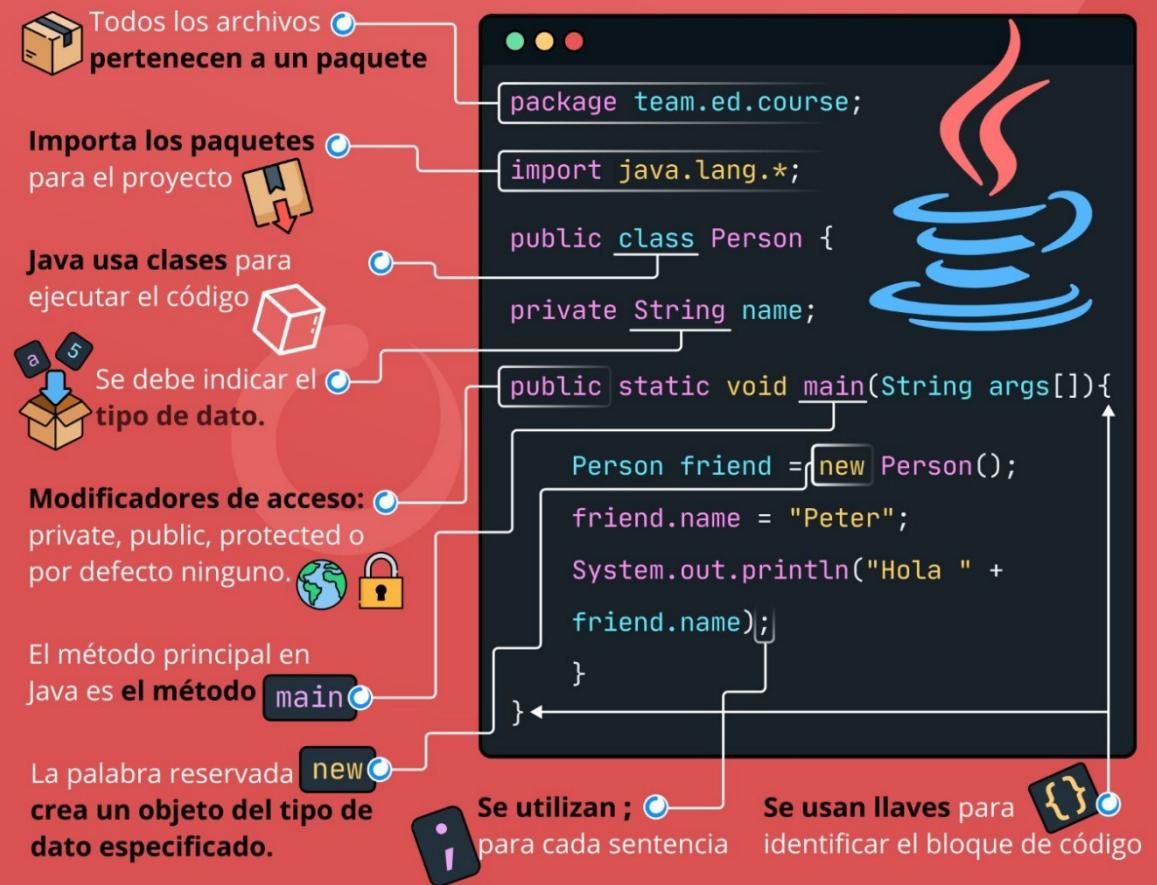


Java™





# SINTAXIS BÁSICA DE JAVA



Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/java](http://ed.team/java)



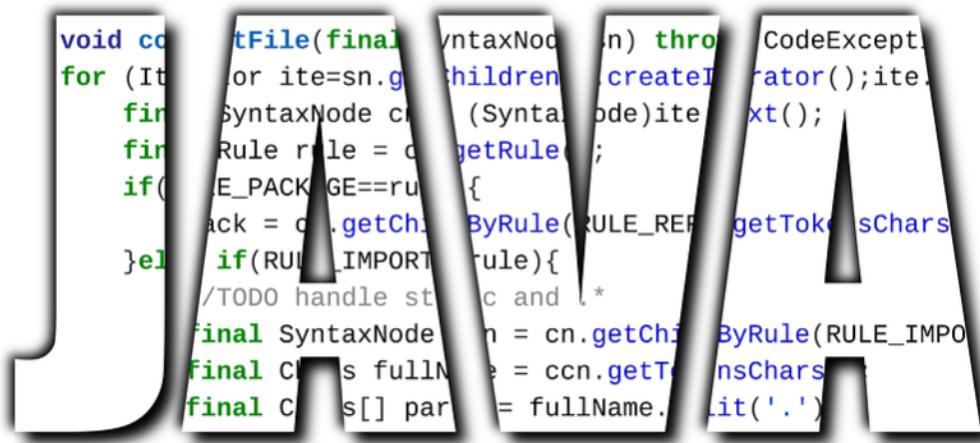
# *Introducción al Lenguaje De Programación JAVA*



- *Porque se creo JAVA*
- *Historial de Versiones de JAVA*
- *Características de JAVA*



# ¿Por qué se creo Java?



Con el inicio de Internet en 1991 el equipo “**Green Team**” teniendo como líder del equipo a **James Gosling** de SUN Microsystems crearon un Lenguaje que inicialmente se llamada OAK.

## Filosofía

“Escribe una vez, ejecuta desde cualquier lado”



# JAMES GOSLING

## Ingeniero **SUN - GOOGLE**



GOSLING es reconocido como el creador del lenguaje de programación Java. Realizó el diseño original y la implementación del compilador original y la máquina virtual Java, por lo que fue elegido miembro de la Academia Nacional de Ingeniería de Estados Unidos (NAE).

En el 2015 recibió la medalla John von Neumann de la IEEE por sus contribuciones al desarrollo informático



# Historial de Versiones

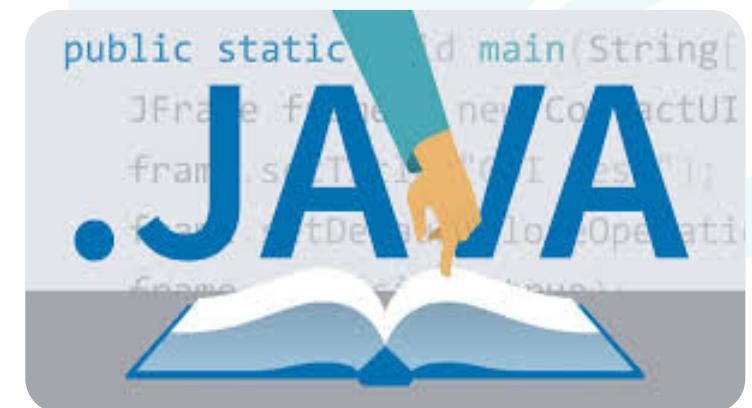
- Junio de 1991 – Se inició el proyecto de lenguaje Java
- JDK 1.0 – enero de 1996
- JDK 1.1 – febrero de 1997
- J2SE 1.2 – diciembre de 1998
- J2SE 1.3 – mayo de 2000
- J2SE 1.4 – febrero de 2002
- J2SE 5.0 – septiembre de 2004
- Java SE 6 – diciembre de 2006
- Java SE 7 – julio de 2011
- Java SE 8 – 18 de marzo de 2014
- Java SE 9 – julio de 2017



# CARACTERÍSTICAS DE JAVA

## INDEPENDIENTE DE LA PLATAFORMA

Java se creó con la filosofía de “escribe una vez, ejecuta en cualquier lado” (WORA). El código de Java (código Java puro y bibliotecas) que escriba en una plataforma (sistema operativo) se ejecutará en otras plataformas sin modificaciones.





# CARACTERÍSTICAS DE JAVA

## INDEPENDIENTE DE LA PLATAFORMA

Para ejecutar Java, se utiliza una máquina abstracta llamada JAVA VIRTUAL MACHINE (JVM). La JVM ejecuta el BYTECODE de Java. Entonces, la CPU ejecuta la JVM.

Dado que todas las JVM funcionan exactamente igual, el mismo código también funciona en otros sistemas operativos, lo que hace que Java sea independiente de la plataforma.





# CARACTERÍSTICAS DE JAVA

## LENGUAJE ORIENTADO A OBJETOS

*Hay diferentes estilos de programación. El enfoque orientado a objetos es uno de los estilos de programación más popular.*

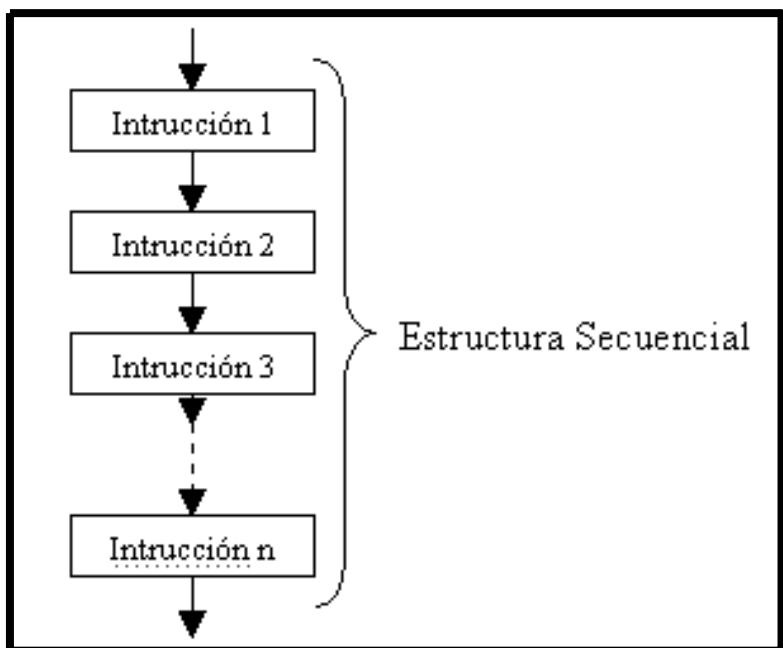
*En la programación orientada a objetos, un problema complejo se divide en conjuntos más pequeños mediante la creación de objetos. Esto hace que el código sea reutilizable, tenga beneficios de diseño y haga que el código sea más fácil de mantener.*





# CARACTERÍSTICAS DE JAVA

## PROGRAMACION ESTRUCTURADA





# ¿QUÉ ES LA PROGRAMACIÓN ORIENTADA A OBJETOS?

- Los objetos se crean a partir de una plantilla llamada clase. Cada objeto es una instancia de su clase.



INSTANCIACIÓN

OBJETO

Es un paradigma de programación que organiza las funciones en entidades llamadas objetos.



- Los objetos tienen datos (atributos) y funcionalidades (métodos).

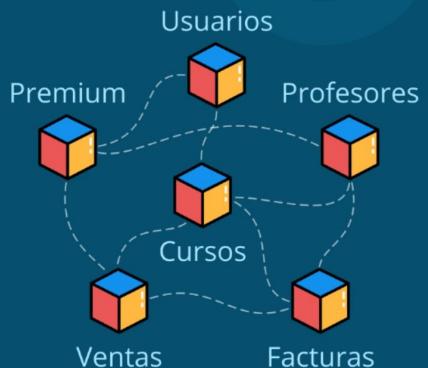
MÉTODOS

Editar perfil  
Iniciar sesión  
Cerrar sesión  
Cambiar contraseña  
Pasar a premium

ATRIBUTOS

Nombres  
Apellidos  
Correo  
Contraseña  
Premium

- En una aplicación los objetos están separados pero se comunican entre ellos.



Puedes programar con este paradigma en la mayoría de lenguajes.



Aprende a crear aplicaciones usando la POO en:

[ed.team/cursos](https://ed.team/cursos)

EDteam





# CARACTERÍSTICAS DE JAVA

## SEGURIDAD

- *Proporciona una plataforma segura para desarrollar y ejecutar aplicaciones.*
- *Administración automática de memoria, reduce la corrupción de la memoria y vulnerabilidades.*
- *Proporciona comunicación segura al proteger la integridad y privacidad de los datos transmitidos*





# CARACTERÍSTICAS DE JAVA

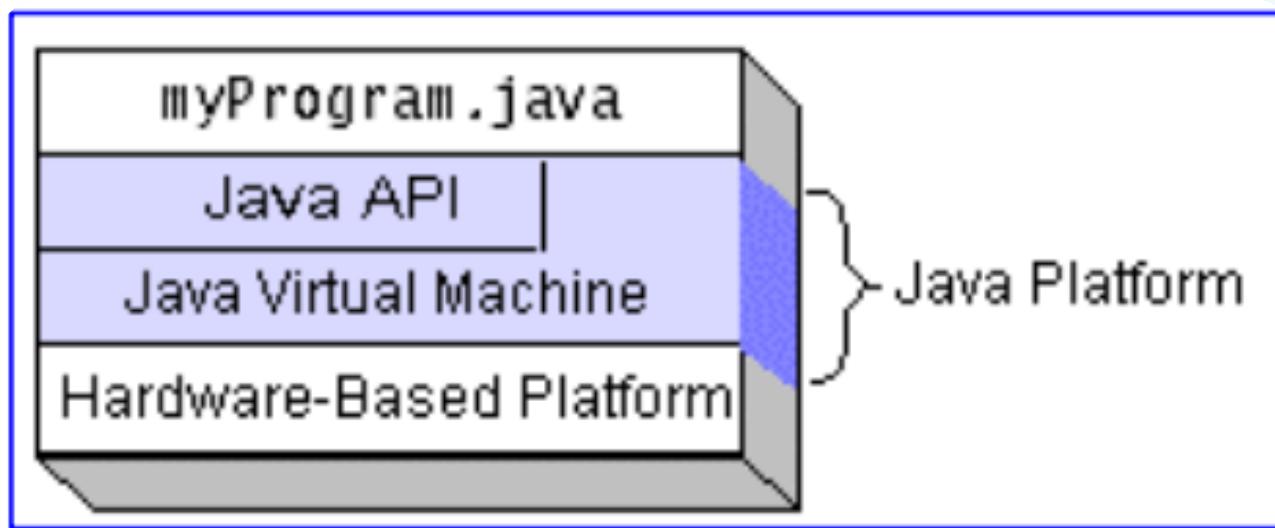
## BIBLIOTECAS

- **JAVA.LANG** – Para funciones avanzadas de cadenas, matrices, etc.
- **JAVA.UTIL** – Para estructuras de datos, expresiones regulares, funciones de fecha y hora, etc.
- **JAVA.IO** – Para archivos E/S, manejo de excepciones, etc.



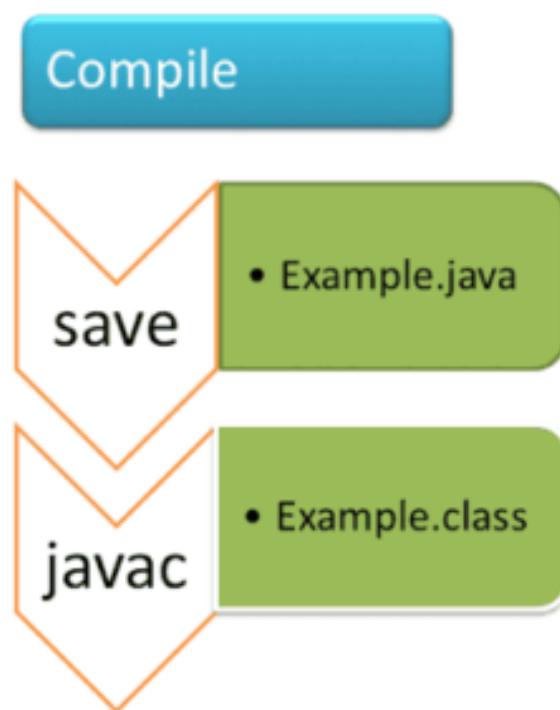


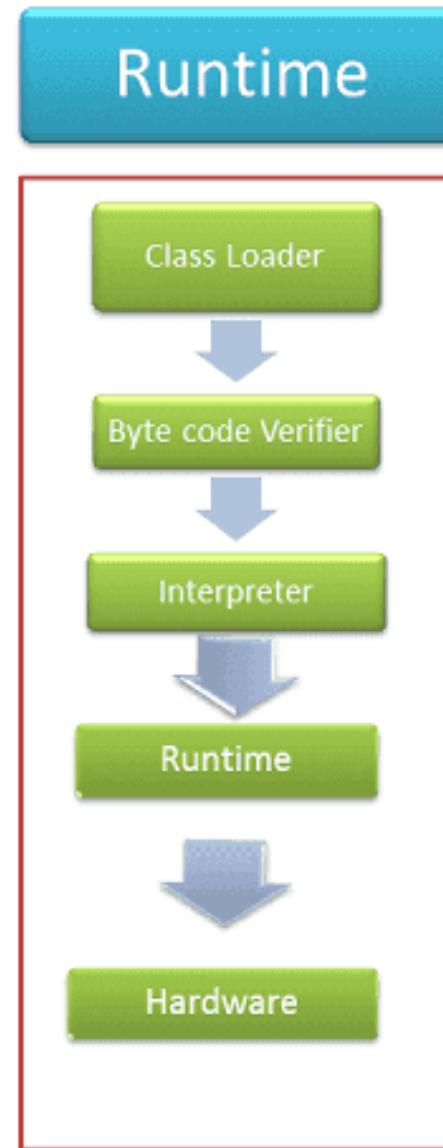
## ARQUITECTURA JAVA





*JAVA RUNTIME ENVIRONMENT o JRE es un conjunto de utilidades que permite la ejecución de programas JAVA*



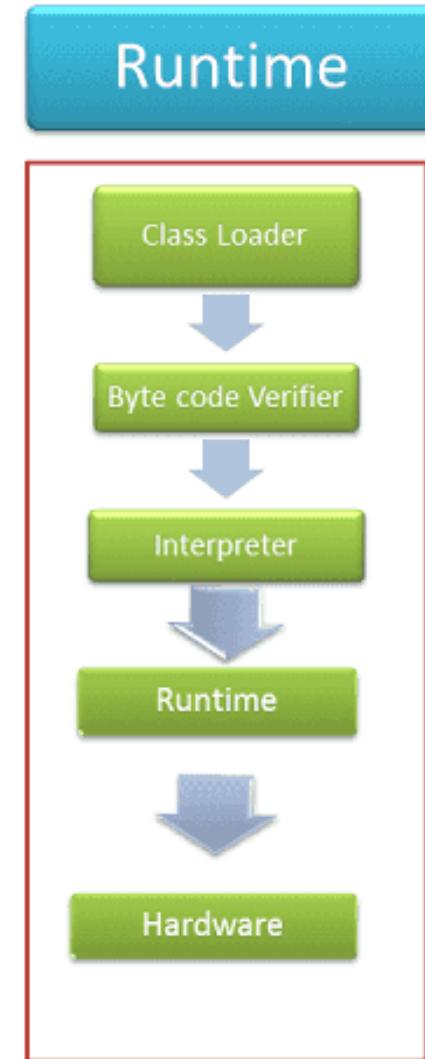




# JRE

## JAVA RUNTIME ENVIRONMENT

**CLASS LOADER:** El Class Loader carga todas las clases necesarias para la ejecución de un programa. Proporciona seguridad al separar los espacios de nombres del sistema de archivos local, de los importados a través de la red. Estos archivos se cargan desde un disco duro, una red o desde otras fuentes.

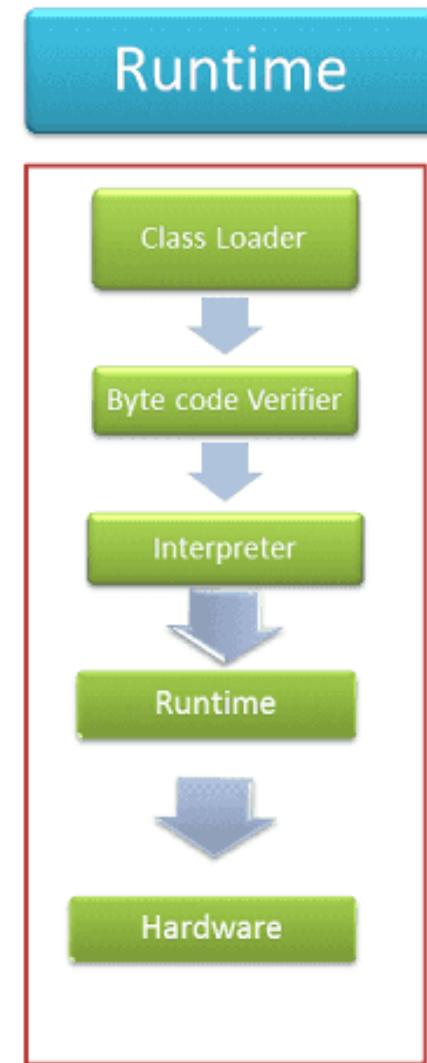




# JRE

## JAVA RUNTIME ENVIRONMENT

**BYTE CODE VERIFIER:** La JVM coloca el código en Byte Code Verifier que verifica el formato y busca un código ilegal. El código ilegal, por ejemplo, es un código que infringe los derechos de acceso en los objetos o viola la implementación de los punteros.



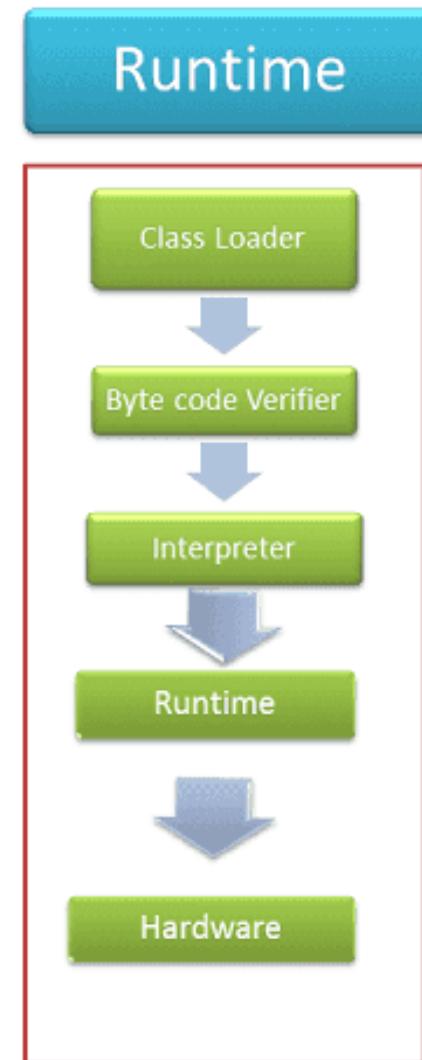


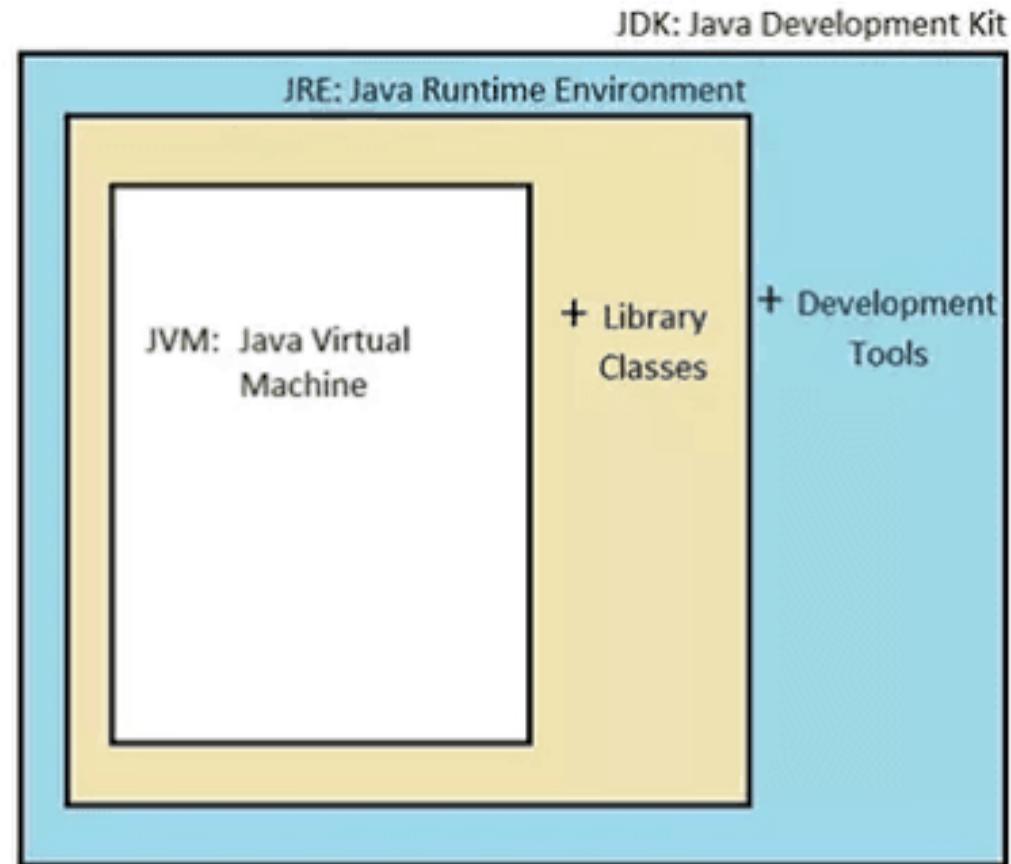
# JRE

## JAVA RUNTIME ENVIRONMENT

*INTPRETER: En tiempo de ejecución, el intérprete carga, verifica y ejecuta el Byte Code. El intérprete tiene las siguientes dos funciones:*

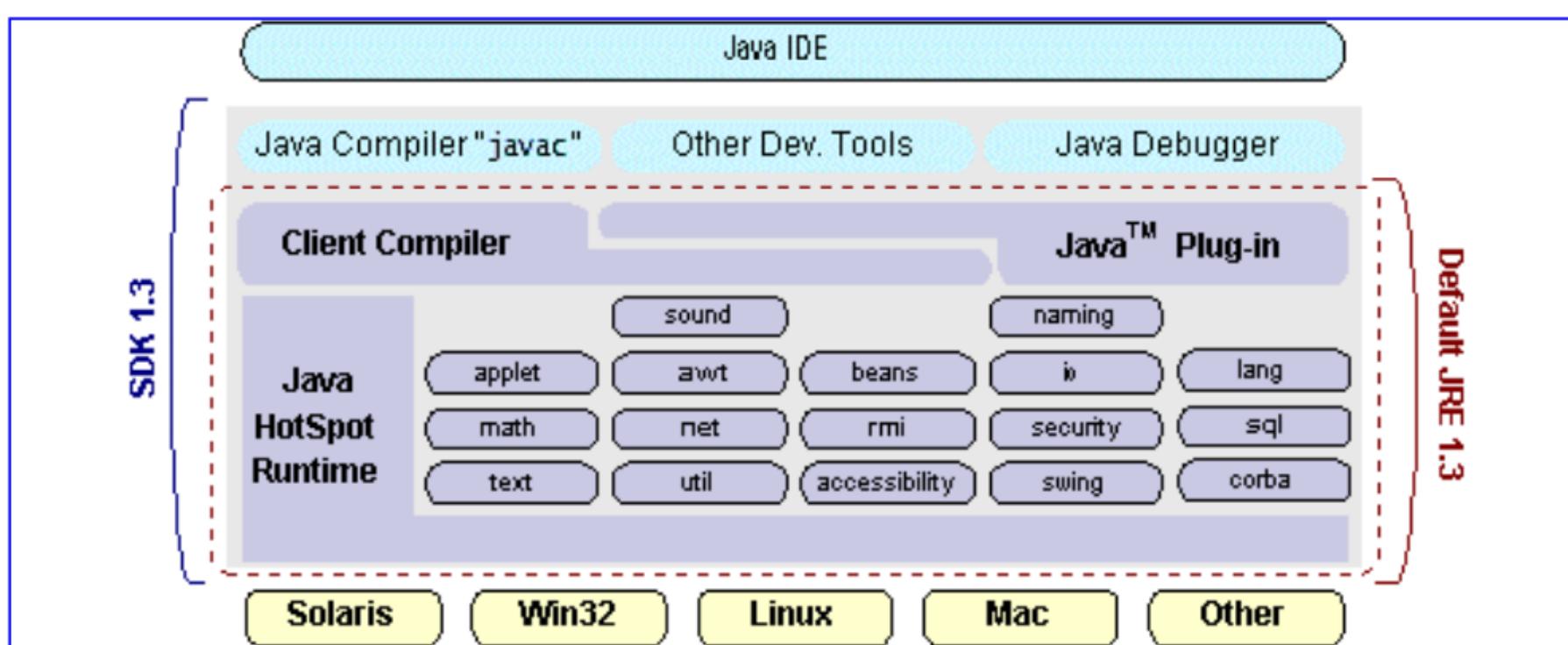
- *Ejecute el ByteCode*
- *Hace llamadas apropiadas al hardware subyacente*

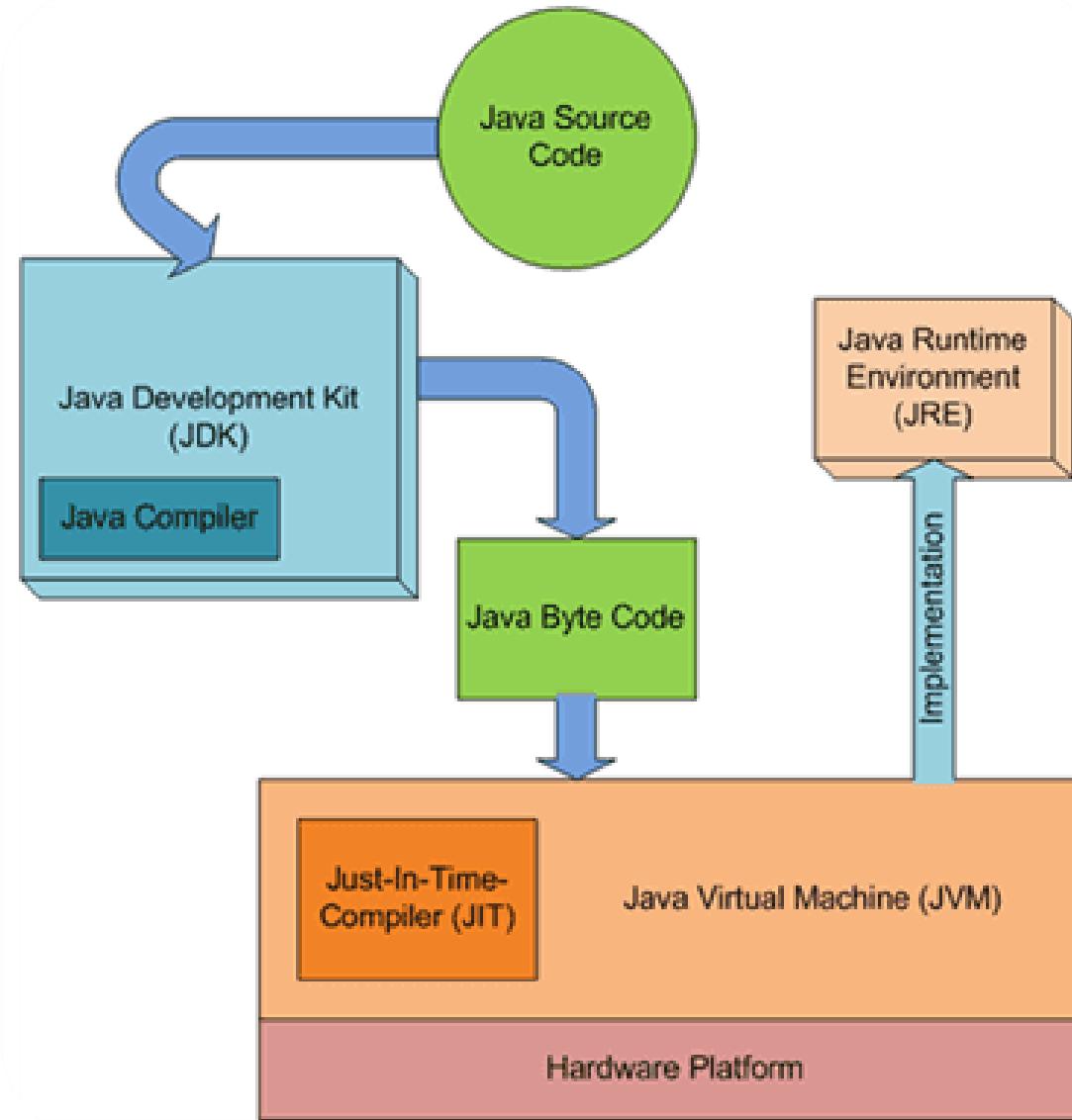




*JDK = JRE + Development Tools*

*JRE = JVM + Library Classes*





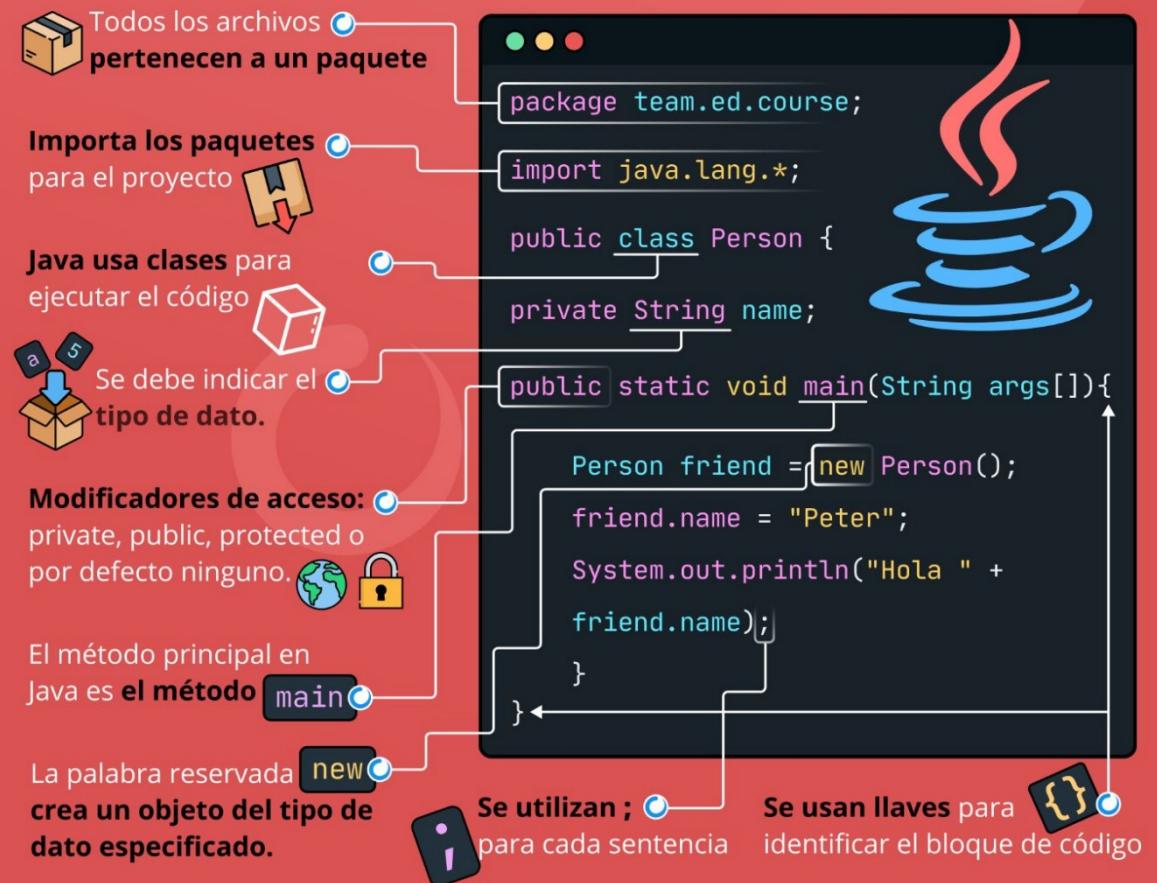


Java™





# SINTAXIS BÁSICA DE JAVA



Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/java](http://ed.team/java)



# 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones creada para resolver una necesidad específica.**



## 1 GOOGLE GUAVA



**Mejora del flujo de trabajo**  
destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

## 2 RXJAVA



**Crea aplicaciones para reaccionar** a los flujos de datos en tiempo real.

## 3 MPANDROIDCHART



**Crea gráficos** de líneas, barras, radares y burbujas para Android.

## 4 FASTJSON



**Convierte objetos** Java en su representación JSON y viceversa.

## 5 MOCKITO



Librería de código abierto para **simular pruebas unitarias**.

## 6 JUNIT



Esta es la librería más **usada para pruebas**.

Aprende Java desde cero hasta nivel Jedi en:

[ed.team/cursos/java](http://ed.team/cursos/java)



Politécnico  
Internacional





Politécnico  
Internacional





Politécnico  
Internacional





## TIPOS DE LENGUAJE



- *Lenguajes Tipo Estático*
- *Lenguajes Tipo Dinámico*

# LENGUAJE TIPADO ESTÁTICO

```
void co    tFile(final Synta    on) throws CodeExcept
for (It    or ite=sn.getChil    xception {
    fir    SyntaxNode chil    dren(Synta
    fir    Rule rule = c    xNode) {
        if(E_PACK == rule
            pack = c.getChil
        }el
            if(RULE_IMPORT
                /TODO handle st
            final SyntaxNode
            final C s fullN
            final C s[] par
```



*Cada variable y tipo de expresión ya se conoce en tiempo de compilación. Una vez que se declara una variable es de un cierto tipo de datos, no puede contener valores de otros tipos de datos.*

Se dice de un lenguaje de programación que usa un tipado estático cuando la comprobación de tipificación se realiza durante la compilación, y no durante la ejecución



# LENGUAJE TIPADO DINÁMICO



*Estos lenguajes pueden recibir diferentes tipos de datos a lo largo del tiempo. Ejemplo: Ruby, Python*

*Un lenguaje de programación **dinámico** es un lenguaje de programación en el que las operaciones realizadas en tiempo de **compilación** pueden realizarse en tiempo de ejecución*



# TIPADO ESTATICO **JAVA**

**JAVA** está tipado **ESTÁTICAMENTE** y es fuertemente tipado porque en Java, cada tipo de datos (como entero, carácter, hexadecimal, decimal empaquetado, etc.) está predefinido como parte del lenguaje de programación y todas las constantes o variables definidas para un programa dado debe describirse con uno de los tipos de datos.



*PRIMITIVOS*  
&  
*OBJETO*

# CATEGORIAS DE DATOS

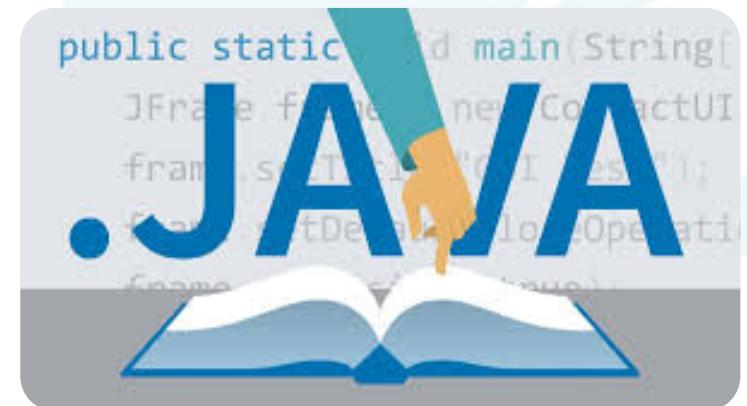




# CATEGORIAS DE DATOS

## DATOS PRIMITIVOS

*Los datos primitivos son solo valores únicos; ellos no tienen capacidades especiales. Java soporta 8 tipos de datos primitivos*





# CATEGORIAS DE DATOS

## DATOS PRIMITIVOS

**NUMÉRICOS ENTEROS:** Son los tipos *byte*, *short*, *int* y *long*. Los 4 representan números enteros con signo.

**CARÁCTER:** El tipo *CHAR* representa un carácter codificado en el sistema unicode.





# CATEGORIAS DE DATOS

## DATOS PRIMITIVOS

### NUMÉRICO

*DECIMAL:* Los tipos **FLOAT** y **DOUBLE** representan números decimales en coma flotante.

*LÓGICOS:* El tipo **BOOLEAN** es el tipo de dato lógico; los dos únicos posibles valores que puede representar un dato lógico son **true** y **false**. **true** y **false** son palabras reservadas de Java.





TIPO	DESCRIPCIÓN	DEFAULT	TAMAÑO	EJEMPLOS
boolean	true o false	false	1 bit	true, false
byte	entero complemento de dos	0	8 bits	100, -50
char	carácter unicode	\u0000	16 bits	'a', '\u0041', '\101', '\U'
short	entero complemento de dos	0	16 bits	10000,-20000
int	entero complemento de dos	0	32 bits	100000,-2,-1,0,1,2,-200000
long	entero complemento de dos	0	64 bits	-2L,-1L,0L,1L,2L
float	coma flotante IEEE 754	0.0	32 bits	1.23e100f, -1.23e-100f, .3ef, 3.14f
double	coma flotante IEEE 754	0.0	64 bits	1.2345e300d, -1.2345e-300f, 1e1d



## TIPO DE DATO BOOLEAN

El tipo de datos booleano representa solo un bit de información: true (verdadero) o false (falso). Los valores de tipo booleano no se convierten implícita o explícitamente (con casts) en ningún otro tipo.

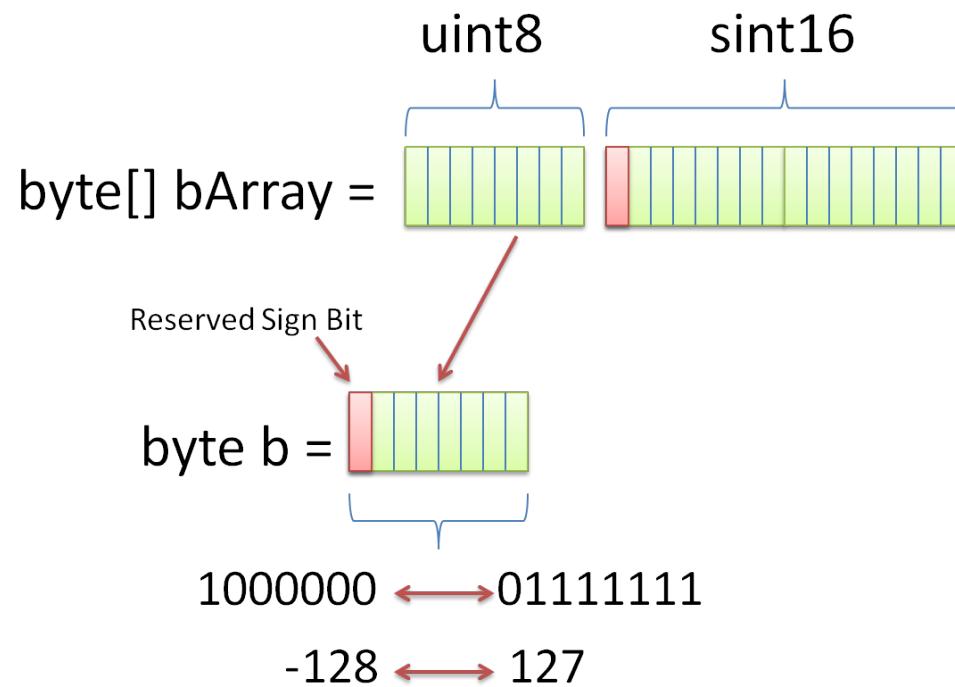
True or False:

True

False



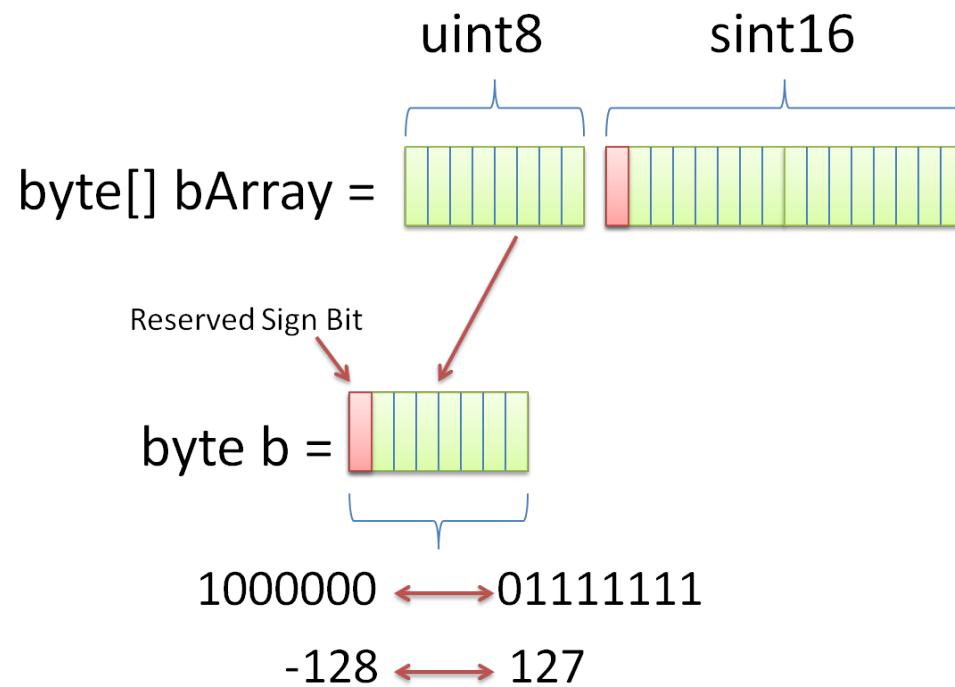
Boolean is a  
Data Type



### TIPO DE DATO BYTE

*El tipo de datos byte es un entero de 8 bits de complemento a dos (una forma de representar datos positivos y negativos en binario). El tipo de datos de byte es útil para guardar en memoria grandes arrays.*

**Tamaño:** 8 bits  
**Valor:** -128 a 127



### TIPO DE DATO BYTE

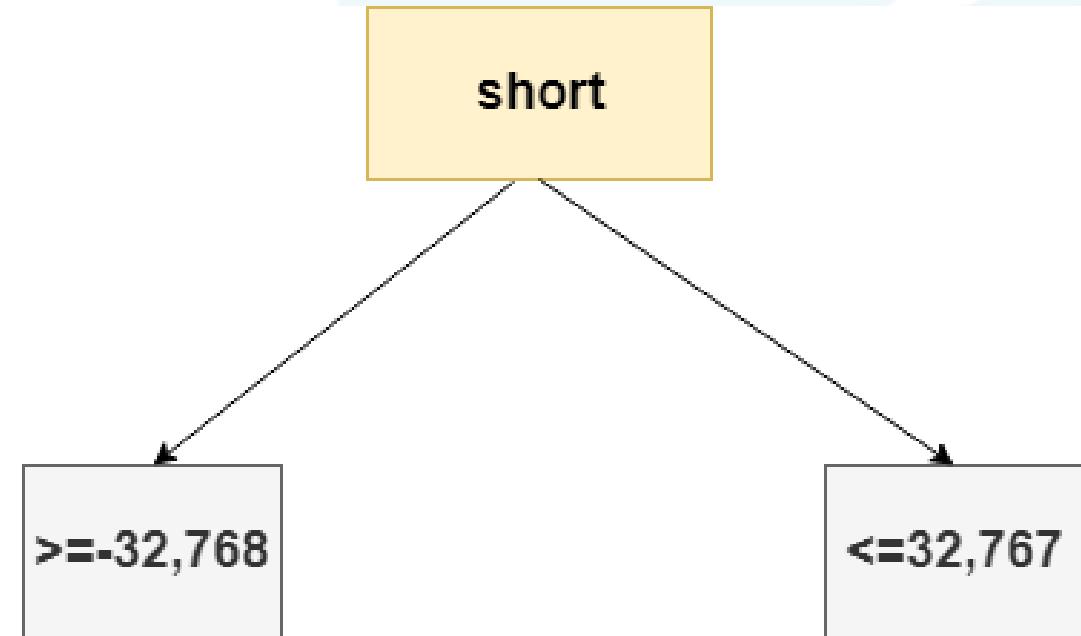
*El tipo de datos byte es un entero de 8 bits de complemento a dos (una forma de representar datos positivos y negativos en binario). El tipo de datos de byte es útil para guardar en memoria grandes arrays.*

**Tamaño:** 8 bits  
**Valor:** -128 a 127



### TIPO DE DATO SHORT

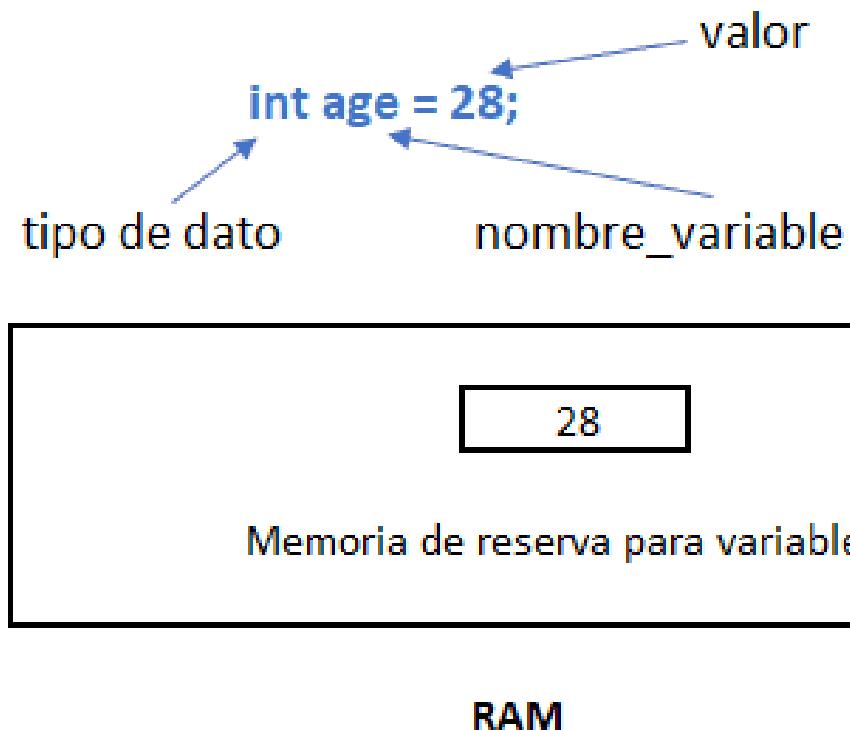
*El tipo de datos **SHORT** es un entero de complemento de dos formado por 16 bits. De forma similar al byte, use short para guardar en memoria arrays grandes, en situaciones donde el ahorro de memoria realmente importa.*





# CATEGORIAS DE DATOS

DATOS  
PRIMITIVOS



## TIPO DE DATO INT

*Es un entero de complemento de dos formado de 32 bits.*

**Tamaño:** 32 bits

**Valor:**  $(-2^{31})$  a  $(2^{31} - 1)$

*En Java SE 8 y posterior, podemos usar el tipo de datos int para representar un entero de 32 bits sin signo, que tiene un valor en el rango  $[0, 2^{32} - 1]$ . Utilice la clase INTEGER para usar el tipo de datos int como un entero sin signo.*



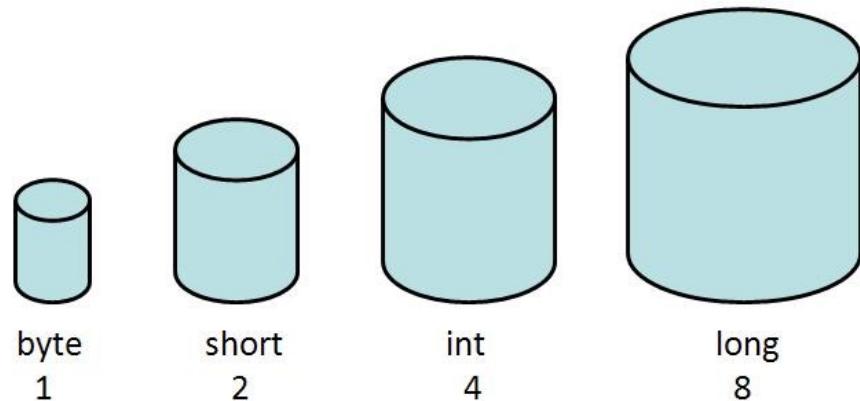
### TIPO DE DATO LONG

*El tipo de datos **LONG** es un entero de complemento de dos de 64 bits..*

**Tamaño:** 64 bit

**Valor:**  $(-2^{63})$  a  $(2^{63}-1)$

*En Java SE 8 y posteriores, puede usar el tipo de datos Long para representar un Long sin signo de 64 bits, que tiene un valor mínimo de 0 y un valor máximo de  $2^{64} - 1$ . La clase Long también contiene métodos como **COMPAREUNSIGNED**, **DIVIDEUNSIGNED**, etc. Para admitir operaciones aritméticas.*





### TIPO DE DATO FLOAT - DOUBLE

*El tipo de dato float es una coma flotante IEEE 754 de precisión simple de 32 bits. Puede usar float (en lugar double) si necesita guardar en memoria grandes arrays de números de coma flotante.*

*Tamaño: 32 bits*

*Sufijo: F/f Ejemplo: 9.8f*

*El tipo de dato double es una coma flotante IEEE 754 de 64 bits de doble precisión. Para valores decimales, este tipo de datos generalmente es la opción predeterminada.*

64bit = double, double precision



32bit = float, single precision



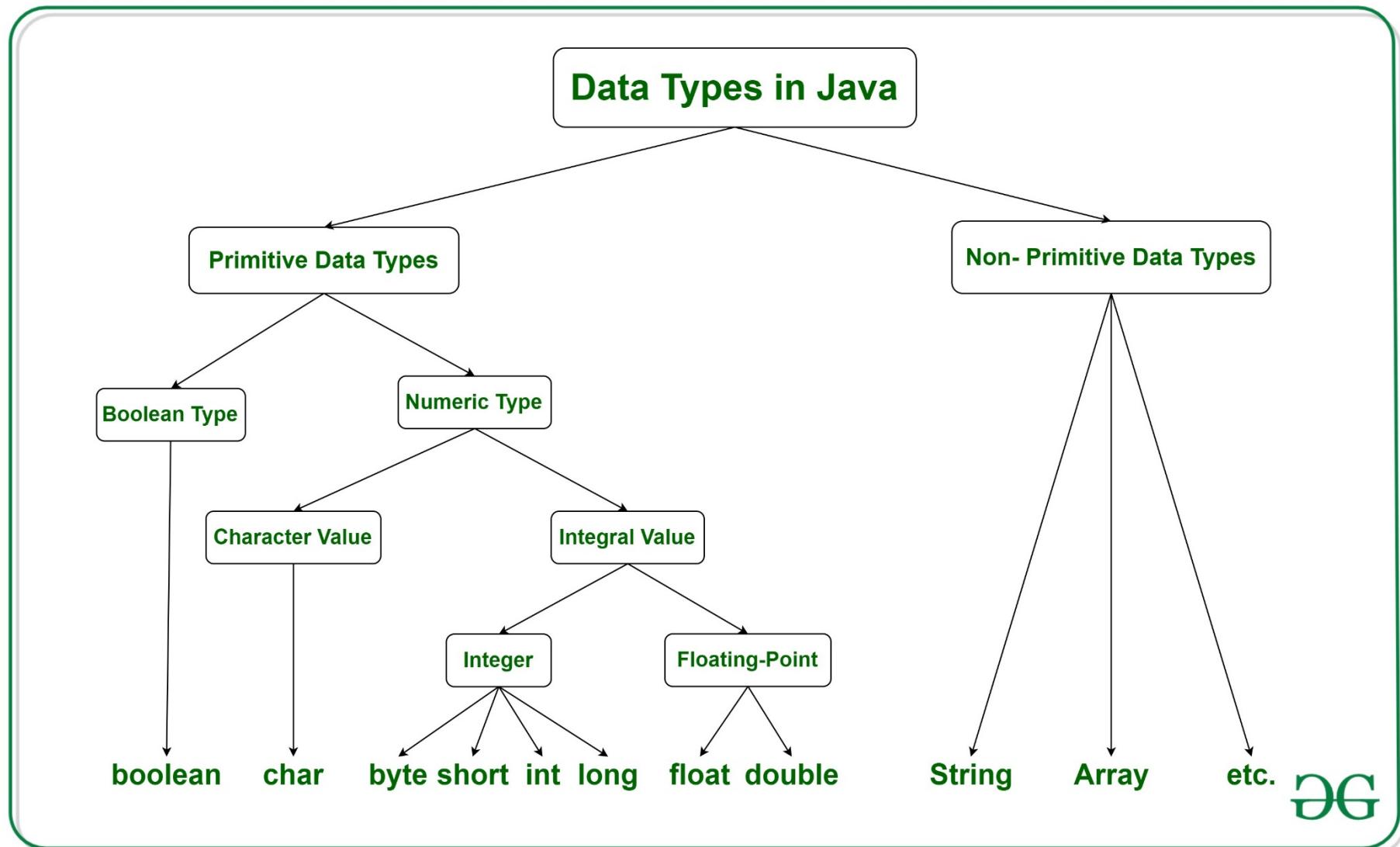
Signed bit

Exponent

Significand

16bit = half, half precision



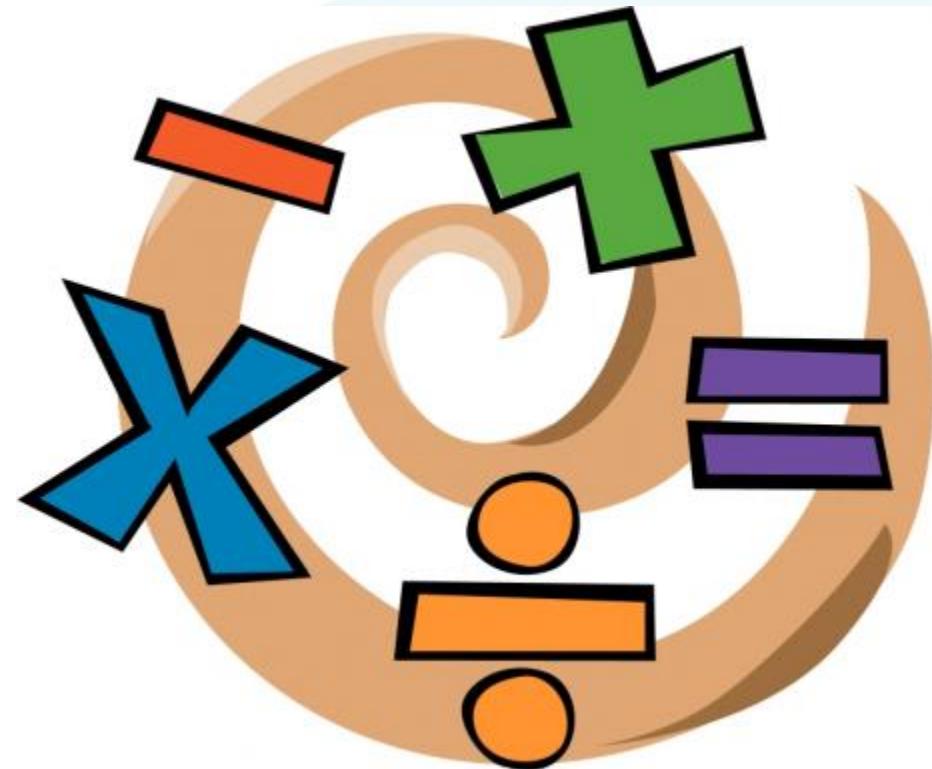




## CLASIFICACION OPERADORES JAVA

*Java proporciona muchos tipos de operadores que se pueden usar según la necesidad. Se clasifican según la funcionalidad que brindan. Algunos de los tipos son los siguientes:*

- Operadores aritméticos
- Operadores unarios
- Operador de asignación





# OPERADORES ARITMÉTICOS

DATOS  
PRIMITIVOS

JAVA + OPERADORES  
ARITMÉTICOS

```
// Programa Java para ilustrar
// operadores aritméticos
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        String x = "Thank", y = "You";

        // Operador + y -
        System.out.println("a + b = "+(a + b));
        System.out.println("a - b = "+(a - b));

        // El operador + si se usa con strings
        // concatena las cadenas dadas.
        System.out.println("x + y = "+x + y);

        // Operador * y /
        System.out.println("a * b = "+(a * b));
        System.out.println("a / b = "+(a / b));

        // operador de módulo da el resto
        // de dividir el primer operando con el segundo
        System.out.println("a % b = "+(a % b));

        // si el denominador es 0 en la división
        // System.out.println(a/c);
        // lanzaría una java.lang.ArithmeticException

    }
}
```

*Se utilizan para realizar operaciones aritméticas simples en tipos de datos primitivos.*

\*: Multiplicación  
/: División  
%: Modulo  
+: Adición  
-: Resta



# OPERADORES UNARIOS

DATOS  
PRIMITIVOS

JAVA + OPERADORES  
UNARIOS

```
// Programa Java para ilustrar
// operadores unarios
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;

        // operador de pre-incremento
        // a = a+1 y entonces c = a;
        c = ++a;
        System.out.println("Valor de c (++a) = " + c);

        // operador de post-incremento
        // c=b entonces b=b+1 (b pasa a ser 11)
        c = b++;
        System.out.println("Valor de c (b++) = " + c);

        // operador de pre-decremento
        // d=d-1 entonces c=d
        c = --d;
        System.out.println("Valor de c (--d) = " + c);

        // operador de post-decremento
        // c=e entonces e=e-1 (e pasa a ser 39)
        c = e--;
        System.out.println("Valor de c (e--) = " + c);

        // Operador lógico not
        System.out.println("Valor de !condition = " + !condition);
    }
}
```

*Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.*

*-: Unario menos, utilizado para negar los valores.  
+: Unario más, usado para dar valores positivos. Solo se usa cuando se convierte deliberadamente un valor negativo en positivo.*

*++: Operador de incremento, utilizado para incrementar el valor en 1. Hay dos variedades de operador de incremento.  
Pre-Incremento: el valor se incrementa primero y luego se calcula el resultado.*



# OPERADORES UNARIOS

DATOS  
PRIMITIVOS

JAVA + OPERADORES  
UNARIOS

Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.

*Post-Incremento: el valor se usa por primera vez para calcular el resultado y luego se incrementa.*

*—: Operador de decremento , usado para incrementar el valor en 1. Hay dos variedades de operador de incremento.*

*Pre-Decremento: el valor se disminuye primero y luego se calcula el resultado.*

*Post-Decremento: el valor se usa por primera vez para calcular el resultado y luego se disminuye.*

*!: Operador lógico “no”, utilizado para invertir un valor booleano.*

```
// Programa Java para ilustrar
// operadores unarios
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;

        // operador de pre-incremento
        // a = a+1 y entonces c = a;
        c = ++a;
        System.out.println("Valor de c (++a) = " + c);

        // operador de post-incremento
        // c=b entonces b=b+1 (b pasa a ser 11)
        c = b++;
        System.out.println("Valor de c (b++) = " + c);

        // operador de pre-decremento
        // d=d-1 entonces c=d
        c = --d;
        System.out.println("Valor de c (--d) = " + c);

        // operador de post-decremento
        // c=e entonces e=e-1 (e pasa a ser 39)
        c = e--;
        System.out.println("Valor de c (e--) = " + c);

        // Operador lógico not
        System.out.println("Valor de !condition = " + !condition);
    }
}
```



# OPERADORES ASIGNACIÓN

DATOS  
PRIMITIVOS

JAVA + OPERADORES  
ASIGNACIÓN

*El operador de asignación se usa para asignar un valor a cualquier variable. Tiene una asociación de derecha a izquierda, es decir, el valor dado en el lado derecho del operador se asigna a la variable de la izquierda y, por lo tanto, el valor del lado derecho debe declararse antes de usarlo o debe ser una constante.*

*El formato general del operador de asignación es,  
[java]variable = valor;[/java]*

```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);

        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```



```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

# OPERADORES ASIGNACIÓN

**DATOS  
PRIMITIVOS**

## JAVA + OPERADORES ASIGNACIÓN

*En muchos casos, el operador de asignación se puede combinar con otros operadores para construir una versión más corta de la declaración llamada Declaración Compuesta (Compound Statement). Por ejemplo, en lugar de `a = a + 5`, podemos escribir `a += 5`.*

```
int a = 5;
a += 5; // a = a + 5;
```



# OPERADORES ASIGNACIÓN

DATOS  
PRIMITIVOS

JAVA + OPERADORES  
ASIGNACIÓN

```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

+ = , para sumar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

- = , para restar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

• \*= , para multiplicar el operando izquierdo con el operando derecho y luego asignándolo a la variable de la izquierda.



# OPERADORES ASIGNACIÓN

DATOS  
PRIMITIVOS

JAVA + OPERADORES  
ASIGNACIÓN

```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

*/ = , para dividir el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*

*^ = , para aumentar la potencia del operando izquierdo al operando derecho y asignarlo a la variable de la izquierda.*

*% = , para asignar el módulo del operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*



## PALABRAS RESERVADAS



En los lenguajes de los programación, identificadores (como su nombre lo indica) se utilizan con fines de identificación. En Java, un identificador puede ser un nombre de clase, un nombre de método o un nombre de variable.



# PALABRAS RESERVADAS

```
public class Test
{
    public static void main(String[] args)
    {
        int a = 28;
    }
}
```



## IDENTIFICADORES

**TEST:** nombre de clase.

**MAIN:** nombre del método.

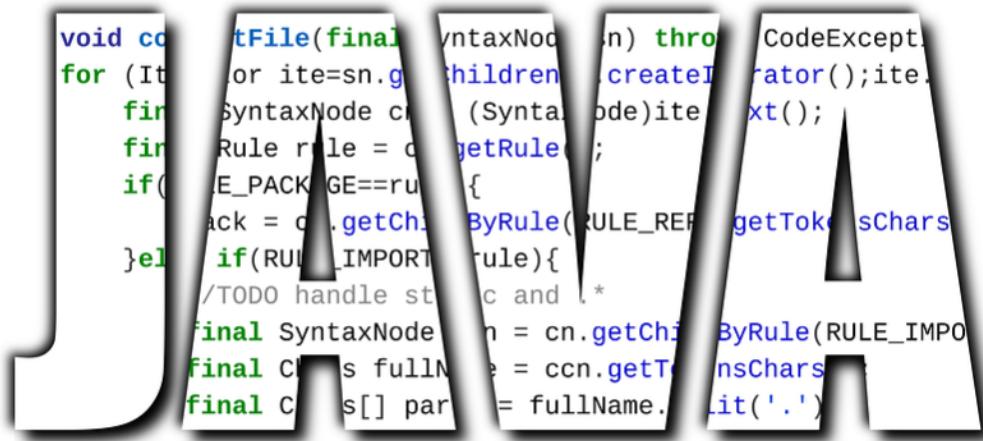
**STRING:** nombre de clase predefinido.

**ARGS:** nombre de la variable.

**A:** nombre de la variable.



# REGLAS IDENTIFICADORES JAVA



```
void co
for (It
    fin
    fin
    if(
        E_PACK
    }el
        if(RU
        /TODO han
        final Sy
        final Cl
        final C
        if(ite
            fin
            fin
            if(c
                if(RU
                    /TODO han
                    final Sy
                    final Cl
                    final C
                    if(cn.
                        fin
                        fin
                        if(ccn.
                            fin
                            fin
                            if(fullN
                                fin
                                fin
                                if('.'
                                    fin
                                    fin
                                    if(ccn.
                                        fin
                                        fin
                                        if(ccn.
                                            fin
                                            fin
                                            if(ccn.
                                                fin
                                                fin
                                                if(ccn.
                                                    fin
                                                    fin
                                                    if(ccn.
                                                        fin
                                                        fin
                                                        if(ccn.
                                                            fin
                                                            fin
                                                            if(ccn.
                                                                fin
                                                                fin
                                                                if(ccn.
                                                                    fin
                                                                    fin
                                                                    if(ccn.
                                                                        fin
                                                                        fin
                                                                        if(ccn.
                                                                            fin
                                                                            fin
                                                                            if(ccn.
                                                                                fin
                                                                                fin
                                                                                if(ccn.
                                                                                    fin
                                                                                    fin
                                                                                    if(ccn.
                                                                                        fin
                                                                                        fin
                                                                                        if(ccn.
                                                                                            fin
                                                                                            fin
                                                                                            if(ccn.
                                                                                                fin
                                                                                                fin
                                                                                                if(ccn.
                                                                                                    fin
                                                                                                    fin
                                                                                                    if(ccn.
                                                                                                        fin
                                                                                                        fin
................................................................
```



Los únicos caracteres permitidos para los identificadores son todos los caracteres alfanuméricos ([AZ], [az], [0-9]), "\$" (signo de dólar) y '\_' (guión bajo). Por ejemplo, "java@" no es un identificador de Java válido ya que contiene "@" – carácter especial.

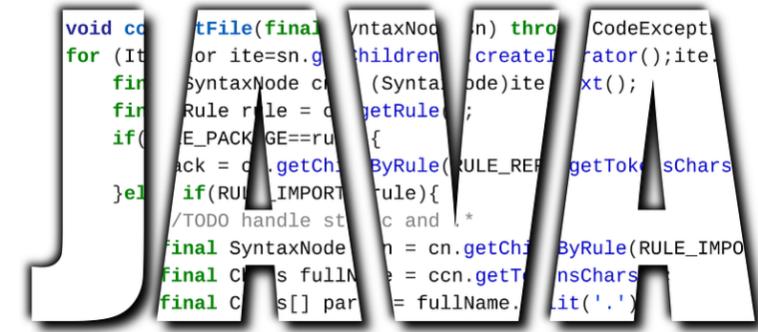


# REGLAS IDENTIFICADORES JAVA



Los identificadores no deben comenzar con dígitos ([0-9]). Por ejemplo, "123java" no es un identificador de Java válido.

Los identificadores de Java distinguen entre mayúsculas y minúsculas.



# REGLAS IDENTIFICADORES JAVA

```
void codeFile(final SyntaxNode cn) throws CodeException {
    for (Iterator ite = cn.getChildren(); ite.hasNext(); ) {
        final SyntaxNode child = (SyntaxNode) ite.next();
        final Rule rule = child.getRule();
        if (rule.getE_PACKAGE == rule) {
            back = child.getChildren();
        } else if (rule.getE_IMPORT == rule) {
            // TODO handle static imports
            final SyntaxNode ccn = child.getChildren();
            final Class fullN = ccn.getT();
            final Class[] pars = fullN.getParams();
            ByRule(RULE_IMPORT).getTokensChars(ccn);
        }
    }
}
```



*No hay límite en la longitud del identificador, pero es aconsejable usar solamente una longitud óptima de 4 a 15 caracteres.*



# EJEMPLOS IDENTIFICADORES JAVA

## BIEN DEFINIDOS

```
MyVariable  
MYVARIABLE  
myvariable  
x  
i  
x1  
i1  
_myvariable  
$myvariable  
sum_of_array  
javadesdecero
```

## MAL DEFINIDOS

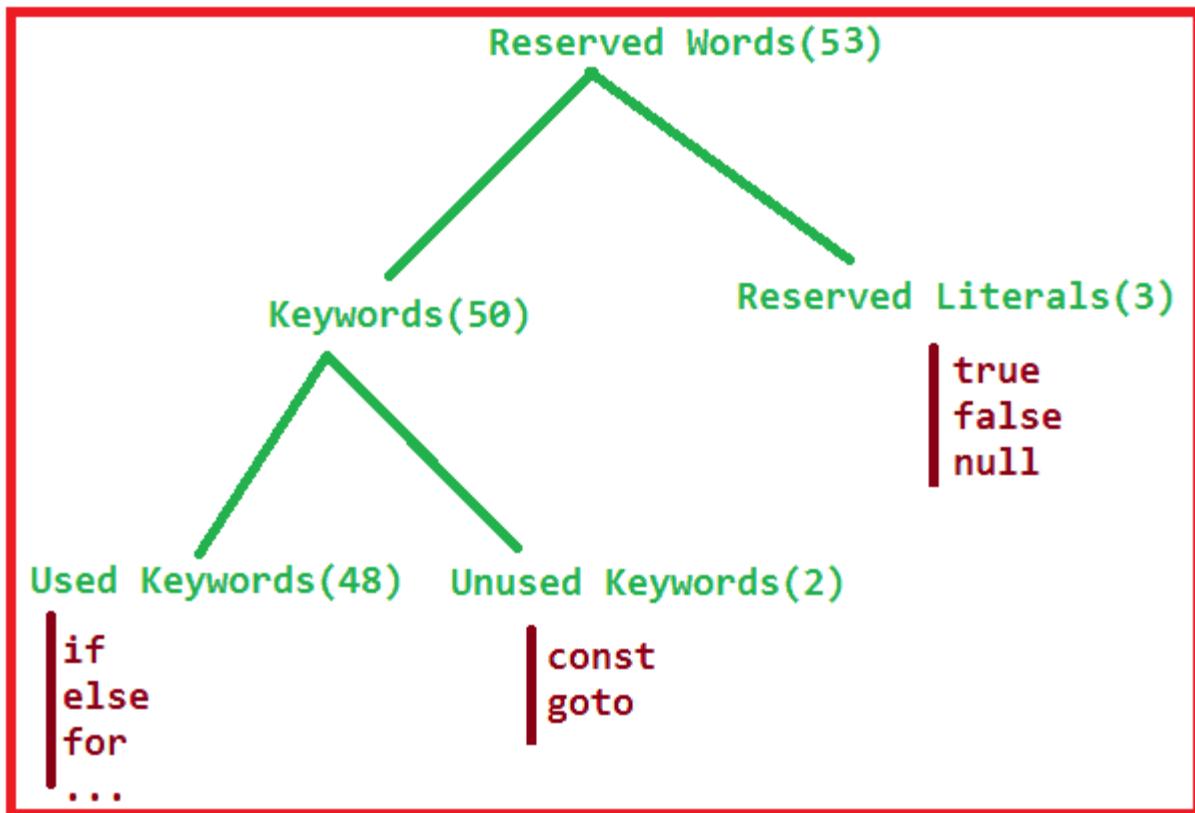
```
My Variable // Contiene un espacio  
123java // Comienza con un dígito  
a+c // El signo más (+) no es un carácter alfanumérico  
variable-2 // El guión (-) no es un carácter alfanumérico  
sum_&_difference // ampersand (&) no es un carácter válido
```



## PALABRAS RESERVADAS EN JAVA



Cualquier lenguaje de programación reserva algunas palabras para representar funcionalidades definidas por ese lenguaje. Estas palabras se llaman palabras reservadas. Estas pueden ser categorizadas brevemente en dos partes: palabras clave/keywords (50) y literales (3).



# PALABRAS RESERVADAS EN JAVA

Las palabras clave definen funcionalidades y las literales definen un valor.



# PALABRAS RESERVADAS EN JAVA

Tabla de Palabras Claves Reservadas en Java

abstract	assert	boolean	break	byte	case	catch	char	class	const	continue
default	do	double	else	enum	extends	false	final	finally	float	for
goto	if	implements	import	instanceof	int	interface	long	native	new	null
package	private	protected	public	return	short	static	strictfp	String	super	switch
synchronized	this	throw	throws	transient	true	try	void	volatile	while	



# *RETO INNOVA COLOMBIA*



*¿preguntas?*



# CONCLUSIONES



Politécnico  
Internacional

# *PROGRAMACION ORIENTADA A OBJETOS*

*“No me importa si funciona en su máquina!  
No me envían su máquina!”*

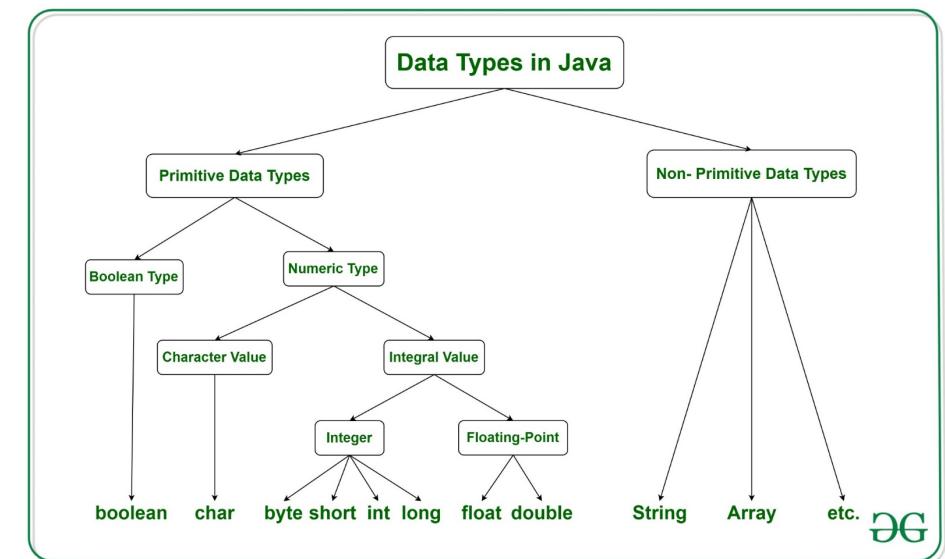
**VIDIU PLATON**

# AGENDA

- *Conceptos Java*
- *Innova Colombia*
- *Conclusiones*

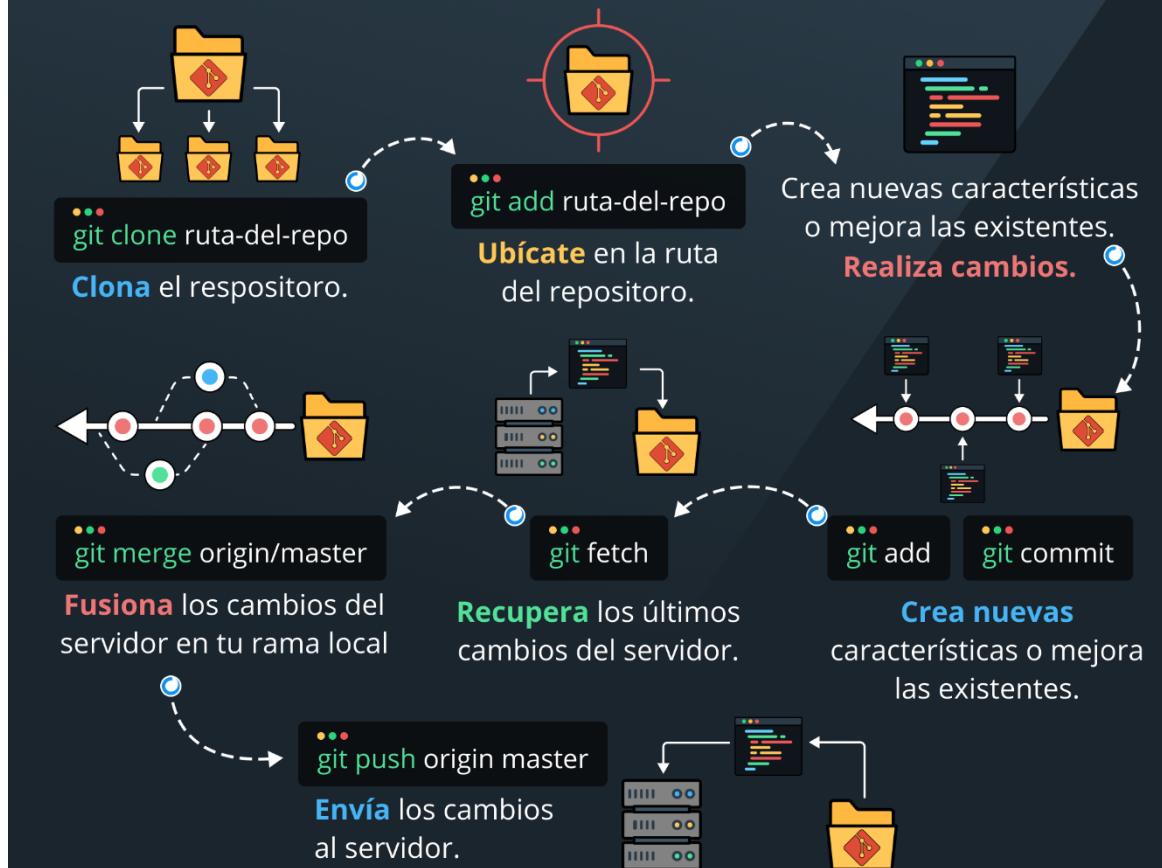
# Retroalimentación

- ¿Qué es Java?
- Características – Seguridad
- POO
- Sintaxis Básica
- Historia
- Tipado – Estático & Dinámico
- Tipos de datos.





## ¿CÓMO CONTRIBUIR EN UN PROYECTO CON GIT?



Aprende a controlar las versiones de tus proyectos:

👉 [ed.team/cursos/git](http://ed.team/cursos/git)



## TÉRMINOS DE git QUE DEBES CONOCER

### REPOSITORY/REPO

Base de datos donde se almacena el historial del código.

### COMMIT

Registro de uno o varios cambios hechos en el repositorio.

### STAGE

Lista de archivos que se usarán para un commit.

### FORK

Copia de un repositorio.

### BRANCH / RAMA

Entorno o espacio de trabajo independiente en Git.

### MASTER

Rama principal de un repositorio.

### CHECKOUT

Es la acción de moverse entre diferentes ramas.

### MERGE

Combina dos o más ramas en una.

Si eres programador es **obligatorio** aprender GIT. Domínalo en:

[ed.team/cursos/git](https://ed.team/cursos/git)

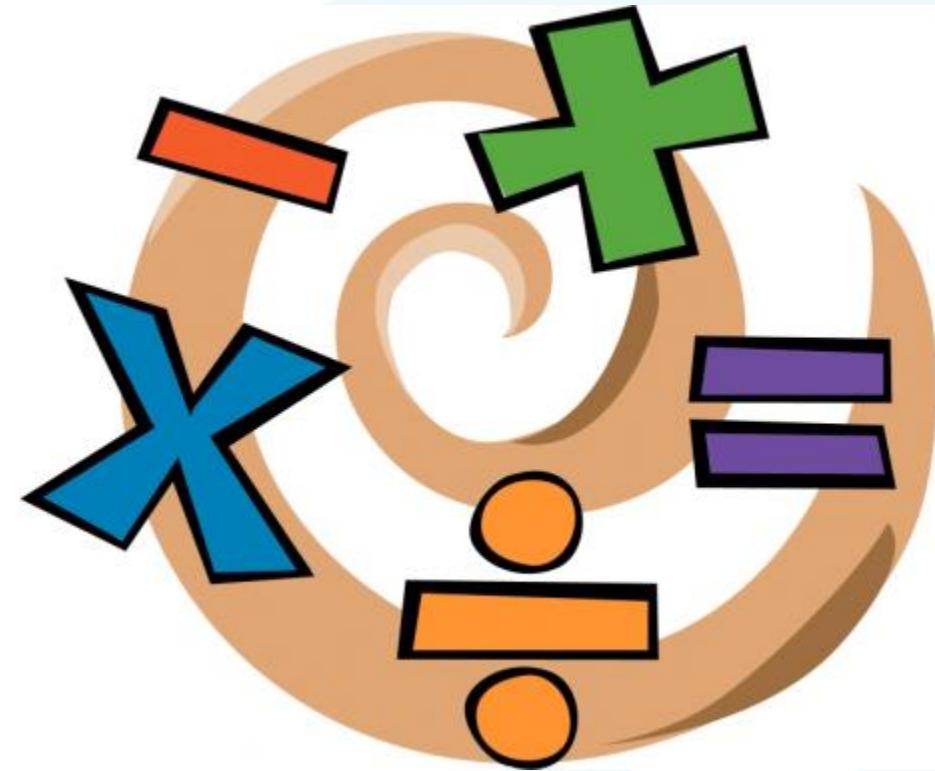


# OPERADORES EN JAVA

## CLASIFICACION OPERADORES JAVA

*Java proporciona muchos tipos de operadores que se pueden usar según la necesidad. Se clasifican según la funcionalidad que brindan. Algunos de los tipos son los siguientes:*

- Operadores aritméticos
- Operadores unarios
- Operador de asignación





# OPERADORES ARITMÉTICOS

**DATOS  
PRIMITIVOS**

**JAVA + OPERADORES  
ARITMÉTICOS**

```
// Programa Java para ilustrar
// operadores aritméticos
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        String x = "Thank", y = "You";

        // Operador + y -
        System.out.println("a + b = "+(a + b));
        System.out.println("a - b = "+(a - b));

        // El operador + si se usa con strings
        // concatena las cadenas dadas.
        System.out.println("x + y = "+x + y);

        // Operador * y /
        System.out.println("a * b = "+(a * b));
        System.out.println("a / b = "+(a / b));

        // operador de módulo da el resto
        // de dividir el primer operando con el segundo
        System.out.println("a % b = "+(a % b));

        // si el denominador es 0 en la división
        // System.out.println(a/c);
        // lanzaría una java.lang.ArithmeticException

    }
}
```

*Se utilizan para realizar operaciones aritméticas simples en tipos de datos primitivos.*

*\*: Multiplicación  
/: División  
%: Modulo  
+: Adición  
-: Resta*



```
// Programa Java para ilustrar
// operadores unarios
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;

        // operador de pre-incremento
        // a = a+1 y entonces c = a;
        c = ++a;
        System.out.println("Valor de c (++a) = " + c);

        // operador de post-incremento
        // c=b entonces b=b+1 (b pasa a ser 11)
        c = b++;
        System.out.println("Valor de c (b++) = " + c);

        // operador de pre-decrecimiento
        // d=d-1 entonces c=d
        c = --d;
        System.out.println("Valor de c (--d) = " + c);

        // operador de post-decrecimiento
        // c=e entonces e=e-1 (e pasa a ser 39)
        c = e--;
        System.out.println("Valor de c (e--) = " + c);

        // Operador lógico not
        System.out.println("Valor de !condition = " + !condition);
    }
}
```

# OPERADORES UNARIOS

DATOS  
PRIMITIVOS

JAVA + OPERADORES  
UNARIOS

*Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.*

-: Unario menos, utilizado para negar los valores.  
+: Unario más, usado para dar valores positivos. Solo se usa cuando se convierte deliberadamente un valor negativo en positivo.

++: Operador de incremento, utilizado para incrementar el valor en 1. Hay dos variedades de operador de incremento.  
Pre-Incremento: el valor se incrementa primero y luego se calcula el resultado.

# OPERADORES UNARIOS

**DATOS  
PRIMITIVOS**

**JAVA + OPERADORES  
UNARIOS**

*Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.*

*Post-Incremento: el valor se usa por primera vez para calcular el resultado y luego se incrementa.*

*—: Operador de decremento , usado para incrementar el valor en 1. Hay dos variedades de operador de incremento.*

*Pre-Decremento: el valor se disminuye primero y luego se calcula el resultado.*

*Post-Decremento: el valor se usa por primera vez para calcular el resultado y luego se disminuye.*

*!: Operador lógico “no”, utilizado para invertir un valor booleano.*

```
// Programa Java para ilustrar
// operadores unarios
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;

        // operador de pre-incremento
        // a = a+1 y entonces c = a;
        c = ++a;
        System.out.println("Valor de c (++a) = " + c);

        // operador de post-incremento
        // c=b entonces b=b+1 (b pasa a ser 11)
        c = b++;
        System.out.println("Valor de c (b++) = " + c);

        // operador de pre-decremento
        // d=d-1 entonces c=d
        c = --d;
        System.out.println("Valor de c (--d) = " + c);

        // operador de post-decremento
        // c=e entonces e=e-1 (e pasa a ser 39)
        c = e--;
        System.out.println("Valor de c (e--) = " + c);

        // Operador lógico not
        System.out.println("Valor de !condition = " + !condition);
    }
}
```

```

// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaria una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);

        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}

```

# OPERADORES ASIGNACIÓN

**DATOS  
PRIMITIVOS**

## JAVA + OPERADORES ASIGNACIÓN

*El operador de asignación se usa para asignar un valor a cualquier variable. Tiene una asociación de derecha a izquierda, es decir, el valor dado en el lado derecho del operador se asigna a la variable de la izquierda y, por lo tanto, el valor del lado derecho debe declararse antes de usarlo o debe ser una constante.*

*El formato general del operador de asignación es,  
[java]variable = valor;[/java]*



```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);

        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

# OPERADORES ASIGNACIÓN

## DATOS PRIMITIVOS

### JAVA + OPERADORES ASIGNACIÓN

En muchos casos, el operador de asignación se puede combinar con otros operadores para construir una versión más corta de la declaración llamada **Declaración Compuesta (Compound Statement)**. Por ejemplo, en lugar de  $a = a + 5$ , podemos escribir  $a += 5$ .

```
int a = 5;
a += 5; // a = a + 5;
```

```

// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaria una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);

        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}

```

# OPERADORES ASIGNACIÓN

**DATOS  
PRIMITIVOS**

**JAVA + OPERADORES  
ASIGNACIÓN**

*+ = , para sumar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*

*- = , para restar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*

- = , para multiplicar el operando izquierdo con el operando derecho y luego asignándolo a la variable de la izquierda.*

```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaria una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

# OPERADORES ASIGNACIÓN

**DATOS  
PRIMITIVOS**

**JAVA + OPERADORES  
ASIGNACIÓN**

*/ = , para dividir el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*

*^ = , para aumentar la potencia del operando izquierdo al operando derecho y asignarlo a la variable de la izquierda.*

*% = , para asignar el módulo del operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*



## PALABRAS RESERVADAS



En los lenguajes de los programación, identificadores (como su nombre lo indica) se utilizan con fines de identificación. En Java, un identificador puede ser un nombre de clase, un nombre de método o un nombre de variable.

# PALABRAS RESERVADAS

```
public class Test
{
    public static void main(String[] args)
    {
        int a = 28;
    }
}
```



## IDENTIFICADORES

**TEST:** nombre de clase.

**MAIN:** nombre del método.

**STRING:** nombre de clase predefinido.

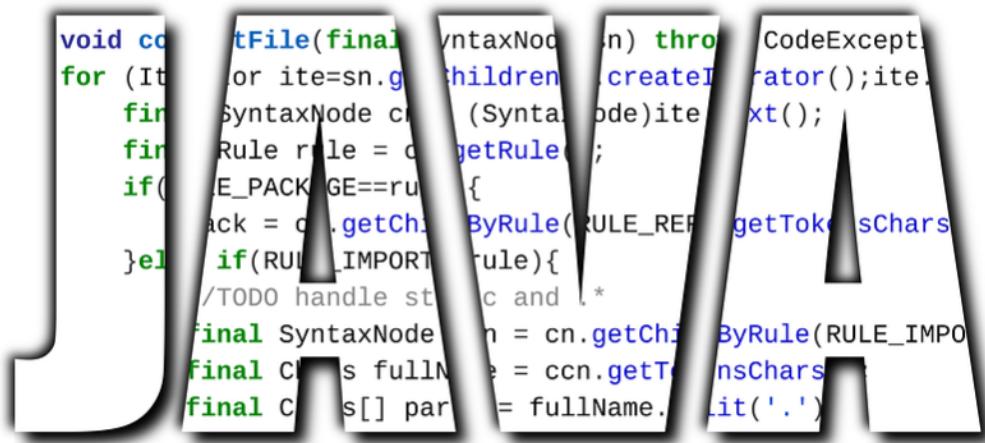
**ARGS:** nombre de la variable.

**A:** nombre de la variable.

# REGLAS IDENTIFICADORES JAVA



Los únicos caracteres permitidos para los identificadores son todos los caracteres alfanuméricos ([AZ], [az], [0-9]), "\$" (signo de dólar) y '\_' (guión bajo). Por ejemplo, "java@" no es un identificador de Java válido ya que contiene "@" – carácter especial.



# REGLAS IDENTIFICADORES JAVA



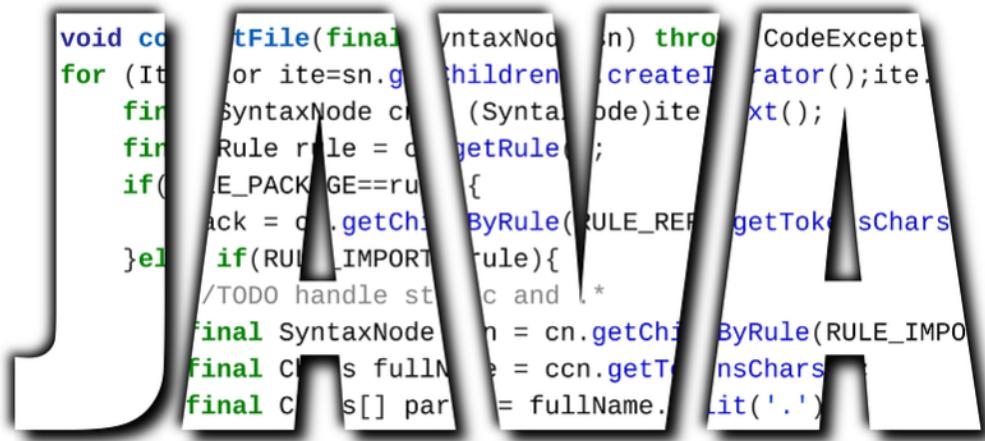
Los identificadores no deben comenzar con dígitos ([0-9]). Por ejemplo, “123java” no es un identificador de Java válido.

*Los identificadores de Java distinguen entre mayúsculas y minúsculas.*

```
void co tFile(final Synta n) thro CodeExcept  
for (It or ite=sn.g hildren createI rator();ite.  
fir SyntaxNode c (Syntaode)ite xt();  
fir Rule rule = c getRule ;  
if( E_PACK GE==ru {  
ack = C_.getCh ByRule(RULE_REF getTok sChars  
}el if(RULE_IMPORT rule){  
//TODO handle stic and /*  
final SyntaxNode n = cn.getCh ByRule(RULE_IMPO  
final C_ s fullN e = ccn.getTok nsChars  
final C_ s[] par = fullName.  
lit('.')
```



# REGLAS IDENTIFICADORES JAVA



```
void co tFile(final SyntaxNode cn) throws CodeExcepti
for (It or ite=sn.getChil
    fin SyntacticNode c
    fin Rule rule = c.getRule();
    if(E_PACKAGE==rule)
    pack = c.getCh ByRule(RULE_REF).getTok
    }el if(RULE_IMPORT==rule){
    /TODO handle st
    final SyntaxNode cn = cn.getCh ByRule(RULE_IMPO
    final Class fullNme = ccn.getTok
    final Class[] par
    fullName. lit('.'
```



No hay límite en la longitud del identificador, pero es aconsejable usar solamente una longitud óptima de 4 a 15 caracteres.



# EJEMPLOS IDENTIFICADORES JAVA

## BIEN DEFINIDOS

```
MyVariable  
MYVARIABLE  
myvariable  
x  
i  
x1  
i1  
_myvariable  
$myvariable  
sum_of_array  
javadesdecero
```

## MAL DEFINIDOS

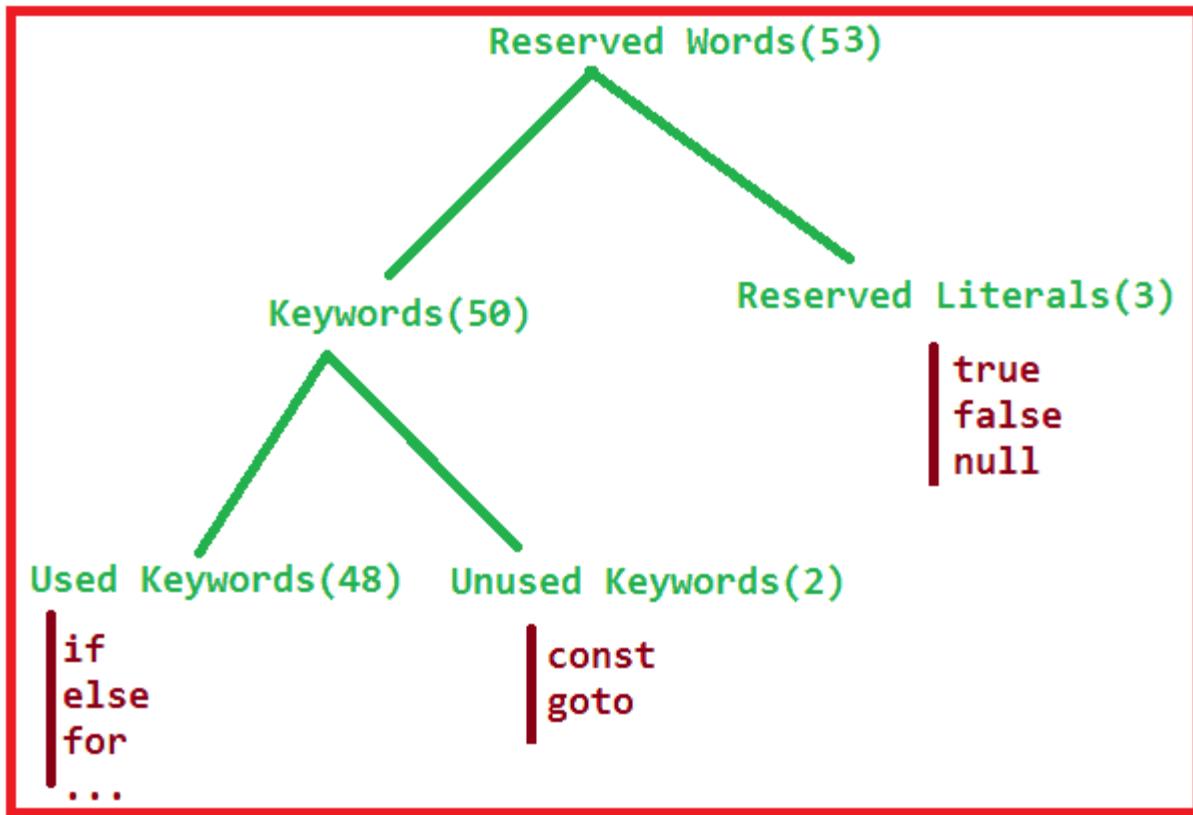
```
My Variable // Contiene un espacio  
123java // Comienza con un dígito  
a+c // El signo más (+) no es un carácter alfanumérico  
variable-2 // El guión (-) no es un carácter alfanumérico  
sum_&_difference // ampersand (&) no es un carácter válido
```



# PALABRAS RESERVADAS EN JAVA



Cualquier lenguaje de programación reserva algunas palabras para representar funcionalidades definidas por ese lenguaje. Estas palabras se llaman palabras reservadas. Estas pueden ser categorizadas brevemente en dos partes: palabras clave/keywords (50) y literales (3).



# PALABRAS RESERVADAS EN JAVA

Las palabras clave definen funcionalidades y las literales definen un valor.

# PALABRAS RESERVADAS EN JAVA

Tabla de Palabras Claves Reservadas en Java

abstract	assert	boolean	break	byte	case	catch	char	class	const	continue
default	do	double	else	enum	extends	false	final	finally	float	for
goto	if	implements	import	instanceof	int	interface	long	native	new	null
package	private	protected	public	return	short	static	strictfp	String	super	switch
synchronized	this	throw	throws	transient	true	try	void	volatile	while	



# CONVENCIONES DE NOMENCLATURA **Java**

*Las convenciones de nombres para el lenguaje de programación Java. Deben seguirse mientras se desarrolla software en Java para un buen mantenimiento y legibilidad del código. Java utiliza CamelCase como una práctica para escribir nombres de métodos, variables, clases, paquetes y constantes.*



## **CLASES E INTERFACES EN JAVA**

```
Interface Bicycle
Class MountainBike implements Bicycle

Interface Sport
Class Football implements Sport
```

# **CONVENCIONES DE NOMENCLATURA JAVA**



Los nombres de las clases deben ser sustantivos, en mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúscula. El nombre de las interfaces también debe estar en mayúscula (la primera) al igual que los nombres de las clases. Use palabras completas y debe evitar acrónimos y abreviaturas.

## METODOS EN JAVA

```
void changeGear(int newValue);  
void speedUp(int increment);  
void applyBrakes(int decrement);
```

# CONVENCIONES DE NOMENCLATURA JAVA



*Los métodos deben ser verbos, en mayúsculas y minúsculas, con la primera letra de cada palabra interna (a partir de la segunda) en mayúscula.*

# CONVENCIONES DE NOMENCLATURA JAVA

## VARIABLES EN JAVA

```
// variables para la clase MountainBike
int speed = 0;
int gear = 1;
```



*Los nombres de las variables deben ser cortos pero significativos.*

*No debería comenzar con un guión bajo ('\_') o caracteres, como por ejemplo, un signo de dólar '\$'.*

*Debe ser mnemotécnico, es decir, diseñado para indicar al observador casual la intención de su uso.*

*Se deben evitar los nombres de variable de un carácter, excepto para variables temporales.*

*Los nombres comunes para las variables temporales son: i, j, k, m y n para enteros; c, d y e para los caracteres.*

# CONVENCIONES DE NOMENCLATURA JAVA

## VARIABLES CONSTANTES EN JAVA

```
static final int MIN_WIDTH = 4;

// Algunas variables constantes utilizadas en la clase Float predefinida
public static final float POSITIVE_INFINITY = 1.0f / 0.0f;
public static final float NEGATIVE_INFINITY = -1.0f / 0.0f;
public static final float NaN = 0.0f / 0.0f;
```



Debería estar todo en mayúsculas con palabras separadas por guiones bajos ("\_").

Hay varias constantes utilizadas en clases predefinidas como *Float, Long, String, etc.*

# CONVENCIONES DE NOMENCLATURA JAVA

## PAQUETES EN JAVA

```
com.sun.eng
com.apple.quicktime.v2

// java.lang packet in JDK
java.lang
```



*El prefijo de un nombre de paquete único siempre se escribe en letras ASCII en minúsculas y debe ser uno de los nombres de dominio de nivel superior, como por ejemplo: com, edu, gov, mil, net, org.*

*Los componentes posteriores del nombre del paquete varían de acuerdo con las convenciones internas de nombres de la organización.*



# CLASES PREDEFINIDAS

*NOMBRE DE CLASE*

O

*VARIABLE EN JAVA*



# CLASES DE PREDIFINIDAS

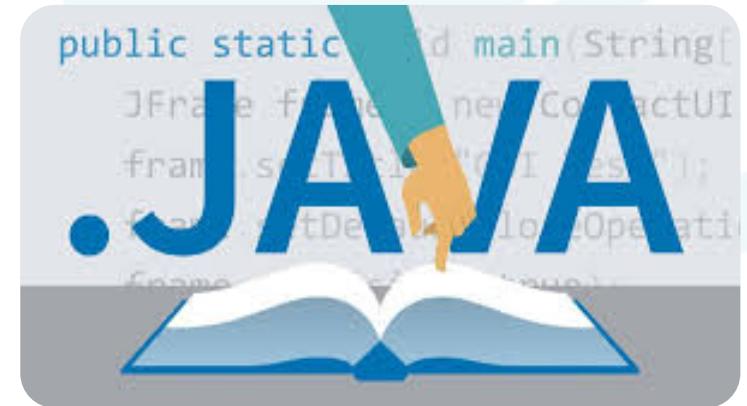
Los siguientes son algunos nombres de clase **NO VÁLIDOS** en Java

```
boolean break = false; // no permitido ya que break es palabra clave
int boolean = 28; // no permitido ya que boolean es palabra clave
boolean goto = false; // no permitido ya que goto es palabra clave
String final = "hola"; // no permitido ya que final es palabra clave
```

# DECLARACION Y TIPOS DE VARIABLES

*Una variable es el nombre dado a una ubicación de memoria. Es la unidad básica de almacenamiento en un programa. El valor almacenado en una variable se puede cambiar durante la ejecución del programa.*

**NOTA:** "En Java, todas las variables deben declararse antes de que puedan ser utilizadas."



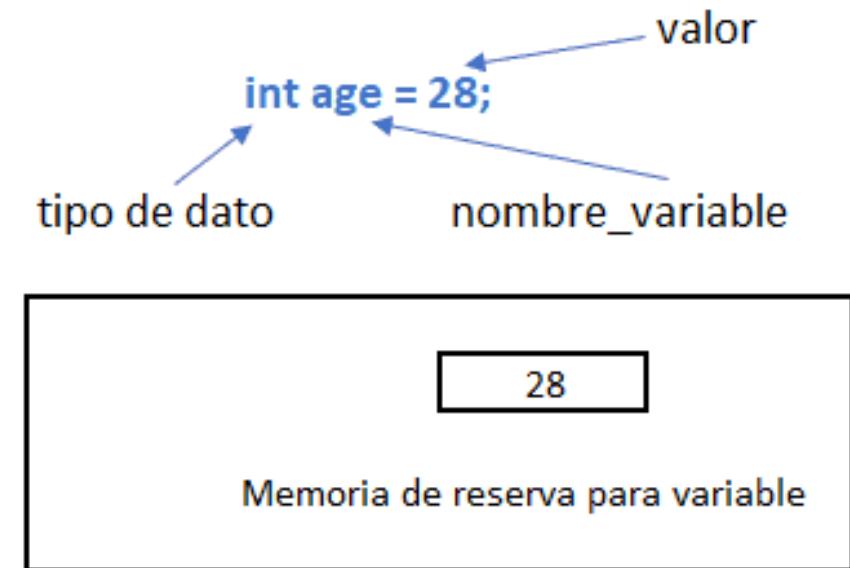
# ¿CÓMO SE DECLARA VARIABLES?

## DATOS PRIMITIVOS

**TIPO DE DATO:** tipo de datos que se pueden almacenar en esta variable.

**NOMBRE\_VARIABLE:** nombre dado a la variable.

**VALOR:** es el valor inicial almacenado en la variable.



# ¿CÓMO DECLARA VARIABLES?

```
float simpleInterest; //Declarando variable float
int time = 10, speed = 20; //Declarando e Inicializando la variable integer
char var = 'h'; // Declarando e Inicializando la variable character
```

# TIPOS DE VARIABLES

**JAVA + TIPOS:**

- *VARIABLES LOCALES*
- *VARIABLES DE INSTANCIA*
- *VARIABLES ESTÁTICAS*



*Una variable definida dentro de un bloque, método o constructor se llama variable local.*

*Estas variables se crean cuando el bloque ingresado o método se llama y destruye después de salir del bloque o cuando la llamada regresa del método.*

*El alcance de estas variables solo existe dentro del bloque en el que se declara la variable, es decir, podemos acceder a estas variables solo dentro de ese bloque.*

```
public class StudentDetails
{
    public void StudentAge()
    {
        //variable local age
        int age = 0;
        age = age + 5;
        System.out.println("La edad del estudiante es : " + age);
    }

    public static void main(String args[])
    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}
```



# TIPOS DE VARIABLES LOCALES

*En el programa anterior, la variable age (edad) es una variable local para el método [java]StudentAge()[/java].*

*Si usamos la variable age fuera del método [java]StudentAge()[/java], el compilador producirá un error como se muestra a continuación en el programa*

```
public class StudentDetails
{
    public void StudentAge()
    {   //variable local age
        int age = 0;
        age = age + 5;
        System.out.println("La edad del estudiante es : " + age);
    }

    public static void main(String args[])
    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}
```

*Las variables de instancia son variables no estáticas y se declaran en una clase fuera de cualquier método, constructor o bloque.*

*Como las variables de instancia se declaran en una clase, estas variables se crean cuando un objeto de la clase se crea y se destruye cuando se destruye el objeto.*

*A diferencia de las variables locales, podemos usar especificadores de acceso para variables de instancia. Si no especificamos ningún especificador de acceso, se utilizará el especificador de acceso predeterminado.*

```
import java.io.*;
class Points
{
    //Estas variables son variables de instancia.
    //Estas variables están en una clase y no están dentro de ninguna función/método
    int engPoints;
    int mathsPoints;
    int phyPoints;
}

class PointsDemo
{
    public static void main(String args[])
    {
        //primer objeto
        Points obj1 = new Points();
        obj1.engPoints = 50;
        obj1.mathsPoints = 80;
        obj1.phyPoints = 90;

        //segundo objeto
        Points obj2 = new Points();
        obj2.engPoints = 80;
        obj2.mathsPoints = 60;
        obj2.phyPoints = 85;

        //mostrando puntos para el primer objeto
        System.out.println("Puntos para el primer objeto:");
        System.out.println(obj1.engPoints);
        System.out.println(obj1.mathsPoints);
        System.out.println(obj1.phyPoints);

        //mostrando puntos para el segundo objeto
        System.out.println("Puntos para el segundo objeto:");
        System.out.println(obj2.engPoints);
        System.out.println(obj2.mathsPoints);
        System.out.println(obj2.phyPoints);
    }
}
```

```
import java.io.*;
class Points
{
    //Estas variables son variables de instancia.
    //Estas variables están en una clase y no están dentro de ninguna función/método
    int engPoints;
    int mathsPoints;
    int phyPoints;
}

class PointsDemo
{
    public static void main(String args[])
    {   //primer objeto
        Points obj1 = new Points();
        obj1.engPoints = 50;
        obj1.mathsPoints = 80;
        obj1.phyPoints = 90;

        //segundo objeto
        Points obj2 = new Points();
        obj2.engPoints = 80;
        obj2.mathsPoints = 60;
        obj2.phyPoints = 85;

        //mostrando puntos para el primer objeto
        System.out.println("Puntos para el primer objeto:");
        System.out.println(obj1.engPoints);
        System.out.println(obj1.mathsPoints);
        System.out.println(obj1.phyPoints);

        //mostrando puntos para el segundo objeto
        System.out.println("Puntos para el segundo objeto:");
        System.out.println(obj2.engPoints);
        System.out.println(obj2.mathsPoints);
        System.out.println(obj2.phyPoints);
    }
}
```



# TIPOS DE VARIABLES

## ESTÁTICAS

*Las variables estáticas también se conocen como variables de clase.*

*Estas variables se declaran de forma similar a las variables de instancia, la diferencia es que las variables estáticas se declaran utilizando la palabra clave [java]static[/java] dentro de una clase fuera de cualquier constructor o bloque de métodos.*

*A diferencia de las variables de instancia, solo podemos tener una copia de una variable estática por clase, independientemente de cuántos objetos creamos.*

*Las variables estáticas se crean al inicio de la ejecución del programa y se destruyen automáticamente cuando finaliza la ejecución.*

```
import java.io.*;
class Emp {

    // salario como variable estatica
    public static double salary;
    public static String name = "Alex";
}

public class EmpDemo
{
    public static void main(String args[]) {

        //acceder a la variable estatica sin objeto
        Emp.salary = 1000;
        System.out.println(Emp.name + " tiene un salario promedio de: " + Emp.salary);
    }
}
```



# TIPOS DE VARIABLES

## CONSTANTE S

### VARIABLES CONSTANTES

Las variables creadas para almacenar valores fijos de esta manera se conocen como "constantes", y es convencional nombrar constantes con todos los caracteres en mayúsculas, para distinguirlas de las variables regulares. Los programas que intentan cambiar un valor constante no se compilarán, y el compilador javac generará un mensaje de error.

```
class Constants
{
    public static void main ( String[] args ) {
        //inicializamos tres constantes enteras
        final int CONSTANTE1 = 6 ;
        final int CONSTANTE2 = 1 ;
        final int CONSTANTE3 = 3 ;

        //declaramos variables regulares del tipo int
        int td,pat,fg,total;

        //Inicializamos las variables regulares
        td = 4*CONSTANTE1;
        pat = 3*CONSTANTE2;
        fg = 2*CONSTANTE3;

        total = (td+pat+fg) ;

        System.out.println("Resultado: " + total);
    }
}
```

# AVANCES INNOVA COLOMBIA



Politécnico  
Internacional



PORCENTAJES DE EVALUACIÓN

*¿preguntas?*



Politécnico  
Internacional



PORCENTAJES DE EVALUACIÓN

# CONCLUSIONES



Politécnico  
Internacional

# PROGRAMACION ORIENTADA A OBJETOS



Politécnico  
Internacional



*“Las grandes oportunidades nacen de  
haber sabido aprovechar las pequeñas.”*



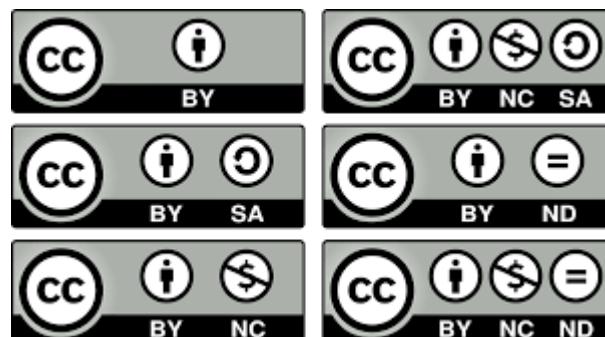
# AGENDA

- *Retroalimentación*
- *¿Que es una clase?*
- *¿Que es un objeto?*
- *¿Que es un metodo?*
- *¿Que es un constructor?*
- *Reto Practico*
- *Conclusiones*

**FORMATO IEEE**  
<https://cutt.ly/3Z0W3pk>



**CREATIVE COMMONS**  
<https://cutt.ly/JZ0EZ9J>



FORMATO IEEE	ENTREGABLE
<i>Titulo del Proyecto</i>	<i>Se debe dar un nombre al proyecto propuesto. Ejemplo: BUHO – Buen Uso de Herramientas Ofimáticas</i>
<i>Licenciamiento CC</i>	<i>Se debe indicar el tipo de licenciamiento de acuerdo a CC. Incluir Imagen de licenciamiento.</i>
<i>Resumen</i>	<i>Se debe realizar un resumen del proyecto y el objetivo del articulo.</i>
<i>Palabras Clave</i>	<i>Indicar 5 palabras clave.</i>
<i>Abstract</i>	<i>Se debe realizar un resumen del proyecto y el objetivo del articulo en inglés.</i>
<i>Introducción</i>	<i>Se debe realizar la introducción del proyecto de acuerdo al proyecto innova y el reto de Stranger Things. Contar que es lo que se esta haciendo.</i>
<i>Definiciones</i>	<ul style="list-style-type: none"> <li>- <i>Listar requerimientos para el desarrollo. (JAVA)</i></li> <li>• <i>Algoritmo</i></li> <li>• <i>Diagrama de flujo</i></li> <li>• <i>Pseudocódigo</i></li> <li>• <i>Pruebas de escritorio</i></li> <li>• <i>Primer programa en código (JAVA)</i></li> </ul>
<i>Implementación</i>	<p><i>Nota: Adjuntar evidencias de la implementación pantalla.</i></p> <p><i>Instalación de NetBeans junto a los requerimientos de hardware.</i></p>
<i>Referencias</i>	<i>De acuerdo a la norma APA v6.</i>

# *¿Que es una clase en POO?*

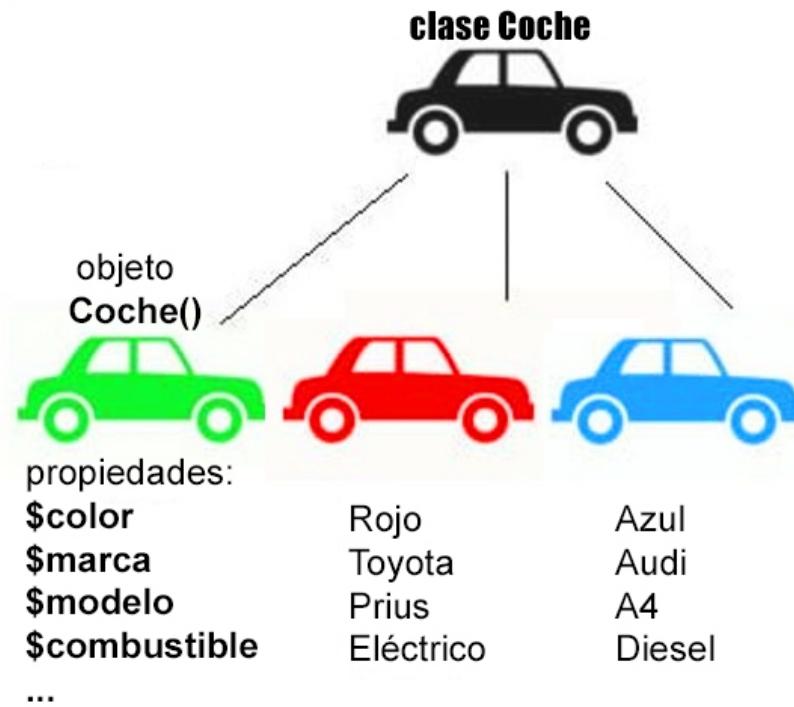
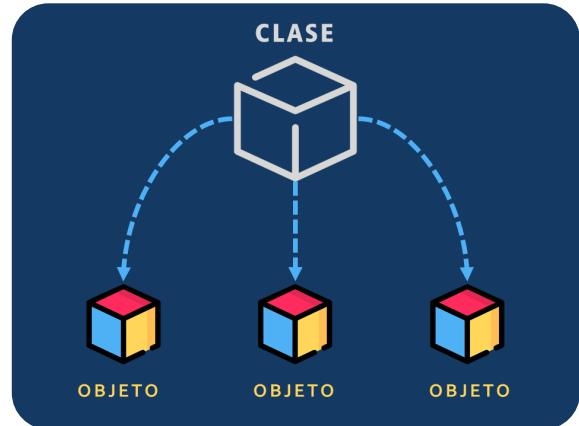


Una clase **es una plantilla** que define la forma de un objeto. **Especifica los datos y el código** que operará en esos datos. Java usa una especificación de clase para construir objetos. Los objetos son instancias de una clase. Por lo tanto, **una clase es esencialmente un conjunto de planes que especifican cómo construir un objeto.**



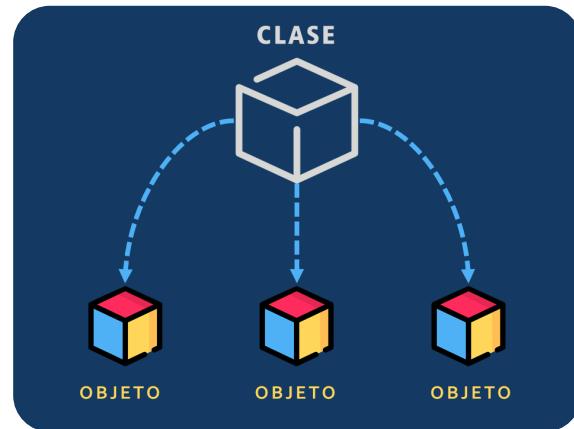
```
class NombreClase{  
    //Declarar variables de instancia  
    tipo var1;  
    tipo var2;  
    //..  
    //Declarar métodos  
    tipo metodo1(parámetros){  
        //Cuerpo del método  
    }  
    tipo metodo2(parámetros){  
        //Cuerpo del método  
    }  
}
```

**Una clase bien diseñada  
debería definir una y solo una  
entidad lógica**

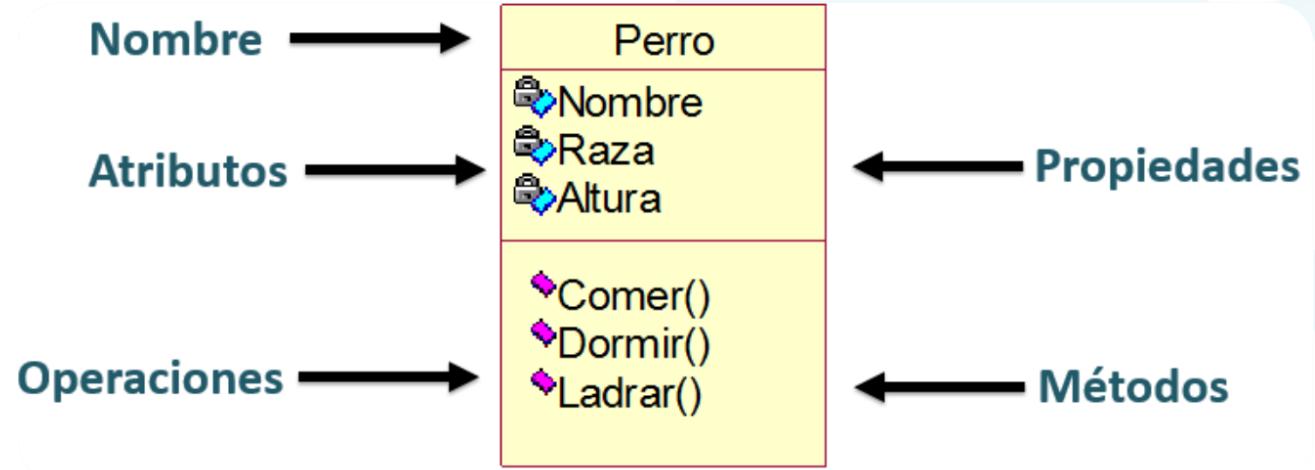
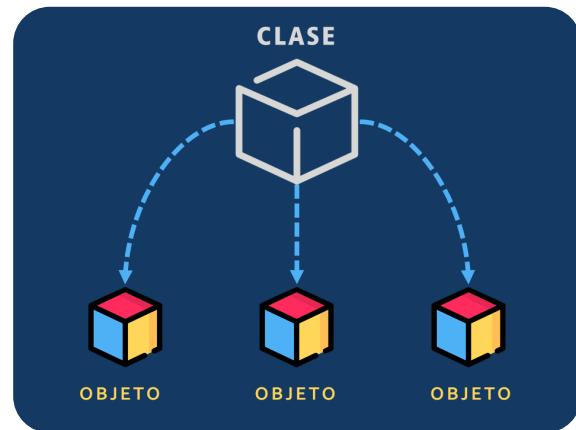


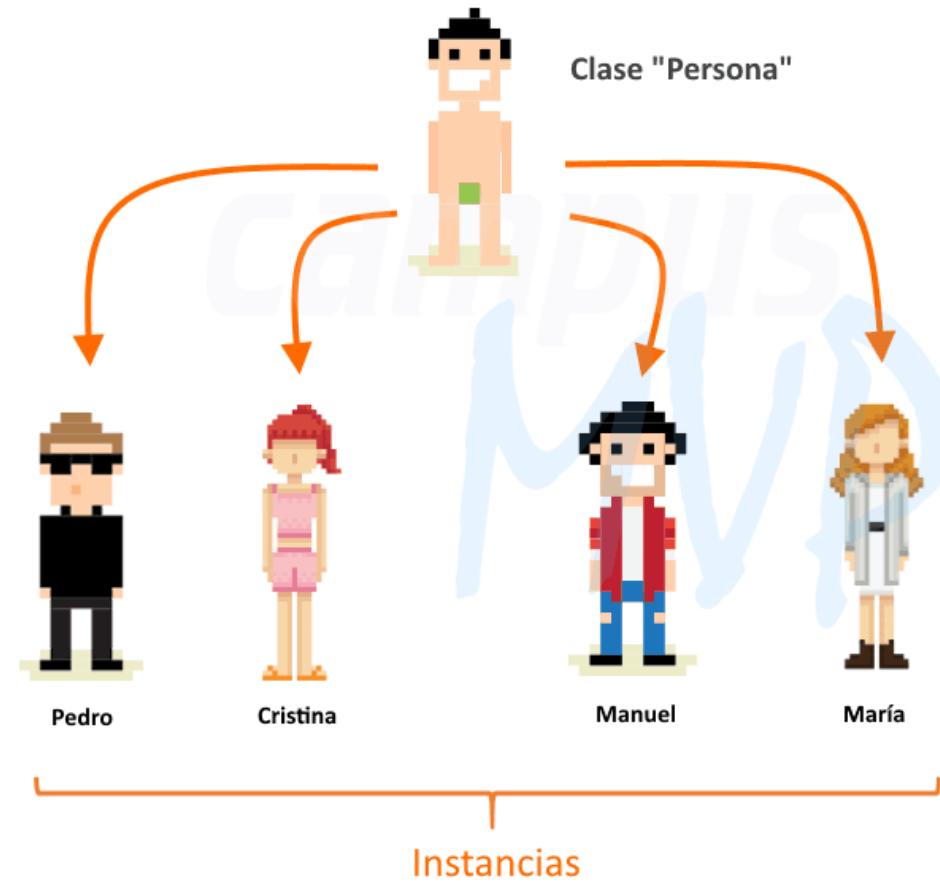
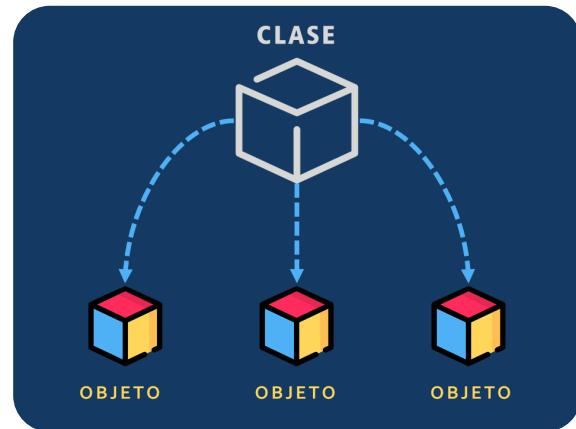
### MÉTODOS

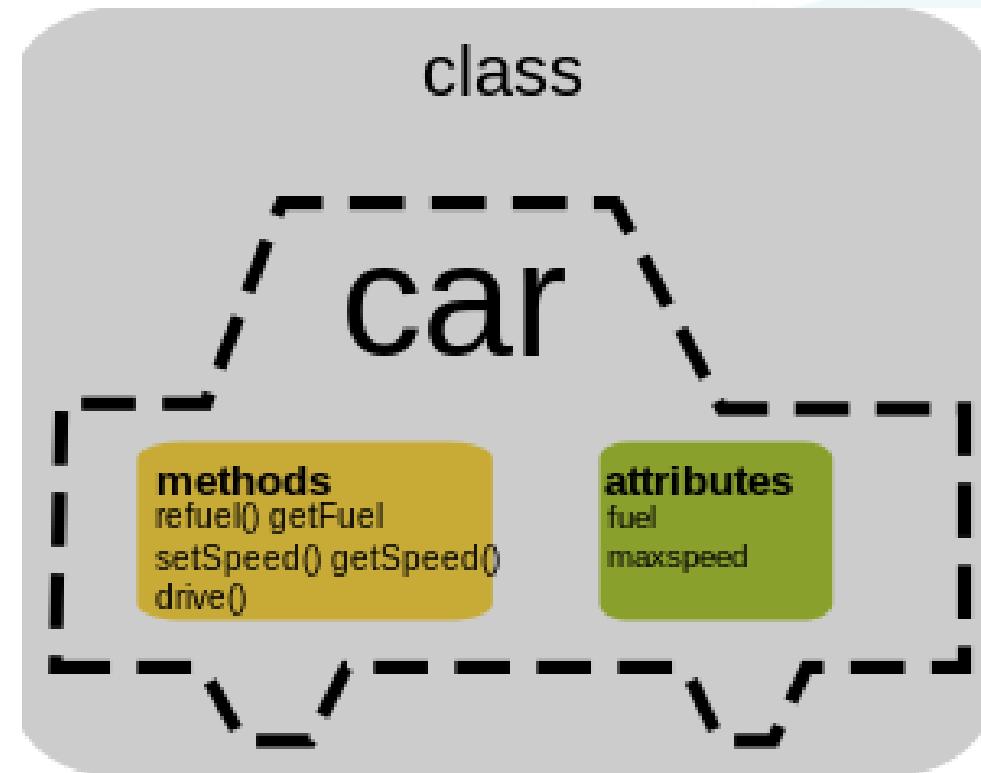
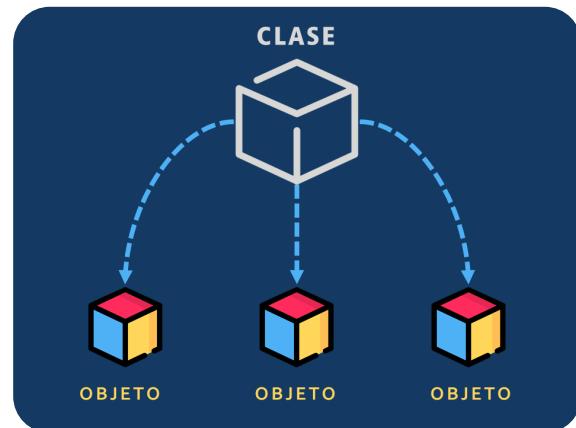
llenarDeposito()  
arrancarMotor()  
frenar()  
acelerar()  
tocarClaxon()  
...

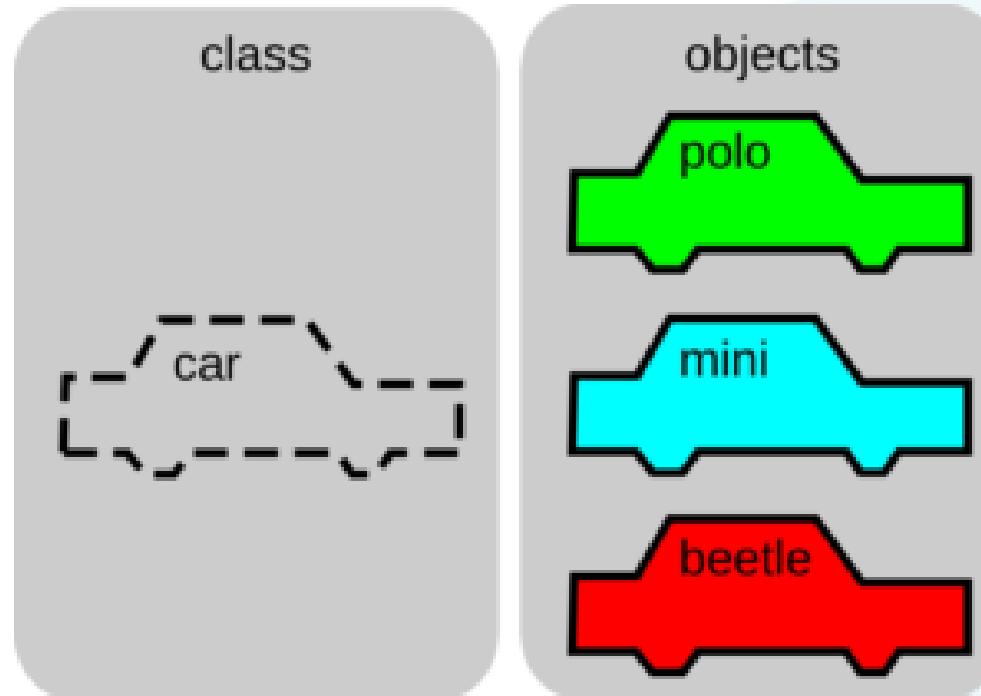
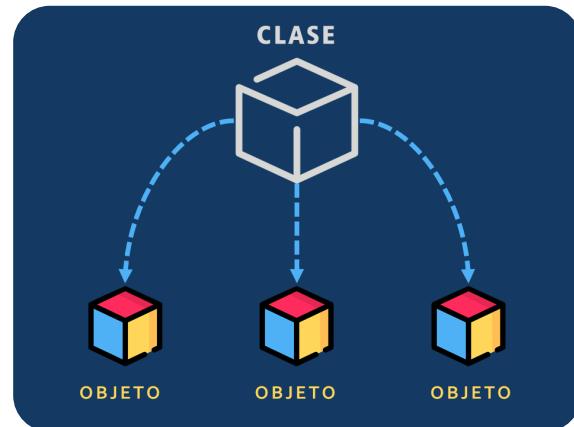


Nombre →	Vehículo
	marca
	modelo
	color
	número de serie
Atributos ←	
	mover()
	abrir()
	girar()
	detener()
Operaciones ←	









```
class Vehiculo{  
    int pasajeros; //número de pasajeros  
    int capacidad; //capacidad en galones  
    int mpg;      //consumo de combustible en millas por galón  
}
```



“ Un objeto en Java no es ni más ni menos que lo mismo que un objeto en el mundo real. ”

### Definición

*Un objeto es una instancia de una clase, creada en tiempo de ejecución. Es una estructura de datos formada por tantos campos como atributos tiene la clase. El estado de un objeto viene dado por el valor de los campos. Los métodos permiten consultar y modificar el estado del objeto*

**“Los objetos corresponden a cosas que se encuentran en el mundo real. Por ejemplo, un sistema de compra en línea podría tener objetos como: carrito de compras, cliente y producto.”**



### Campos o atributos.

Se trata del componente de un objeto en Java que recopila y almacena datos. Pueden ser primitivos o, a su vez, otro tipo de objetos. Lo que se conoce como agregación o composición de objetos. La idea detrás de esta característica es que cada atributo o campo representa una propiedad determinada del objeto.

### Rutinas o métodos.

Las rutinas llevan a cabo una serie de acciones o tareas en función de los atributos que definen al objeto.



### Estado

*Está representado por atributos de un objeto. También refleja las propiedades de un objeto.*

### Comportamiento

se representa mediante métodos de un objeto. También refleja la respuesta de un objeto con otros objetos

### Identidad

le da un nombre único a un objeto y permite que un objeto interactúe con otros objetos.



### Estado

*Está representado por atributos de un objeto. También refleja las propiedades de un objeto.*

### Comportamiento

se representa mediante métodos de un objeto. También refleja la respuesta de un objeto con otros objetos

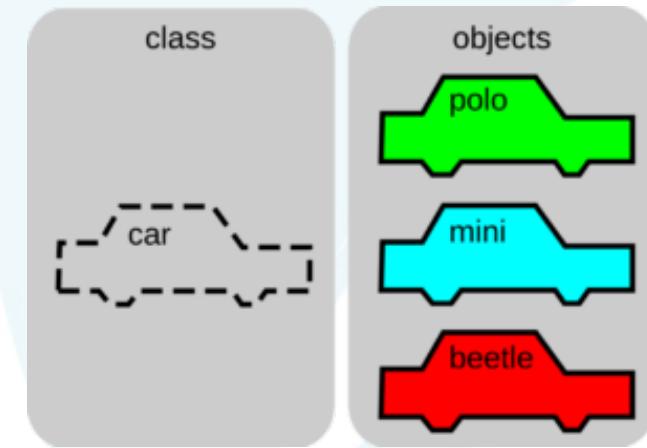
### Identidad

le da un nombre único a un objeto y permite que un objeto interactúe con otros objetos.



```
Vehiculo minivan = new Vehiculo(); //Creando un objeto vehículo llamado minivan
```

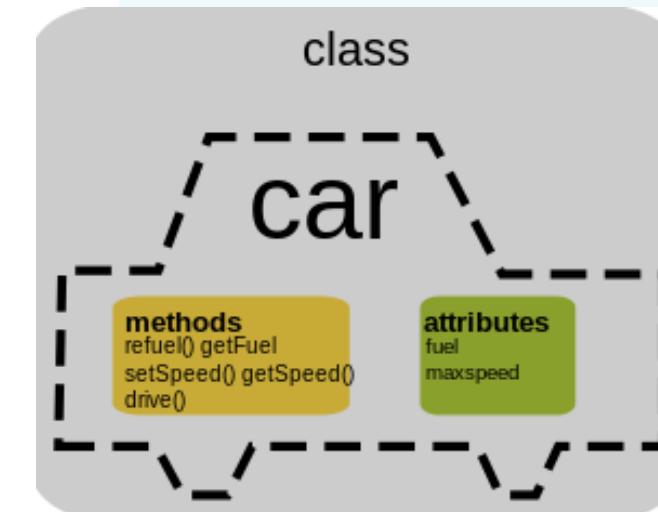
```
/* Un programa que usa la clase Vehiculo
El archivo se llama DemoVehiculo.java
*/
class Vehiculo {
    int pasajeros; //números de pasajeros
    int capacidad; //capacidad del combustible en galones
    int mpg;      //combustible consumido en millas por galon
}
//Esta clase declara un objeto de tipo Vehiculo
class DemoVehiculo {
    public static void main(String [] args) {
        Vehiculo minivan = new Vehiculo();
        int rango;
        //asignando valores a los campos de minivan
        minivan.pasajeros = 9;
        minivan.capacidad = 15;
        minivan.mpg = 20;
        //Calcular el rango asumiendo un tanque lleno
        rango = minivan.capacidad * minivan.mpg;
        System.out.println("La Minivan puede llevar " + minivan.pasajeros + " pasajeros con un rango de " +
rango + " millas");
    }
}
```



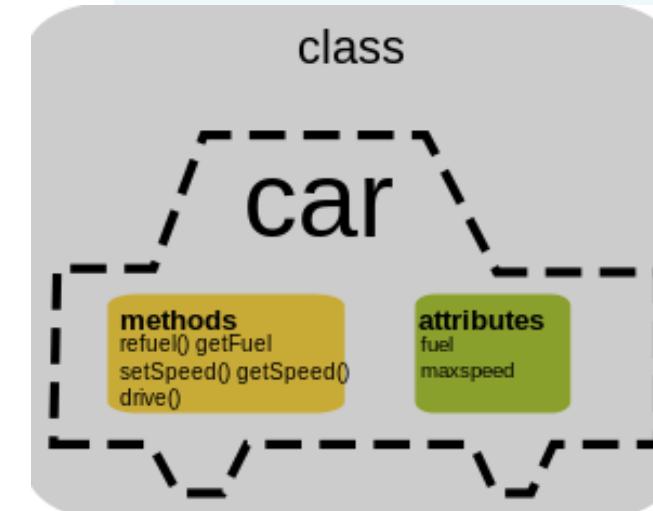
```

//Este programa crea dos objetos Vehiculo
class Vehiculo {
    int pasajeros; //números de pasajeros
    int capacidad; //capacidad del combustible en galones
    int mpg; //combustible consumido en millas por galon
}
//Esta clase declara un objeto de tipo Vehiculo
class DosVehiculo {
    public static void main(String[] args) {
        Vehiculo minivan = new Vehiculo();
        Vehiculo sportscar = new Vehiculo();
        int rango1, rango2;
        //asignando valores a los campos de minivan
        minivan.pasajeros = 9;
        minivan.capacidad = 15;
        minivan.mpg = 20;
        //asignando valores a los campos de sportscar
        sportscar.pasajeros = 10;
        sportscar.capacidad = 25;
        sportscar.mpg = 30;
        //Calcular el rango asumiendo un tanque lleno
        rango1 = minivan.capacidad * minivan.mpg;
        rango2 = sportscar.capacidad * sportscar.mpg;
        System.out.println("La Minivan puede llevar " + minivan.pasajeros + " pasajeros con un rango de " +
rango1 + " millas");
        System.out.println("El Sportscar puede llevar " + sportscar.pasajeros + " pasajeros con un rango de " +
rango2 + " millas");
    }
}

```



```
//Este programa crea dos objetos Vehiculo
class Vehiculo {
    int pasajeros; //números de pasajeros
    int capacidad; //capacidad del combustible en galones
    int mpg; //combustible consumido en millas por galon
}
//Esta clase declara un objeto de tipo Vehiculo
class DosVehiculo {
    public static void main(String[] args) {
        Vehiculo minivan = new Vehiculo();
        Vehiculo sportscar = new Vehiculo();
        int rango1, rango2;
        //asignando valores a los campos de minivan
        minivan.pasajeros = 9;
        minivan.capacidad = 15;
        minivan.mpg = 20;
        //asignando valores a los campos de sportscar
        sportscar.pasajeros = 10;
        sportscar.capacidad = 25;
        sportscar.mpg = 30;
        //Calcular el rango asumiendo un tanque lleno
        rango1 = minivan.capacidad * minivan.mpg;
        rango2 = sportscar.capacidad * sportscar.mpg;
        System.out.println("La Minivan puede llevar " + minivan.pasajeros + " pasajeros con un rango de " +
rango1 + " millas");
        System.out.println("El Sportscar puede llevar " + sportscar.pasajeros + " pasajeros con un rango de " +
rango2 + " millas");
    }
}
```





Politécnico  
Internacional



PORCENTAJES DE EVALUACIÓN

*¿preguntas?*



Politécnico  
Internacional



PORCENTAJES DE EVALUACIÓN

# CONCLUSIONES



Politécnico  
Internacional

# *PROGRAMACION ORIENTADA A OBJETOS*

*“Nuestro pasado es el enemigo nº1 de la Innovación.”*

David Cánovas



# AGENDA

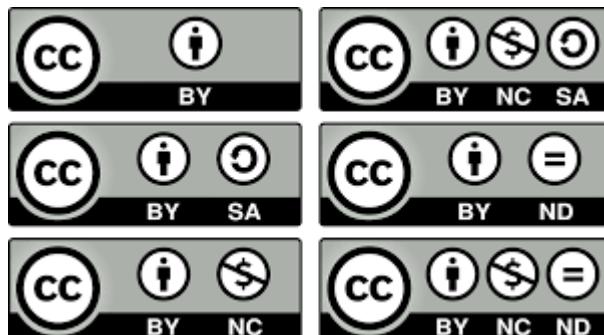
- *Retroalimentación*
- *¿Que es herencia?*
- *Tipos de herencia*
- *Ejemplo de herencia*
- *Reto Practico*
- *Conclusiones*



**FORMATO IEEE**  
<https://cutt.ly/3Z0W3pk>



**CREATIVE COMMONS**  
<https://cutt.ly/JZ0EZ9J>



SEGUNDO CORTE Formato IEEE	SEGUNDO CORTE DETALLE
Diagrama de Arquitectura	<i>De acuerdo a las diferentes propuestas de arquitectura. Adicionar el diagrama de arquitectura seleccionado (MVC + 3Capas + Cliente Servidor)</i>
Diagrama de Clases - UML	<i>De acuerdo a los parámetros de diagramas de UML adicionar diagrama de Clases.</i>
Diagrama de Secuencia - UML	<i>De acuerdo a los parámetros de diagramas de UML adicionar diagrama de Secuencia.</i>
Diccionario de Datos	<i>De acuerdo a los parámetros de diagramas de UML adicionar Diccionario de Datos.</i>
Definición de Clases, Atributos, Métodos, Objetos en el proyecto ST.	<i>Indicar de acuerdo al proyecto donde se define cada uno, listarlos y presentar código con comentarios.</i> <ul style="list-style-type: none"><li>• Clases.</li><li>• Atributos.</li><li>• Métodos.</li><li>• Objetos.</li></ul>
Definición en el proyecto ST. (Herencia – Polimorfismo – abstracción – encapsulamiento)	<i>Indicar de acuerdo al proyecto donde se define: herencia – Polimorfismo – abstracción – encapsulamiento y presentar el código con comentarios.</i>



Politécnico  
Internacional



GITHUB - ST2



[https://github.com/Harol003/StrangerThings\\_APP](https://github.com/Harol003/StrangerThings_APP)



# ¡PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS!

**ABSTRACCIÓN**  Es el proceso de **definir los atributos y los métodos** de una clase.

**ENCAPSULAMIENTO**  Protege la información de manipulaciones no autorizadas.

**POLIMORFISMO**  Da la misma orden a varios objetos para que respondan de diferentes maneras.

**HERENCIA**  Las **clases hijo heredan atributos y métodos** de las clases padre.

Según el paradigma, la programación orientada objetos, se basa en estos 4 pilares. **Estos definen la simplicidad y la funcionalidad del código.**

¿Cómo utilizas la POO actualmente?  
 [ed.team/cursos/poo](http://ed.team/cursos/poo)





Politécnico  
Internacional



# *¿Qué es herencia en POO?*



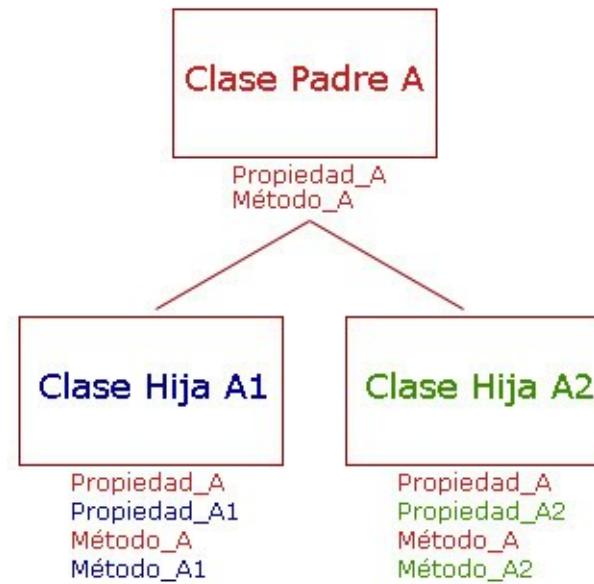
La herencia es un pilar importante de OOP (Programación Orientada a Objetos). Es el mecanismo en Java por el cual una clase permite **heredar las características** (atributos y métodos) de otra clase



En el lenguaje de Java, una clase que se hereda se denomina **superclase**. La clase que hereda se llama **subclase**. Por lo tanto, una subclase es una versión especializada de una superclase. Hereda todas las variables y métodos definidos por la superclase y agrega sus propios elementos únicos.

- **Superclase:** la clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal).
- **Subclase:** la clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos además de los campos y métodos de la superclase.
- **Reutilización:** la herencia respalda el concepto de “reutilización”, es decir, cuando queremos crear una clase nueva y ya hay una clase que incluye parte del código que queremos, podemos衍生 nuestra nueva clase de la clase existente. Al hacer esto, estamos reutilizando los campos/atributos y métodos de la clase existente.

## Herencia de Clases





Podemos definir la **HERENCIA** como la **capacidad de crear clases** que adquieren de manera automática los miembros (atributos y métodos) de otras clases que ya existen, pudiendo al mismo tiempo añadir atributos y métodos propios.

Java soporta la herencia permitiendo una clase a incorporar otra clase en su declaración. Esto se hace mediante el uso de la palabra clave **EXTENDS**. Por lo tanto, la subclase se añade (se extiende) a la superclase.

### Mantenimiento de aplicaciones existentes

Utilizando la herencia, si tenemos una clase con una determinada funcionalidad y tenemos la necesidad de ampliar dicha funcionalidad, no necesitamos modificar la clase existente (la cual se puede seguir utilizando para el tipo de programa para la que fue diseñada) sino que podemos crear una clase que herede a la primera, adquiriendo toda su funcionalidad y añadiendo la suya propia.

### Reutilización del código

En aquellos casos donde se necesita crear una clase que, además de otros propios, deba incluir los métodos definidos en otra, la herencia evita tener que reescribir todos esos métodos en la nueva clase.

La palabra clave utilizada para la herencia es **extends**

```
//Clase para objetos de dos dimensiones
class DosDimensiones{
    double base;
    double altura;
    void mostrarDimension(){
        System.out.println("La base y altura es: "+base+" y "+altura);
    }
}

//Una subclase de DosDimensiones para Triangulo
class Triangulo extends DosDimensiones{
    String estilo;
    double area(){
        return base*altura/2;
    }
    void mostrarEstilo(){
        System.out.println("Triangulo es: "+estilo);
    }
}
class Lados3{
    public static void main(String[] args) {
        Triangulo t1=new Triangulo();
        Triangulo t2=new Triangulo();
        t1.base=4.0;
        t1.altura=4.0;
        t1.estilo="Estilo 1";
        t2.base=8.0;
        t2.altura=12.0;
        t2.estilo="Estilo 2";
        System.out.println("Información para T1: ");
        t1.mostrarEstilo();
        t1.mostrarDimension();
        System.out.println("Su área es: "+t1.area());
        System.out.println();
        System.out.println("Información para T2: ");
        t2.mostrarEstilo();
        t2.mostrarDimension();
        System.out.println("Su área es: "+t2.area());
    }
}
```

## SALIDA

```
Información para T1:
Triangulo es: Estilo 1
La base y altura es: 4.0 y 4.0
Su área es: 8.0
Información para T2:
Triangulo es: Estilo 2
La base y altura es: 8.0 y 12.0
Su área es: 48.0
```

```
//Clase para objetos de dos dimensiones
class DosDimensiones{
    private double base;
    private double altura;
    void mostrarDimension(){
        System.out.println("La base y altura es: "+base+" y "+altura);
    }
}

//Una subclase de DosDimensiones para Triangulo
class Triangulo extends DosDimensiones{
    String estilos;
    double area(){
        return base*altura/2; //Error! no se puede acceder
    }
    void mostrarEstilo(){
        System.out.println("Triangulo es: "+estilo);
    }
}
```

La base y la altura se vuelven privados en **DosDimensiones**, entonces Triangulo no podrá acceder a ellos.

```
//Clase para objetos de dos dimensiones
class DosDimensiones{
    private double base;
    private double altura;
    //Métodos de acceso para base y altura
    double getBase(){return base;}
    double getAltura(){return altura;}
    void setBase(double b){base=b;}
    void setAltura (double h){altura=h;}
    void mostrarDimension(){
        System.out.println("La base y altura es: "+base+" y "+altura);
    }
}
```

Reescritura de las clases *DosDimensiones* y *Triangulo* que usa métodos para acceder a las variables de instancia privadas *base* y *altura*

```
//Una subclase de DosDimensiones para Triangulo
class Triangulo extends DosDimensiones{
    String estilos;
    double area(){
        return getBase()*getAltura()/2;
    }
    void mostrarEstilo(){
        System.out.println("Triangulo es: "+estilos);
    }
}
class Lados3{
    public static void main(String[] args) {
        Triangulo t1=new Triangulo();
        Triangulo t2=new Triangulo();
        t1.setBase(4.0);
        t1.setAltura(4.0);
        t1.estilos="Estilo 1";
        t2.setBase(8.0);
        t2.setAltura(12.0);
        t2.estilos="Estilo 2";
        System.out.println("Información para T1: ");
        t1.mostrarEstilo();
        t1.mostrarDimension();
        System.out.println("Su área es: "+t1.area());
        System.out.println();
        System.out.println("Información para T2: ");
        t2.mostrarEstilo();
        t2.mostrarDimension();
        System.out.println("Su área es: "+t2.area());
    }
}
```



```
//Clase para objetos de dos dimensiones
class DosDimensiones{
    private double base;
    private double altura;
    //Métodos de acceso para base y altura
    double getBase(){return base;}
    double getAltura(){return altura;}
    void setBase(double b){base=b;}
    void setAltura (double h){altura=h;}
    void mostrarDimension(){
        System.out.println("La base y altura es: "+base+" y "+altura);
    }
}
```

Reescritura de las clases *DosDimensiones* y *Triangulo* que usa métodos para acceder a las variables de instancia privadas *base* y *altura*

```
//Una subclase de DosDimensiones para Triangulo
class Triangulo extends DosDimensiones{
    String estilos;
    double area(){
        return getBase()*getAltura()/2;
    }
    void mostrarEstilo(){
        System.out.println("Triangulo es: "+estilos);
    }
}
class Lados3{
    public static void main(String[] args) {
        Triangulo t1=new Triangulo();
        Triangulo t2=new Triangulo();
        t1.setBase(4.0);
        t1.setAltura(4.0);
        t1.estilos="Estilo 1";
        t2.setBase(8.0);
        t2.setAltura(12.0);
        t2.estilos="Estilo 2";
        System.out.println("Información para T1: ");
        t1.mostrarEstilo();
        t1.mostrarDimension();
        System.out.println("Su área es: "+t1.area());
        System.out.println();
        System.out.println("Información para T2: ");
        t2.mostrarEstilo();
        t2.mostrarDimension();
        System.out.println("Su área es: "+t2.area());
    }
}
```



Politécnico  
Internacional



SEGUNDO CORTE



*Puesta en Marcha !!!*

# RETOS PENDIENTES

- **Taller 1** – *Codificar 24 ejercicios propuestos.*
- **Taller 2 Arreglos** – *Array 10 ejercicios propuestos.*
- **Reto 2** – *Codificar 9 ejercicios propuestos.*
- **Reto 3** – *Codificar Clase Atributo Metodo Objeto Stranger Things*
- **Reto 4** – *Codificar Herencia Stranger Things*



Politécnico  
Internacional



PORCENTAJES DE EVALUACIÓN

*¿preguntas?*



Politécnico  
Internacional



PORCENTAJES DE EVALUACIÓN

# CONCLUSIONES



Politécnico  
Internacional



# ANTES DE EMPEZAR

**“Un emprendedor ve oportunidades allá donde otros solo ven problemas”**



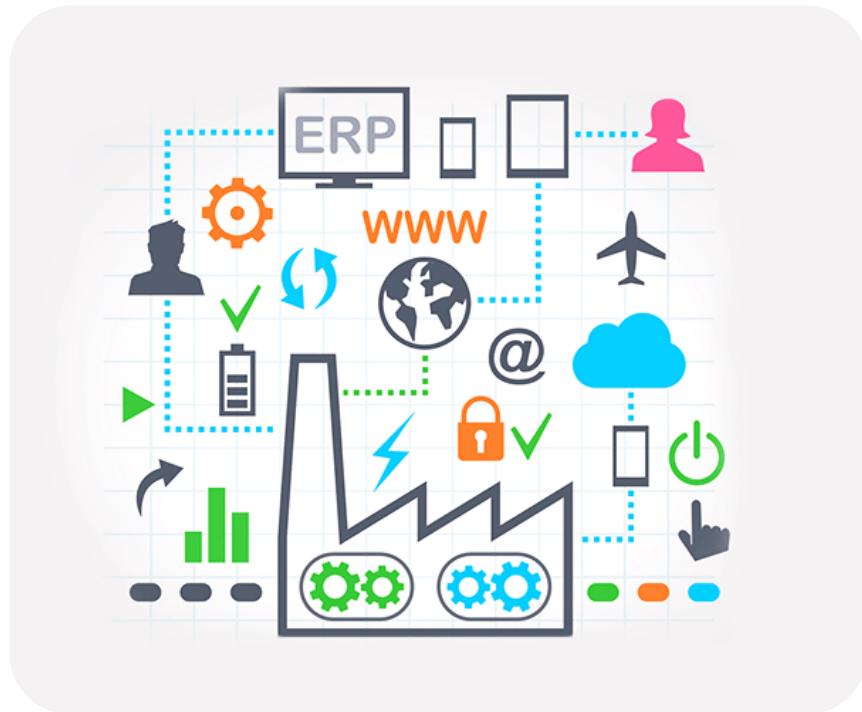
Michael Gerber



# AGENDA

- *Retroalimentación - Clase Anterior*
- *Taller Número 2*
- *Tipos de Datos*
- *Operadores*
- *Preguntas*
- *Conclusiones*





1 Sábados  
2 Break

8 AM a 12 M - 4 Horas  
9:30 AM a 09:45 AM

## ESPACIOS DE FORMACIÓN

# A TENER EN CUENTA



1. Cerrar los micrófonos y desactivar las cámaras.



2. Solicitar la palabra mediante el chat.



3. Preguntar

FUNDAMENTOS DE PROGRAMACIÓN



## CONTENIDOS

1. Algoritmos
2. Elementos lógicos de un algoritmo
3. Técnicas para la formulación de algoritmos. DFD, pseudocódigo, prueba de escritorio.
4. Introducción al Lenguaje de programación C#
5. Historia, evolución y futuro del lenguaje de programación C#
6. Introducción a la sintaxis de C#
7. Tipos de datos en C#
8. Operadores en C#
9. Estructuras de control en C#
10. Arreglos en C#
11. Manejo de excepciones en C#
12. Colecciones en C# (Espacio de nombres System.Collections)
13. LinQ con C#

# CONTENIDOS



# RETROALIMENTACIÓN

## CLASE ANTERIOR

- *Introducción al lenguaje de programación JAVA*
- *Historia, Evolución y futuro del lenguaje de programación JAVA.*
- *Introducción a la sintaxis de Java*



# FORO DE CONOCIMIENTOS

*FORO de conocimientos de la clase anterior  
Se determinan los contenidos respecto a:*

- *Historia de java*
- *Características de Java / Sintaxis.*

*Aula Virtual*



<https://cutt.ly/1fbX9Nk>

## ARCHIVOS DE TRABAJO



Archivos con las temáticas presentadas



Archivo de texto con las URL para desarrollar las actividades en clase.



# RETO



Sobre nosotros    Padres    Impacto    Mis clases    Mi Cuenta ▾    español

## CIDE\_FUNDAMENTOS [editar la configuración de la clase](#)

FUNDAMENTOS\_PROGRAMACION

Resumen de la clase

Idioma : JavaScript

Estudiantes : 0

Tiempo de juego de nivel medio :

Tiempo total de juego :

Niveles promedio completados :

Niveles totales completados :

Creado : 14/8/2020

Agregar estudiantes :

**NightShirtMilk**

Copiar código de clase

Los estudiantes pueden unirse a su clase usando este Código de clase. No se requiere una dirección de correo electrónico al crear una cuenta de estudiante con este código de clase.

<https://codecombat.com/stude>

Copiar URL de clase

También puede publicar esta URL de clase única en una página web compartida.



# RETO

## FLEXBOX FROGGY

Nivel 1 de 24

Bienvenido a Flexbox Froggy, un juego donde ayudarás a Froggy ya sus amigos escribiendo código CSS. Guía a esta rana hacia la hoja de lirio en la derecha, usando la propiedad **justify-content**, la cual alinea elementos horizontalmente y acepta los siguientes valores:

- **flex-start**: Alinea elementos al lado izquierdo del contenedor.
- **flex-end**: Alinea elementos al lado derecho del contenedor.
- **center**: Alinea elementos en el centro del contenedor.
- **space-between**: Muestra elementos con la misma distancia entre ellos.
- **space-around**: Muestra elementos con la misma separación alrededor de ellos.

Por ejemplo, **justify-content: flex-end;** moverá la rana a la derecha.

```
1 #pond {  
2   pantalla: flex;  
3   justify-content: flex-end;  
4 }  
5  
6
```





## TIPOS DE LENGUAJE



- *Lenguajes Tipo Estático*
- *Lenguajes Tipo Dinámico*



# LENGUAJE TIPADO ESTÁTICO

```
void co
for (It
    if
    if(
        if(
            if(RU
                if(RU
                    final Synt
                    final Cl
                    final C

```



*Cada variable y tipo de expresión ya se conoce en tiempo de compilación. Una vez que se declara una variable es de un cierto tipo de datos, no puede contener valores de otros tipos de datos.*

*Se dice de un lenguaje de programación que usa un tipado estático cuando la comprobación de tipificación se realiza durante la compilación, y no durante la ejecución*



## LENGUAJE TIPADO DINÁMICO



*Estos lenguajes pueden recibir diferentes tipos de datos a lo largo del tiempo.  
Ejemplo: Ruby, Python*

*Un lenguaje de programación dinámico es un lenguaje de programación en el que las operaciones realizadas en tiempo de compilación pueden realizarse en tiempo de ejecución*



# TIPADO ESTATICO **JAVA**

JAVA está tipado **ESTÁTICAMENTE** y es fuertemente tipado porque en Java, cada tipo de datos (como entero, carácter, hexadecimal, decimal empaquetado, etc.) está predefinido como parte del lenguaje de programación y todas las constantes o variables definidas para un programa dado debe describirse con uno de los tipos de datos.



PRIMITIVOS  
&  
OBJETO

# CATEGORIAS DE DATOS

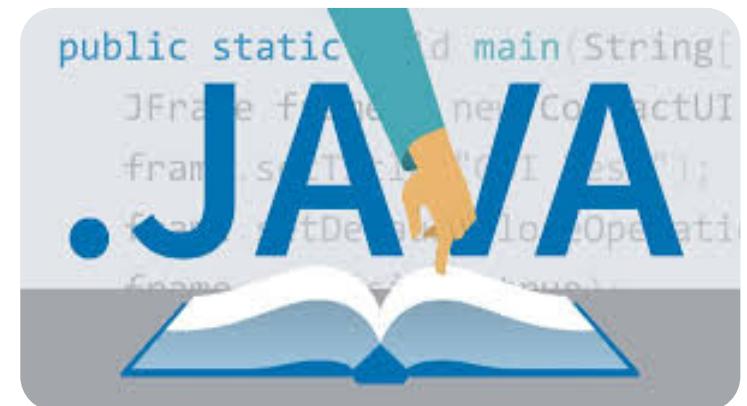




# CATEGORIAS DE DATOS

## DATOS PRIMITIVOS

*Los datos primitivos son solo valores únicos; ellos no tienen capacidades especiales. Java soporta 8 tipos de datos primitivos*





# CATEGORIAS DE DATOS

## DATOS PRIMITIVOS

**NUMÉRICOS ENTEROS:** Son los tipos *byte*, *short*, *int* y *long*. Los 4 representan números enteros con signo.

**CARÁCTER:** El tipo *CHAR* representa un carácter codificado en el sistema unicode.





# CATEGORIAS DE DATOS

## DATOS PRIMITIVOS

**NUMÉRICO DECIMAL:** Los tipos **FLOAT** y **DOUBLE** representan números decimales en coma flotante.

**LÓGICOS:** El tipo **BOOLEAN** es el tipo de dato lógico; los dos únicos posibles valores que puede representar un dato lógico son **true** y **false**. **true** y **false** son palabras reservadas de Java.





TIPO	DESCRIPCIÓN	DEFAULT	TAMAÑO	EJEMPLOS
boolean	true o false	false	1 bit	true, false
byte	entero complemento de dos	0	8 bits	100, -50
char	carácter unicode	\u0000	16 bits	'a', '\u0041', '\101', '\U'
short	entero complemento de dos	0	16 bits	10000,-20000
int	entero complemento de dos	0	32 bits	100000,-2,-1,0,1,2,-200000
long	entero complemento de dos	0	64 bits	-2L,-1L,0L,1L,2L
float	coma flotante IEEE 754	0.0	32 bits	1.23e100f, -1.23e-100f, .3ef, 3.14f
double	coma flotante IEEE 754	0.0	64 bits	1.2345e300d, -1.2345e-300f, 1e1d



### TIPO DE DATO BOOLEAN

*El tipo de datos booleano representa solo un bit de información: true (verdadero) o false (falso). Los valores de tipo booleano no se convierten implícita o explícitamente (con casts) en ningún otro tipo*

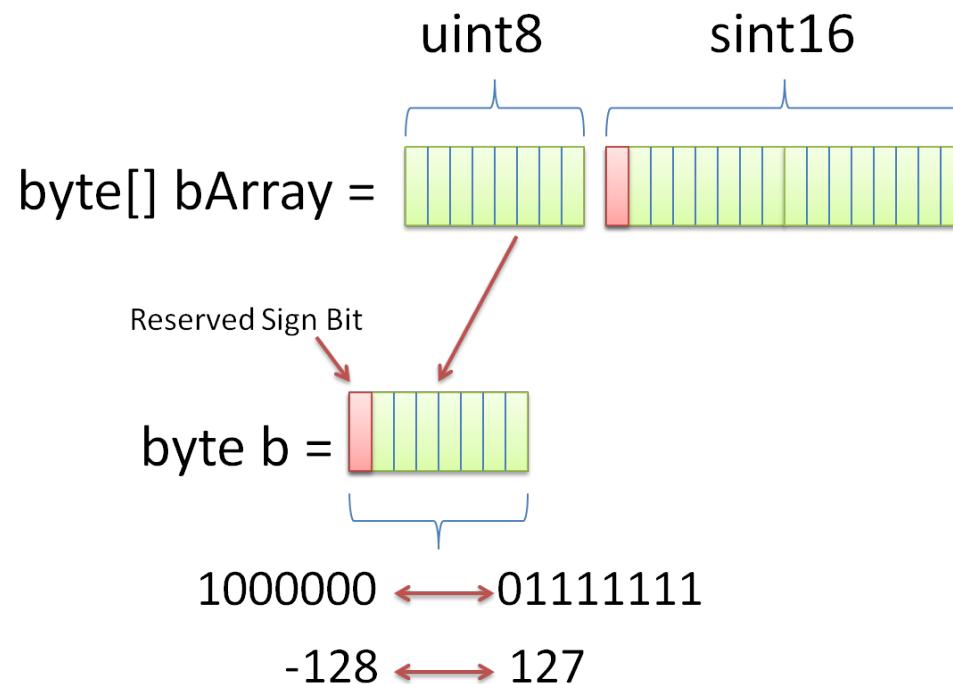
True or False:

True

False



Boolean is a  
Data Type



### TIPO DE DATO BYTE

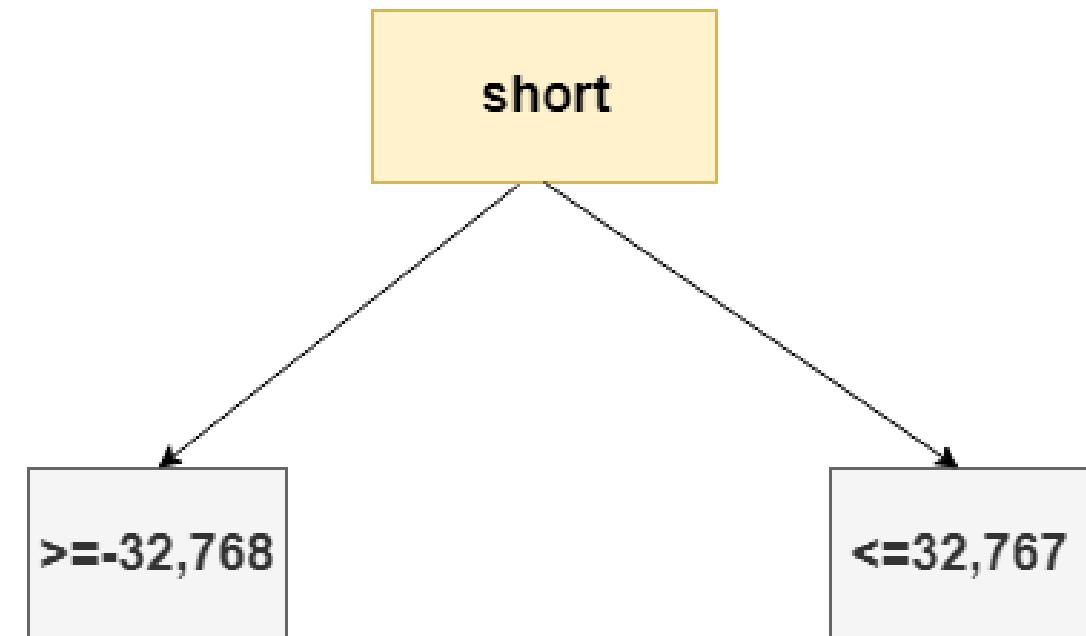
*El tipo de datos byte es un entero de 8 bits de complemento a dos (una forma de representar datos positivos y negativos en binario). El tipo de datos de byte es útil para guardar en memoria grandes arrays.*

**Tamaño:** 8 bits  
**Valor:** -128 a 127



### TIPO DE DATO SHORT

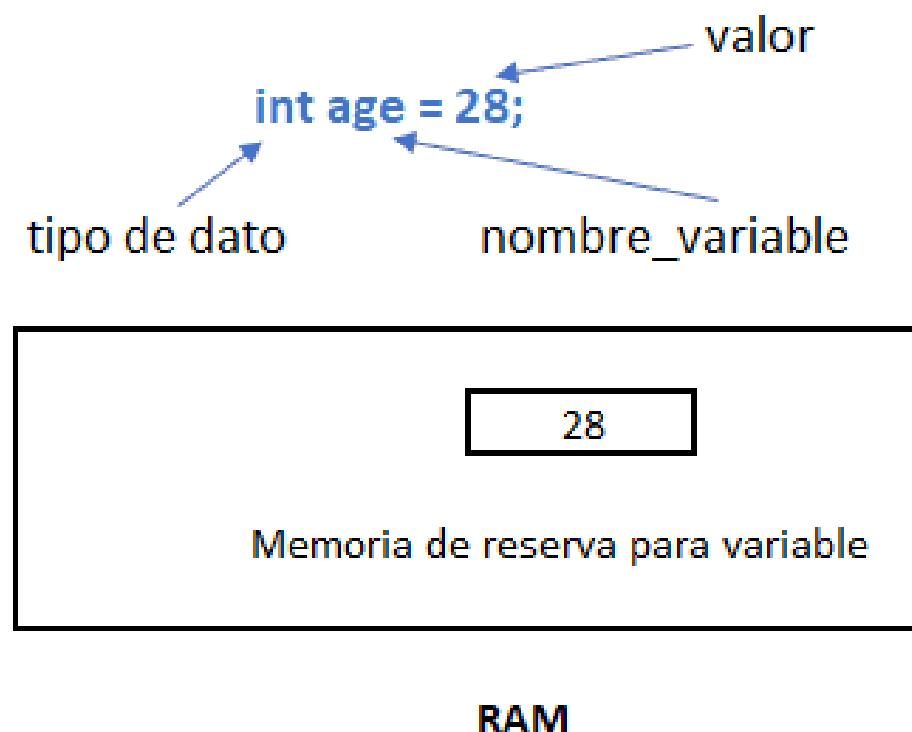
*El tipo de datos **SHORT** es un entero de complemento de dos formado por 16 bits. De forma similar al byte, use short para guardar en memoria arrays grandes, en situaciones donde el ahorro de memoria realmente importa.*





# CATEGORIAS DE DATOS

## DATOS PRIMITIVOS



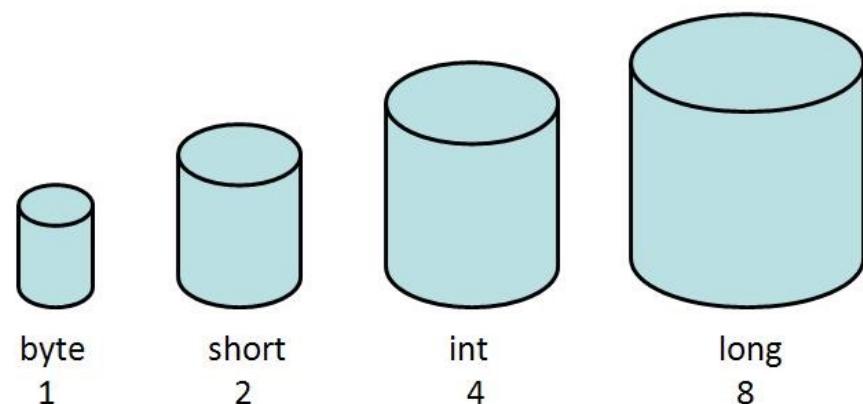
### TIPO DE DATO INT

*Es un entero de complemento de dos formado de 32 bits.*

**Tamaño:** 32 bits

**Valor:**  $(-2^{31})$  a  $(2^{31} - 1)$

*En Java SE 8 y posterior, podemos usar el tipo de datos int para representar un entero de 32 bits sin signo, que tiene un valor en el rango  $[0, 2^{32} - 1]$ . Utilice la clase INTEGER para usar el tipo de datos int como un entero sin signo.*



### TIPO DE DATO LONG

*El tipo de datos **LONG** es un entero de complemento de dos de 64 bits..*

**Tamaño:** 64 bit

**Valor:**  $(-2^{63})$  a  $(2^{63}-1)$

*En Java SE 8 y posteriores, puede usar el tipo de datos Long para representar un Long sin signo de 64 bits, que tiene un valor mínimo de 0 y un valor máximo de  $2^{64} - 1$ . La clase Long también contiene métodos como **COMPAREUNSIGNED**, **DIVIDEUNSIGNED**, etc. Para admitir operaciones aritméticas.*



### TIPO DE DATO FLOAT - DOUBLE

*El tipo de dato float es una coma flotante IEEE 754 de precisión simple de 32 bits. Puede usar float (en lugar double) si necesita guardar en memoria grandes arrays de números de coma flotante.*

**Tamaño: 32 bits**

**Sufijo: F/f Ejemplo: 9.8f**

*El tipo de dato double es una coma flotante IEEE 754 de 64 bits de doble precisión. Para valores decimales, este tipo de datos generalmente es la opción predeterminada.*

64bit = double, double precision



32bit = float, single precision



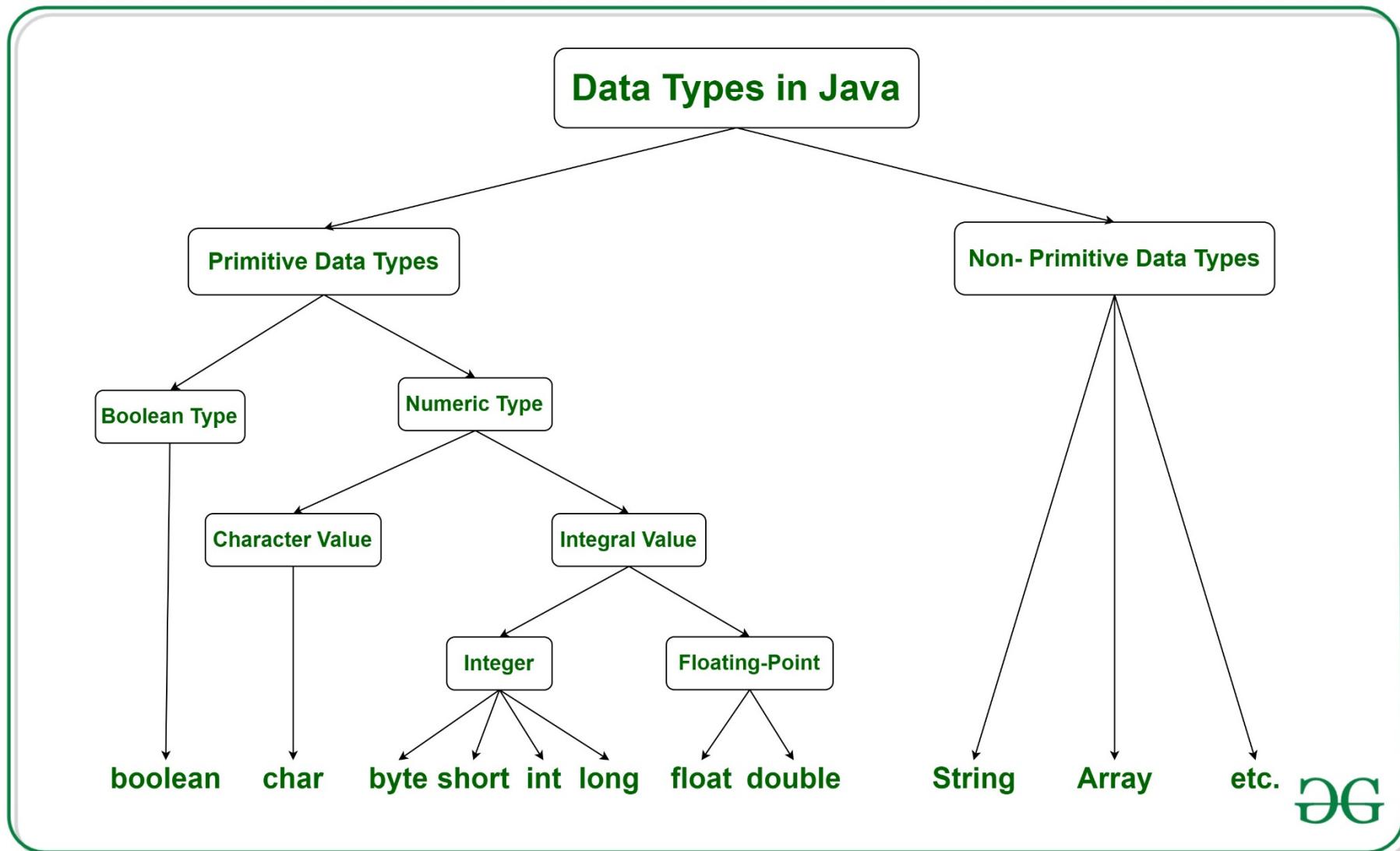
Signed bit

Exponent

Significand

16bit = half, half precision







# **RETO PRACTICO**

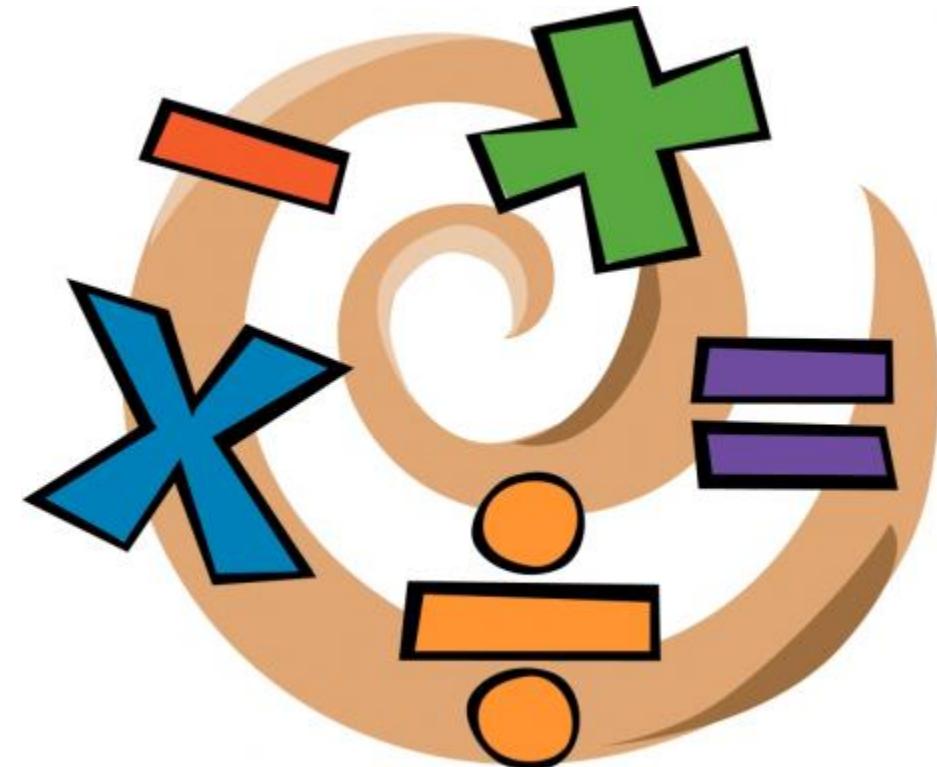
## **Taller 1**



## CLASIFICACION OPERADORES JAVA

*Java proporciona muchos tipos de operadores que se pueden usar según la necesidad. Se clasifican según la funcionalidad que brindan. Algunos de los tipos son los siguientes:*

- Operadores aritméticos
- Operadores unarios
- Operador de asignación





```
// Programa Java para ilustrar
// operadores aritméticos
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        String x = "Thank", y = "You";

        // Operador + y -
        System.out.println("a + b = "+(a + b));
        System.out.println("a - b = "+(a - b));

        // El operador + si se usa con strings
        // concatena las cadenas dadas.
        System.out.println("x + y = "+x + y);

        // Operador * y /
        System.out.println("a * b = "+(a * b));
        System.out.println("a / b = "+(a / b));

        // operador de módulo da el resto
        // de dividir el primer operando con el segundo
        System.out.println("a % b = "+(a % b));

        // si el denominador es 0 en la división
        // System.out.println(a/c);
        // lanzaría una java.lang.ArithmeticException

    }
}
```

## JAVA + OPERADORES ARITMÉTICOS

*Se utilizan para realizar operaciones aritméticas simples en tipos de datos primitivos.*

*\*: Multiplicación*

*/: División*

*%: Modulo*

*+: Adición*

*-: Resta*



# OPERADORES UNARIOS

## DATOS PRIMITIVOS

```
// Programa Java para ilustrar
// operadores unarios
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;

        // operador de pre-incremento
        // a = a+1 y entonces c = a;
        c = ++a;
        System.out.println("Valor de c (++a) = " + c);

        // operador de post-incremento
        // c=b entonces b=b+1 (b pasa a ser 11)
        c = b++;
        System.out.println("Valor de c (b++) = " + c);

        // operador de pre-decrecimiento
        // d=d-1 entonces c=d
        c = --d;
        System.out.println("Valor de c (--d) = " + c);

        // operador de post-decrecimiento
        // c=e entonces e=e-1 (e pasa a ser 39)
        c = e--;
        System.out.println("Valor de c (e--) = " + c);

        // Operador lógico not
        System.out.println("Valor de !condition = " + !condition);
    }
}
```

## JAVA + OPERADORES UNARIOS

*Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.*

*-: Unario menos, utilizado para negar los valores.*

*+: Unario más, usado para dar valores positivos. Solo se usa cuando se convierte deliberadamente un valor negativo en positivo.*

*++: Operador de incremento, utilizado para incrementar el valor en 1. Hay dos variedades de operador de incremento.*

*Pre-Incremento: el valor se incrementa primero y luego se calcula el resultado.*



# OPERADORES UNARIOS

## DATOS PRIMITIVOS

```
// Programa Java para ilustrar
// operadores unarios
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;

        // operador de pre-incremento
        // a = a+1 y entonces c = a;
        c = ++a;
        System.out.println("Valor de c (++a) = " + c);

        // operador de post-incremento
        // c=b entonces b=b+1 (b pasa a ser 11)
        c = b++;
        System.out.println("Valor de c (b++) = " + c);

        // operador de pre-decremento
        // d=d-1 entonces c=d
        c = --d;
        System.out.println("Valor de c (--d) = " + c);

        // operador de post-decremento
        // c=e entonces e=e-1 (e pasa a ser 39)
        c = e--;
        System.out.println("Valor de c (e--) = " + c);

        // Operador lógico not
        System.out.println("Valor de !condition = " + !condition);
    }
}
```

## JAVA + OPERADORES UNARIOS

*Los operadores unarios solo necesitan un operando. Se usan para incrementar, disminuir o negar un valor.*

*Post-Incremento: el valor se usa por primera vez para calcular el resultado y luego se incrementa.*

*—: Operador de decremento , usado para incrementar el valor en 1. Hay dos variedades de operador de incremento.*

*Pre-Decremento: el valor se disminuye primero y luego se calcula el resultado.*

*Post-Decremento: el valor se usa por primera vez para calcular el resultado y luego se disminuye.*

*! : Operador lógico “no”, utilizado para invertir un valor booleano.*



```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);

        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

## JAVA + OPERADORES ASIGNACIÓN

*El operador de asignación se usa para asignar un valor a cualquier variable. Tiene una asociación de derecha a izquierda, es decir, el valor dado en el lado derecho del operador se asigna a la variable de la izquierda y, por lo tanto, el valor del lado derecho debe declararse antes de usarlo o debe ser una constante.*

*El formato general del operador de asignación es, [java]variable = valor;[/java]*



```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

## JAVA + OPERADORES ASIGNACIÓN

*En muchos casos, el operador de asignación se puede combinar con otros operadores para construir una versión más corta de la declaración llamada **Declaración Compuesta (Compound Statement)**. Por ejemplo, en lugar de  $a = a + 5$ , podemos escribir  $a += 5$ .*

```
int a = 5;
a += 5; // a = a + 5;
```



# OPERADORES ASIGNACIÓN

## DATOS PRIMITIVOS

```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

## JAVA + OPERADORES ASIGNACIÓN

*+ = , para sumar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*

*- = , para restar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*

- \* = , para multiplicar el operando izquierdo con el operando derecho y luego asignándolo a la variable de la izquierda.*



# OPERADORES ASIGNACIÓN

## DATOS PRIMITIVOS

```
// Programa Java para ilustrar
// Operadores de Asignación
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;

        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;

        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + ","
                           + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;

        // operadores de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " +
                           a + "," + b + "," + e + "," + f);
    }
}
```

## JAVA + OPERADORES ASIGNACIÓN

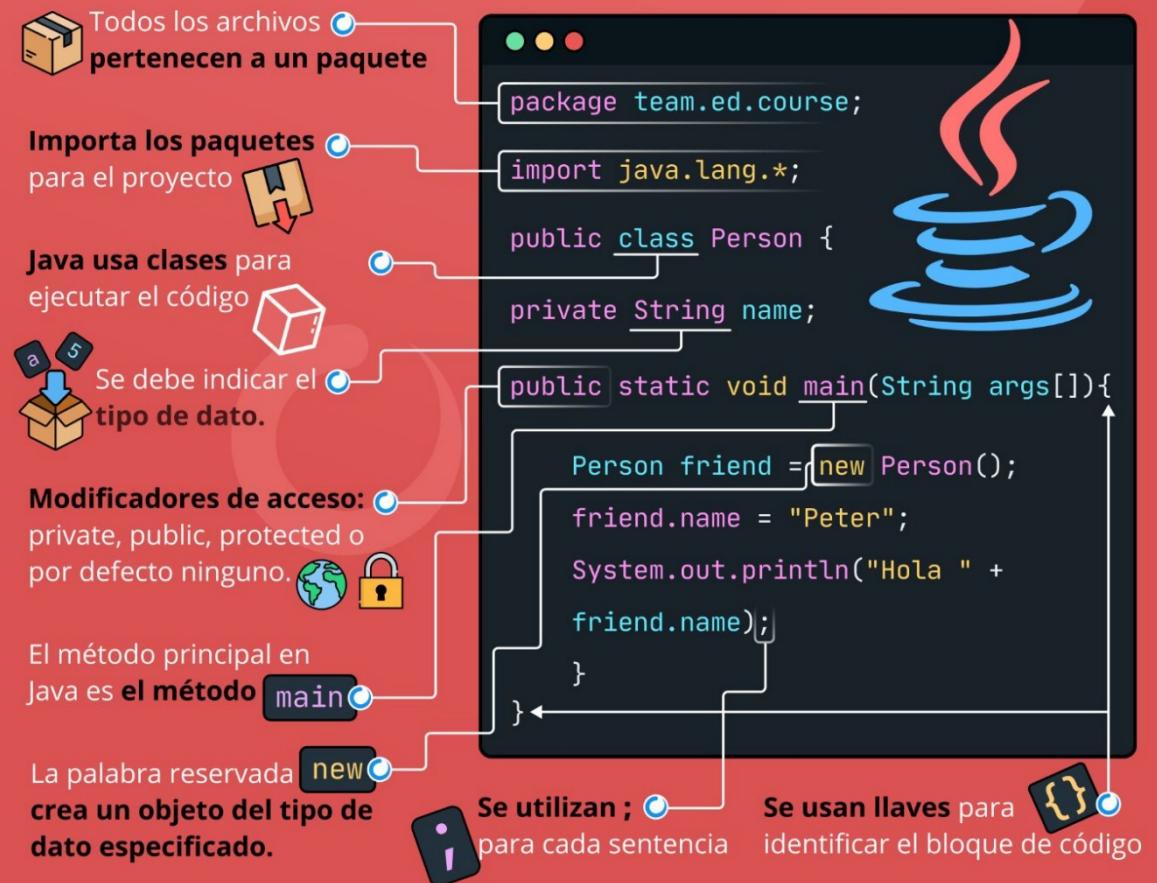
*/= , para dividir el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*

*^ = , para aumentar la potencia del operando izquierdo al operando derecho y asignarlo a la variable de la izquierda.*

*% = , para asignar el módulo del operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.*



# SINTAXIS BÁSICA DE JAVA



Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/java](http://ed.team/java)

 EDteam



# 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones creada para resolver una necesidad específica.**



## 1 GOOGLE GUAVA



### Mejora del flujo de trabajo

destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

## 2 RXJAVA



Crea aplicaciones para **reaccionar** a los flujos de datos en tiempo real.

## 3 MPANDROIDCHART



Crea **gráficos** de líneas, barras, radares y burbujas para Android.

## 4 FASTJSON



Convierte **objetos** Java en su representación JSON y viceversa.

## 5 MOCKITO



Librería de código abierto para **simular pruebas unitarias**.



## 6 JUNIT



Esta es la librería más **usada para pruebas**.

Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/cursos/java](http://ed.team/cursos/java)



# **RETO PRACTICO**

# **INSTALACIÓN**



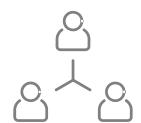
# CONCLUSIONES

.





**¿ PREGUNTAS ?**



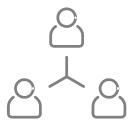
# Gracias



Politécnico  
Internacional



“Una máquina puede hacer el trabajo de cincuenta hombres ordinarios. Ninguna máquina puede hacer el trabajo de un **hombre extraordinario**”



ELBERT HUBBARD – Escritor.

# ANTES DE EMPEZAR



# AGENDA

- *Retroalimentación - Clase Anterior*
- *Actividades Pendientes*
- *Palabras Reservadas*
- *Convenciones de Nomenclatura*
- *Clases Predefinidas*
- *Declaración y tipos de variables*
- *Preguntas*
- *Conclusiones*
- 





1 Sábados  
2 Break

8 AM a 12 M - 4 Horas  
9:30 AM a 09:45 AM

## ESPACIOS DE FORMACIÓN

# A TENER EN CUENTA



1. Cerrar los micrófonos y desactivar las cámaras.



2. Solicitar la palabra mediante el chat.



3. Preguntar

FUNDAMENTOS DE PROGRAMACIÓN



## CONTENIDOS

1. Algoritmos
2. Elementos lógicos de un algoritmo
3. Técnicas para la formulación de algoritmos. DFD, pseudocódigo, prueba de escritorio.
4. Introducción al Lenguaje de programación C#
5. Historia, evolución y futuro del lenguaje de programación C#
6. Introducción a la sintaxis de C#
7. Tipos de datos en C#
8. Operadores en C#
9. Estructuras de control en C#
10. Arreglos en C#
11. Manejo de excepciones en C#
12. Colecciones en C# (Espacio de nombres System.Collections)
13. LinQ con C#

# CONTENIDOS



# RETROALIMENTACIÓN: CLASE ANTERIOR

- Tipos de Datos
- Operadores

TIPO	DESCRIPCIÓN	DEFAULT	TAMAÑO	EJEMPLOS
boolean	true o false	false	1 bit	true, false
byte	entero complemento de dos	0	8 bits	100, -50
char	carácter unicode	\u0000	16 bits	'a', '\u0041', '\101', '\'
short	entero complemento de dos	0	16 bits	10000,-20000
int	entero complemento de dos	0	32 bits	100000,-2,-1,0,1,2,-200000
long	entero complemento de dos	0	64 bits	-2L,-1L,0L,1L,2L
float	coma flotante IEEE 754	0.0	32 bits	1.23e100f, -1.23e-100f, .3ef, 3.14f
double	coma flotante IEEE 754	0.0	64 bits	1.2345e300d, -1.2345e-300f, 1e1d

```
int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
String x = "Thank", y = "You";

// Operador + y -
System.out.println("a + b = "+(a + b));
System.out.println("a - b = "+(a - b));

// El operador + si se usa con strings
// concatena las cadenas dadas.
System.out.println("x + y = "+x + y);

// Operador * y /
System.out.println("a * b = "+(a * b));
System.out.println("a / b = "+(a / b));

// operador de módulo da el resto
// de dividir el primer operando con el segundo
System.out.println("a % b = "+(a % b));

// si el denominador es 0 en la división
// System.out.println(a/c);
// lanzaría una java.lang.ArithmetricException
```



## FUNDAMENTOS DE PROGRAMACION

Página Principal / Mis cursos / FUNDAMENTOS DE PROGRAMACION / Unidad 2 / FORO: TIPOS DE DATOS + OPERADORES



## FORO DE CONOCIMIENTOS

*FORO de conocimientos de la clase anterior  
Se determinan los contenidos respecto a:*

- *Tipos de Datos*
- *Operadores Java.*

*Aula Virtual*



<https://cutt.ly/1fbX9Nk>

## ARCHIVOS DE TRABAJO



Archivos con las temáticas presentadas



Archivo de texto con las URL para desarrollar las actividades en clase.



# RETO



Sobre nosotros    Padres    Impacto    Mis clases    Mi Cuenta ▾    español

## CIDE\_FUNDAMENTOS [editar la configuración de la clase](#)

FUNDAMENTOS\_PROGRAMACION

Resumen de la clase

Idioma : JavaScript

Estudiantes : 0

Tiempo de juego de nivel medio :

Tiempo total de juego :

Niveles promedio completados :

Niveles totales completados :

Creado : 14/8/2020

Agregar estudiantes :

**NightShirtMilk**

Copiar código de clase

Los estudiantes pueden unirse a su clase usando este Código de clase. No se requiere una dirección de correo electrónico al crear una cuenta de estudiante con este código de clase.

<https://codecombat.com/stude>

Copiar URL de clase

También puede publicar esta URL de clase única en una página web compartida.



# RETO

## FLEXBOX FROGGY

Nivel 1 de 24

Bienvenido a Flexbox Froggy, un juego donde ayudarás a Froggy ya sus amigos escribiendo código CSS. Guía a esta rana hacia la hoja de lirio en la derecha, usando la propiedad **justify-content**, la cual alinea elementos horizontalmente y acepta los siguientes valores:

- **flex-start**: Alinea elementos al lado izquierdo del contenedor.
- **flex-end**: Alinea elementos al lado derecho del contenedor.
- **center**: Alinea elementos en el centro del contenedor.
- **space-between**: Muestra elementos con la misma distancia entre ellos.
- **space-around**: Muestra elementos con la misma separación alrededor de ellos.

Por ejemplo, **justify-content: flex-end;** moverá la rana a la derecha.

```
1 #pond {  
2   pantalla: flex;  
3   justify-content: flex-end;  
4 }  
5  
6
```





# ACTIVIDADES PENDIENTES

## TALLER ALGORITMOS II

Fase de envío

## FUNDAMENTOS PROGRAMACIÓN



 Versión para impresión

A continuación se presenta la construcción de un glosario de forma colaborativa. La idea es definir cada uno de los conceptos desarrollados en clase. Cada estudiante debe definir y/o ampliar tres conceptos desarrollados entre estos se encuentran:

- Algoritmo
- Diagrama de Flujo
- Lengua de programación

Por favor recuerden trabajar de forma colaborativa.

Saludos.



# ACTIVIDADES PENDIENTES

## TALLER TIPOS DE DATOS

A continuación se abre el espacio para subir el taller de tipos de datos en JAVA. Por favor subir el archivo de acuerdo a los parámetros establecidos.

Saludos.

## INSTALADORES IDE JAVA

Buen día,

Estimados adjunto los instaladores para NetBeans - JDK para desarrollar los ejercicios propuestos. Por favor realizar la instalacion en sus equipos. Luego ejecutar la primera interacción con "Hola Mundo".

Saludos.

## INSTALACIÓN NETBEANS

A continuación por favor suba las pantallas confirmando la instalación de NetBeans + JDK para el desarrollo de las actividades propuestas en la clase.

Por favor adjuntar las imágenes en un único archivo en formato .PDF

Gracias.



## PALABRAS RESERVADAS



*En los lenguajes de programación, los identificadores (como su nombre lo indica) se utilizan con fines de identificación. En Java, un identificador puede ser un nombre de clase, un nombre de método o un nombre de variable.*



```
public class Test
{
    public static void main(String[] args)
    {
        int a = 28;
    }
}
```

# PALABRAS RESERVADAS



## IDENTIFICADORES

*TEST: nombre de clase.*

*MAIN: nombre del método.*

*STRING: nombre de clase predefinido.*

*ARGS: nombre de la variable.*

*A: nombre de la variable.*

# REGLAS IDENTIFICADORES JAVA

```
void co    tFile(final SyntaxNod    n) throws CodeExcepti
for (It    or ite=sn.getChil    dren(ite) {
    fin    SyntaxNode cn = ite.createI
    fin    Rule rule = cn.getRule();
    if(    E_PACKAGE==rule)
        back = cn.getCh
    }el    if(RULE_IMPORT==rule){
        /TODO handle st
        final SyntaxNode cn = cn.getChil
        final Cn s fullN
        final Cn s[] par
        ByRule(RULE_IMPO
```



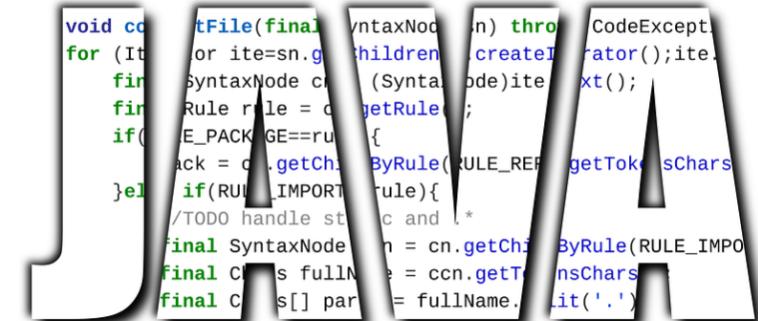
Los únicos caracteres permitidos para los identificadores son todos los caracteres alfanuméricos ([AZ], [az], [0-9]), “\$” (signo de dólar) y ‘\_’ (guión bajo). Por ejemplo, “java@” no es un identificador de Java válido ya que contiene “@” – carácter especial.



*Los identificadores no deben comenzar con dígitos ([0-9]). Por ejemplo, “123java” no es un identificador de Java válido.*

*Los identificadores de Java distinguen entre mayúsculas y minúsculas.*

# REGLAS IDENTIFICADORES JAVA



# REGLAS IDENTIFICADORES JAVA

```
void codeFile(final SyntaxNode cn) throws CodeException {
    for (Iterator ite = cn.getChildren(); ite.hasNext(); ) {
        final SyntaxNode child = (SyntaxNode) ite.next();
        final Rule rule = child.getRule();
        if (rule.getE_PACKAGE == rule) {
            back = child.getChildren();
        } else if (rule.getE_IMPORT == rule) {
            // TODO handle static imports
            final SyntaxNode ccn = child.getChildren();
            final Class fullN = ccn.getT();
            final Class[] pars = fullN.getParams();
            ByRule(RULE_IMPORT).getTokensChars(ccn);
        }
    }
}
```



No hay límite en la longitud del identificador, pero es aconsejable usar solamente una longitud óptima de 4 a 15 caracteres.



# EJEMPLOS IDENTIFICADORES JAVA

## BIEN DEFINIDOS

```
MyVariable  
MYVARIABLE  
myvariable  
x  
i  
x1  
i1  
_myvariable  
$myvariable  
sum_of_array  
javadesdecero
```

## MAL DEFINIDOS

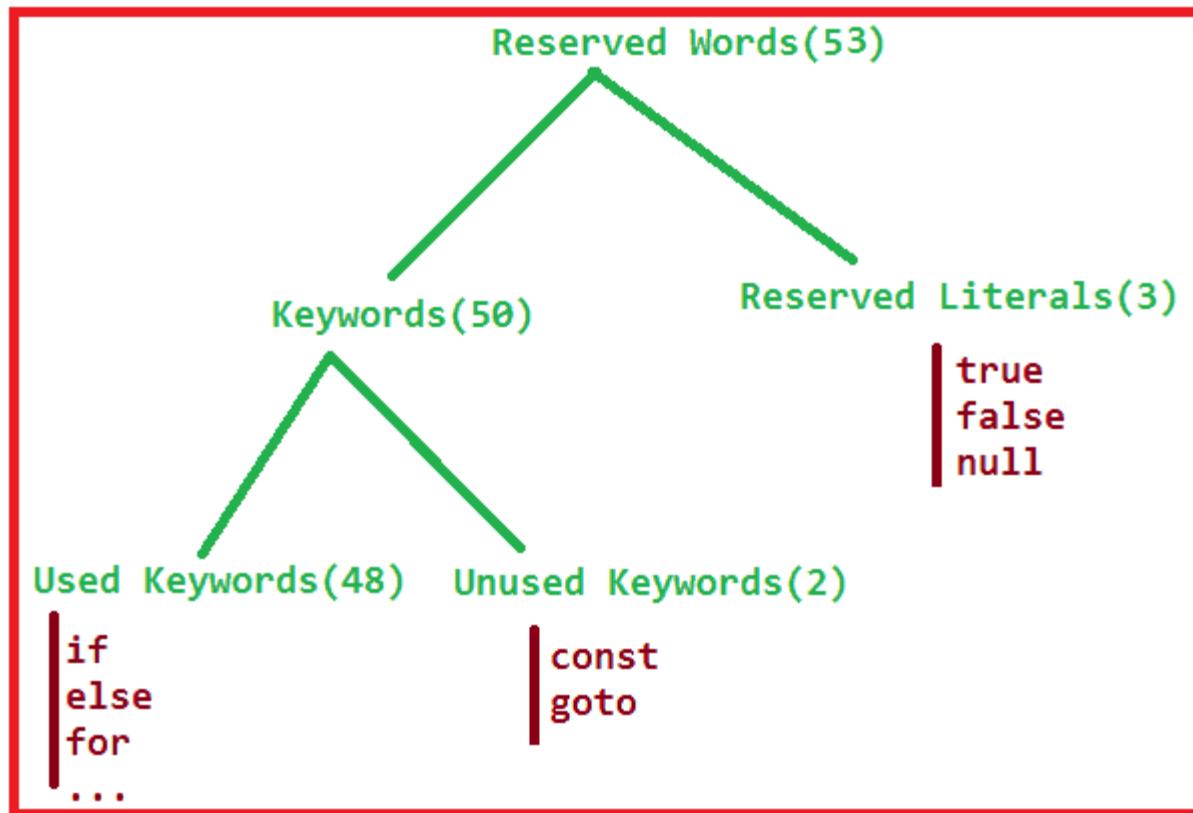
```
My Variable // Contiene un espacio  
123java // Comienza con un dígito  
a+c // El signo más (+) no es un carácter alfanumérico  
variable-2 // El guión (-) no es un carácter alfanumérico  
sum_&_difference // ampersand (&) no es un carácter válido
```



# PALABRAS RESERVADAS EN JAVA



Cualquier lenguaje de programación reserva algunas palabras para representar funcionalidades definidas por ese lenguaje. Estas palabras se llaman palabras reservadas. Estas pueden ser categorizadas brevemente en dos partes: palabras clave/keywords (50) y literales (3).



# PALABRAS RESERVADAS EN JAVA

*Las palabras clave definen funcionalidades y las literales definen un valor.*





# PALABRAS RESERVADAS EN JAVA

Tabla de Palabras Claves Reservadas en Java

abstract	assert	boolean	break	byte	case	catch	char	class	const	continue
default	do	double	else	enum	extends	false	final	finally	float	for
goto	if	implements	import	instanceof	int	interface	long	native	new	null
package	private	protected	public	return	short	static	strictfp	String	super	switch
synchronized	this	throw	throws	transient	true	try	void	volatile	while	





# CONVENCIONES DE NOMENCLATURA

## JAVA

*Las convenciones de nombres para el lenguaje de programación Java. Deben seguirse mientras se desarrolla software en Java para un buen mantenimiento y legibilidad del código. Java utiliza CamelCase como una práctica para escribir nombres de métodos, variables, clases, paquetes y constantes.*



## CLASES E INTERFACES EN JAVA

```
Interface Bicycle
Class MountainBike implements Bicycle

Interface Sport
Class Football implements Sport
```

# CONVENCIONES DE NOMENCLATURA **JAVA**



*Los nombres de las clases deben ser sustantivos, en mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúscula. El nombre de las interfaces también debe estar en mayúscula (la primera) al igual que los nombres de las clases. Use palabras completas y debe evitar acrónimos y abreviaturas.*





## METODOS EN JAVA

```
void changeGear(int newValue);  
void speedUp(int increment);  
void applyBrakes(int decrement);
```



# CONVENCIONES DE NOMENCLATURA **JAVA**



Los métodos deben ser verbos, en mayúsculas y minúsculas, con la primera letra de cada palabra interna (a partir de la segunda) en mayúscula.



# CONVENCIONES DE NOMENCLATURA JAVA

## VARIABLES EN JAVA

```
// variables para la clase MountainBike
int speed = 0;
int gear = 1;
```



*Los nombres de las variables deben ser cortos pero significativos.*

*No debería comenzar con un guión bajo ('\_') o caracteres, como por ejemplo, un signo de dólar '\$'.*

*Debe ser mnemotécnico, es decir, diseñado para indicar al observador casual la intención de su uso.*

*Se deben evitar los nombres de variable de un carácter, excepto para variables temporales.*

*Los nombres comunes para las variables temporales son: i, j, k, m y n para enteros; c, d y e para los caracteres.*





# CONVENCIONES DE NOMENCLATURA **JAVA**

## VARIABLES CONSTANTES EN JAVA

```
static final int MIN_WIDTH = 4;

// Algunas variables constantes utilizadas en la clase Float predefinida
public static final float POSITIVE_INFINITY = 1.0f / 0.0f;
public static final float NEGATIVE_INFINITY = -1.0f / 0.0f;
public static final float NaN = 0.0f / 0.0f;
```



*Debería estar todo en mayúsculas con palabras separadas por guiones bajos ("\_").*

*Hay varias constantes utilizadas en clases predefinidas como Float, Long, String, etc.*





# CONVENCIONES DE NOMENCLATURA **JAVA**

## PAQUETES EN JAVA

```
com.sun.eng
com.apple.quicktime.v2

// java.lang packet in JDK
java.lang
```



*El prefijo de un nombre de paquete único siempre se escribe en letras ASCII en minúsculas y debe ser uno de los nombres de dominio de nivel superior, como por ejemplo: com, edu, gov, mil, net, org.*

*Los componentes posteriores del nombre del paquete varían de acuerdo con las convenciones internas de nombres de la organización.*





# CLASES PREDEFINIDAS

NOMBRE DE CLASE

O

VARIABLE EN JAVA





# CLASES DE PREDIFINIDAS

*Los siguientes son algunos nombres de clase NO VÁLIDOS en Java*

```
boolean break = false; // no permitido ya que break es palabra clave
int boolean = 28; // no permitido ya que boolean es palabra clave
boolean goto = false; // no permitido ya que goto es palabra clave
String final = "hola"; // no permitido ya que final es palabra clave
```

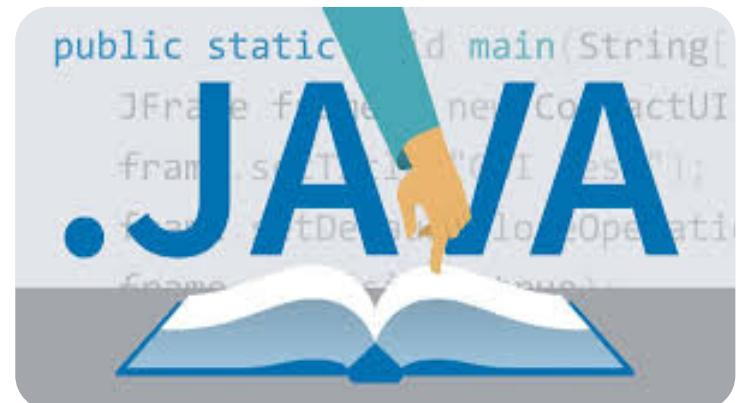




# DECLARACION Y TIPOS DE VARIABLES

*Una variable es el nombre dado a una ubicación de memoria. Es la unidad básica de almacenamiento en un programa. El valor almacenado en una variable se puede cambiar durante la ejecución del programa.*

*NOTA: “En Java, todas las variables deben declararse antes de que puedan ser utilizadas.”*





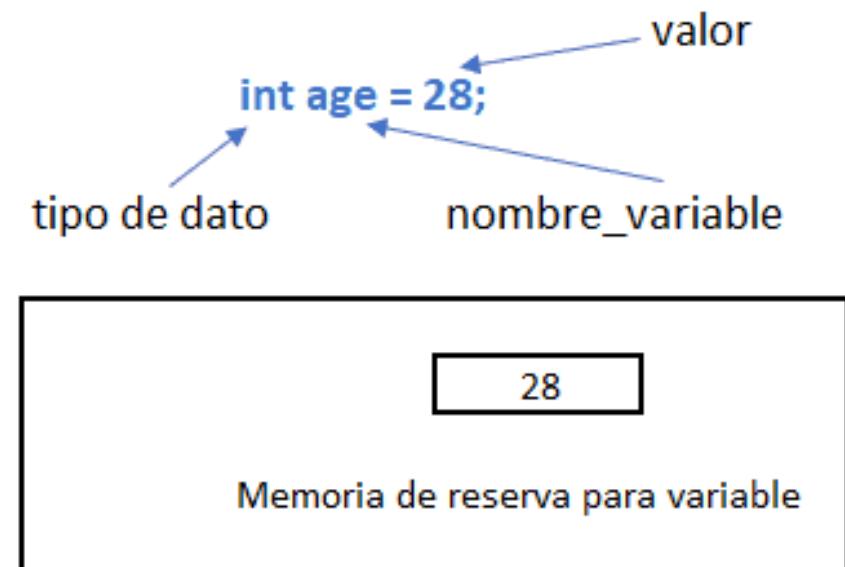
# ¿CÓMO SE DECLARA VARIABLES?

## DATOS PRIMITIVOS

**TIPO DE DATO:** tipo de datos que se pueden almacenar en esta variable.

**NOMBRE\_VARIABLE:** nombre dado a la variable.

**VALOR:** es el valor inicial almacenado en la variable.





# ¿CÓMO DECLARA VARIABLES?

```
float simpleInterest; //Declarando variable float
int time = 10, speed = 20; //Declarando e Inicializando la variable integer
char var = 'h'; // Declarando e Inicializando la variable character
```





# TIPOS DE VARIABLES

JAVA + TIPOS:

- **VARIABLES LOCALES**
- **VARIABLES DE INSTANCIA**
- **VARIABLES ESTÁTICAS**





*Una variable definida dentro de un bloque, método o constructor se llama variable local.*

*Estas variables se crean cuando el bloque ingresado o método se llama y destruye después de salir del bloque o cuando la llamada regresa del método.*

*El alcance de estas variables solo existe dentro del bloque en el que se declara la variable, es decir, podemos acceder a estas variables solo dentro de ese bloque.*

```
public class StudentDetails
{
    public void StudentAge()
    {
        //variable local age
        int age = 0;
        age = age + 5;
        System.out.println("La edad del estudiante es : " + age);
    }

    public static void main(String args[])
    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}
```





En el programa anterior, la variable age (edad) es una variable local para el método [java]StudentAge()[/java].

Si usamos la variable age fuera del método [java]StudentAge()[/java], el compilador producirá un error como se muestra a continuación en el programa

```
public class StudentDetails
{
    public void StudentAge()
    {   //variable local age
        int age = 0;
        age = age + 5;
        System.out.println("La edad del estudiante es : " + age);
    }

    public static void main(String args[])
    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}
```





*Las variables de instancia son variables no estáticas y se declaran en una clase fuera de cualquier método, constructor o bloque.*

*Como las variables de instancia se declaran en una clase, estas variables se crean cuando un objeto de la clase se crea y se destruye cuando se destruye el objeto.*

*A diferencia de las variables locales, podemos usar especificadores de acceso para variables de instancia. Si no especificamos ningún especificador de acceso, se utilizará el especificador de acceso predeterminado.*



```
import java.io.*;
class Points
{
    //Estas variables son variables de instancia.
    //Estas variables están en una clase y no están dentro de ninguna función/método
    int engPoints;
    int mathsPoints;
    int phyPoints;
}

class PointsDemo
{
    public static void main(String args[])
    {
        //primer objeto
        Points obj1 = new Points();
        obj1.engPoints = 50;
        obj1.mathsPoints = 80;
        obj1.phyPoints = 90;

        //segundo objeto
        Points obj2 = new Points();
        obj2.engPoints = 80;
        obj2.mathsPoints = 60;
        obj2.phyPoints = 85;

        //mostrando puntos para el primer objeto
        System.out.println("Puntos para el primer objeto:");
        System.out.println(obj1.engPoints);
        System.out.println(obj1.mathsPoints);
        System.out.println(obj1.phyPoints);

        //mostrando puntos para el segundo objeto
        System.out.println("Puntos para el segundo objeto:");
        System.out.println(obj2.engPoints);
        System.out.println(obj2.mathsPoints);
        System.out.println(obj2.phyPoints);
    }
}
```



```
import java.io.*;
class Points
{
    //Estas variables son variables de instancia.
    //Estas variables están en una clase y no están dentro de ninguna función/método
    int engPoints;
    int mathsPoints;
    int phyPoints;
}

class PointsDemo
{
    public static void main(String args[])
    {   //primer objeto
        Points obj1 = new Points();
        obj1.engPoints = 50;
        obj1.mathsPoints = 80;
        obj1.phyPoints = 90;

        //segundo objeto
        Points obj2 = new Points();
        obj2.engPoints = 80;
        obj2.mathsPoints = 60;
        obj2.phyPoints = 85;

        //mostrando puntos para el primer objeto
        System.out.println("Puntos para el primer objeto:");
        System.out.println(obj1.engPoints);
        System.out.println(obj1.mathsPoints);
        System.out.println(obj1.phyPoints);

        //mostrando puntos para el segundo objeto
        System.out.println("Puntos para el segundo objeto:");
        System.out.println(obj2.engPoints);
        System.out.println(obj2.mathsPoints);
        System.out.println(obj2.phyPoints);
    }
}
```





### VARIABLES ESTÁTICAS

*Las variables estáticas también se conocen como variables de clase.*

*Estas variables se declaran de forma similar a las variables de instancia, la diferencia es que las variables estáticas se declaran utilizando la palabra clave **[java]static[/java]** dentro de una clase fuera de cualquier constructor o bloque de métodos.*

*A diferencia de las variables de instancia, solo podemos tener una copia de una variable estática por clase, independientemente de cuántos objetos creemos.*

*Las variables estáticas se crean al inicio de la ejecución del programa y se destruyen automáticamente cuando finaliza la ejecución.*

```
import java.io.*;
class Emp {

    // salario como variable estatica
    public static double salary;
    public static String name = "Alex";
}

public class EmpDemo
{
    public static void main(String args[]) {

        //acceder a la variable estatica sin objeto
        Emp.salary = 1000;
        System.out.println(Emp.name + " tiene un salario promedio de: " + Emp.salary);
    }
}
```





### VARIABLES CONSTANTES

*Las variables creadas para almacenar valores fijos de esta manera se conocen como “constantes”, y es convencional nombrar constantes con todos los caracteres en mayúsculas, para distinguirlas de las variables regulares. Los programas que intentan cambiar un valor constante no se compilarán, y el compilador javac generará un mensaje de error.*

```
class Constants
{
    public static void main ( String[] args ) {
        //inicializamos tres constantes enteras
        final int CONSTANTE1 = 6 ;
        final int CONSTANTE2 = 1 ;
        final int CONSTANTE3 = 3 ;

        //declaramos variables regulares del tipo int
        int td,pat,fg,total;

        //Inicializamos las variables regulares
        td = 4*CONSTANTE1;
        pat = 3*CONSTANTE2;
        fg = 2*CONSTANTE3;

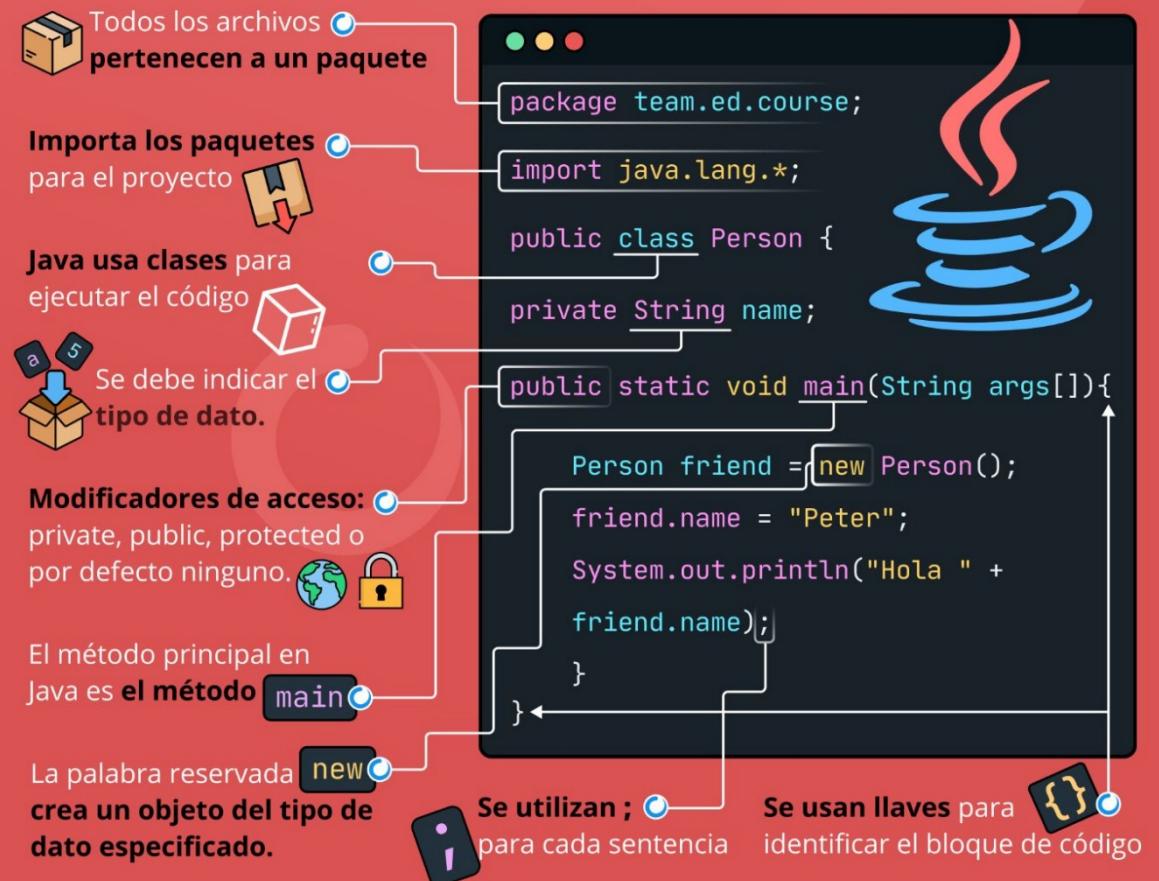
        total = (td+pat+fg) ;

        System.out.println("Resultado: " + total);
    }
}
```





# SINTAXIS BÁSICA DE JAVA



Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/java](http://ed.team/java)

 EDteam



# 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones creada para resolver una necesidad específica.**



## 1 GOOGLE GUAVA



### Mejora del flujo de trabajo

destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

## 2 RXJAVA



Crea aplicaciones para **reaccionar** a los flujos de datos en tiempo real.

## 3 MPANDROIDCHART



Crea **gráficos** de líneas, barras, radares y burbujas para Android.

## 4 FASTJSON



Convierte **objetos** Java en su representación JSON y viceversa.

## 5 MOCKITO



Librería de código abierto para **simular pruebas unitarias**.

## 6 JUNIT



Esta es la librería más **usada para pruebas**.

Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/cursos/java](http://ed.team/cursos/java)



# RETO PRACTICO

## Taller 1



# **RETO PRACTICO**

# **TALLER 2**



# CONCLUSIONES

.



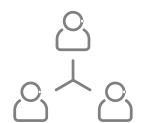


**¿ PREGUNTAS ?**



# RETROALIMENTACIÓN CLASE

**<https://forms.gle/7Ny6jwVKCYvXRqao9>**



# Gracias



Politécnico  
Internacional



“El problema real no es si las maquinas piensan, sino si lo hacen los hombres”



B. F – SKINNER

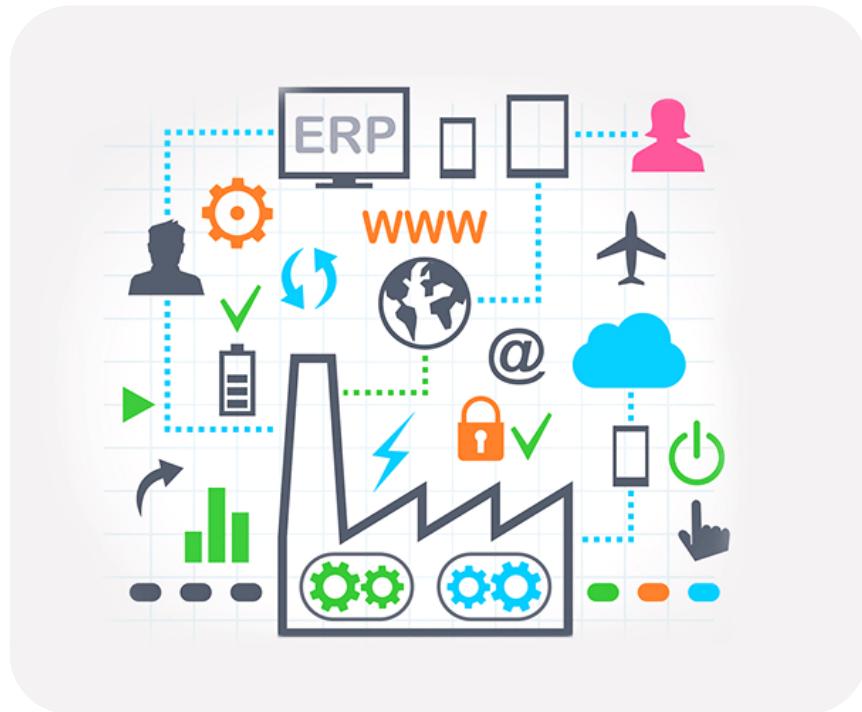
# ANTES DE EMPEZAR



# AGENDA

- *Retroalimentación - Clase Anterior*
- *Actividades Pendientes*
- *Palabras Reservadas*
- *Convenciones de Nomenclatura*
- *Clases Predefinidas*
- *Declaración y tipos de variables*
- *Preguntas*
- *Conclusiones*
- 





1 Sábados  
2 Break

8 AM a 12 M - 4 Horas  
9:30 AM a 09:45 AM

## ESPACIOS DE FORMACIÓN

# A TENER EN CUENTA



1. Cerrar los micrófonos y desactivar las cámaras.



2. Solicitar la palabra mediante el chat.



3. Preguntar

FUNDAMENTOS DE PROGRAMACIÓN



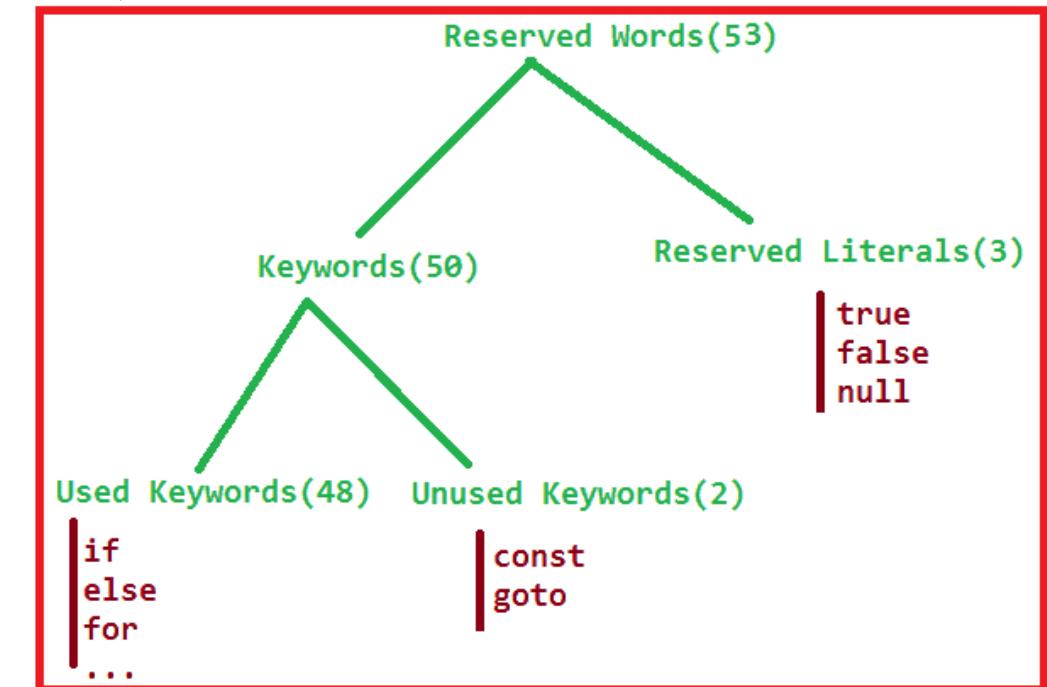
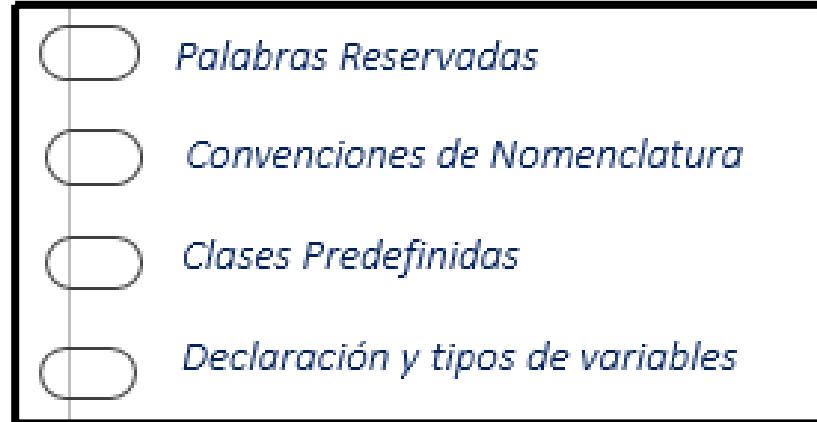
## CONTENIDOS

1. Algoritmos
2. Elementos lógicos de un algoritmo
3. Técnicas para la formulación de algoritmos. DFD, pseudocódigo, prueba de escritorio.
4. Introducción al Lenguaje de programación C#
5. Historia, evolución y futuro del lenguaje de programación C#
6. Introducción a la sintaxis de C#
7. Tipos de datos en C#
8. Operadores en C#
9. Estructuras de control en C#
10. Arreglos en C#
11. Manejo de excepciones en C#
12. Colecciones en C# (Espacio de nombres System.Collections)
13. LinQ con C#

# CONTENIDOS

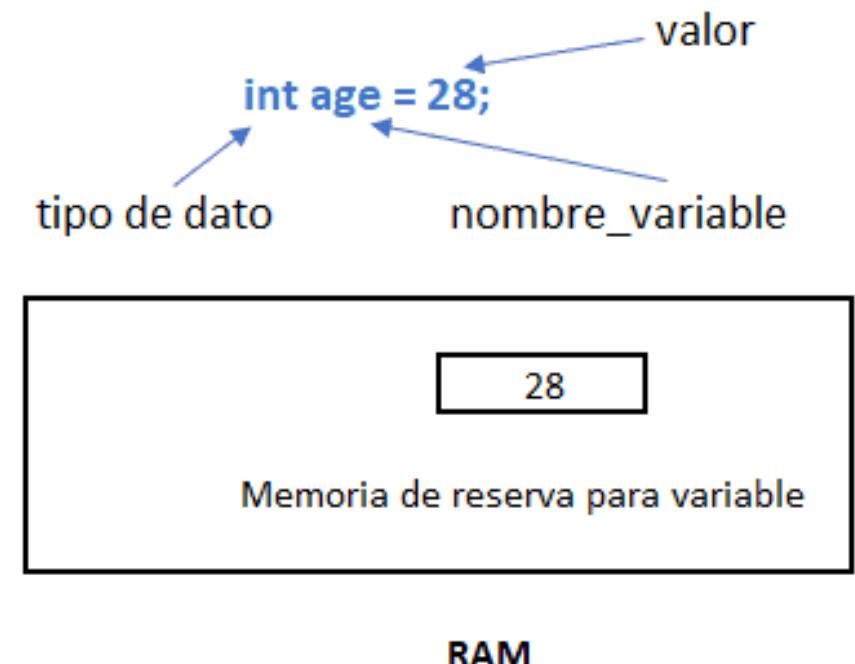
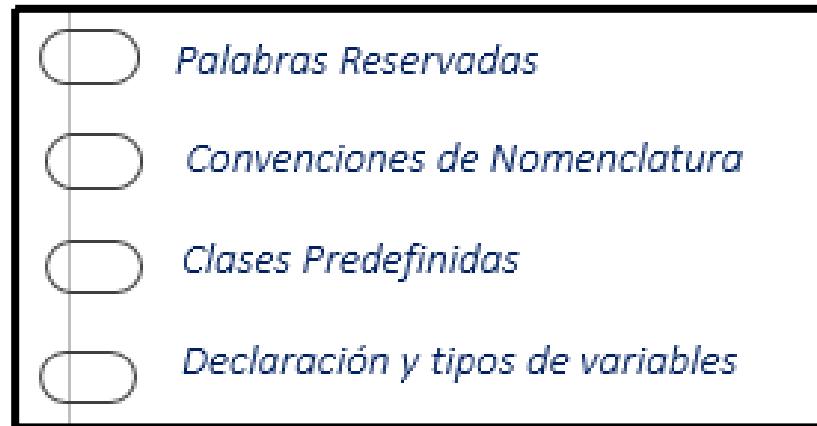


# RETROALIMENTACIÓN: CLASE ANTERIOR





# RETROALIMENTACIÓN: CLASE ANTERIOR





# RETROALIMENTACIÓN: CLASE ANTERIOR

- Palabras Reservadas*
- Convenciones de Nomenclatura*
- Clases Predefinidas*
- Declaración y tipos de variables*

## **JAVA + TIPOS:**

- **VARIABLES LOCALES**
- **VARIABLES DE INSTANCIA**
- **VARIABLES ESTÁTICAS**



# RETROALIMENTACIÓN: PRUEBA DE ESCRITORIO

## FUNDAMENTOS DE PROGRAMACION

Página Principal / Mis cursos / FUNDAMENTOS DE PROGRAMACION / Aprendizaje Individual

[General](#) [Información del curso](#) [Material de Estudio y Apoyo](#)

[Aprendizaje Individual](#)

[Aprendizaje Colaborativo](#)

Seguimiento y Evaluación

[Unidad 1](#) [Unidad 2](#) [Unidad 3](#)



[PRUEBA DE ESCRITORIO](#)

*FUNDAMENTOS DE PROGRAMACIÓN*



1	2
3	4
5	6
7	8
A	1
0101	1010
0	
2B	AF
0	0

DIAGRAMA DE FLUJO - Ejercicio 1 - ANÁLISIS, CONSTRUCCIÓN Y PRUEBA DE ESCRITORIO

## Diagramas de flujo

• Resta de 2 números:

a) Análisis

- 2 números reales
- 2 var. de entrada:  $a, b$
- 1 var. alfanumérica:  $c$
- $C = a - b$

b) Construcción

c) Prueba de escritorio

1. Inicio
2. 5, 2
3.  $a = 5, b = 2$
4.  $c = 5 - 2$   
 $c \leftarrow 3$

```
graph TD; Start([Inicio]) --> Input[/Da 2 números/]; Input --> Decision{a, b}; Decision --> Process[C = a - b<br/>C ← a - b]; Process --> Decision; Output["El resultado es: ", c] --- Fin([Fin]);
```

INCINERIA



# FORO DE CONOCIMIENTOS

## FOROS



### Habilidades de un programador



Al hablar de programación, encontramos temas técnicos en cuanto a conceptualización se trata. De forma adicional se encuentran algunas habilidades que son reconocidas en los profesionales dedicados al desarrollo de software para esto por favor lea el artículo <https://cutt.ly/xfHnbWo>

Luego de leer el artículo mencione como se pueden desarrollar estas habilidades y en que escenarios se puede potenciar el fortalecimiento de las mismas.

Saludos.





# ENTREGA TALLERES JAVA I - II

**Evaluable**



# RETO



Sobre nosotros    Padres    Impacto    Mis clases    Mi Cuenta ▾    español

## CIDE\_FUNDAMENTOS [editar la configuración de la clase](#)

FUNDAMENTOS\_PROGRAMACION

Resumen de la clase

Idioma : JavaScript

Estudiantes : 0

Tiempo de juego de nivel medio :

Tiempo total de juego :

Niveles promedio completados :

Niveles totales completados :

Creado : 14/8/2020

Agregar estudiantes :

**NightShirtMilk**

Copiar código de clase

Los estudiantes pueden unirse a su clase usando este Código de clase. No se requiere una dirección de correo electrónico al crear una cuenta de estudiante con este código de clase.

<https://codecombat.com/stude>

Copiar URL de clase

También puede publicar esta URL de clase única en una página web compartida.



# RETO

## FLEXBOX FROGGY

Nivel 1 de 24

Bienvenido a Flexbox Froggy, un juego donde ayudarás a Froggy ya sus amigos escribiendo código CSS. Guía a esta rana hacia la hoja de lirio en la derecha, usando la propiedad **justify-content**, la cual alinea elementos horizontalmente y acepta los siguientes valores:

- **flex-start**: Alinea elementos al lado izquierdo del contenedor.
- **flex-end**: Alinea elementos al lado derecho del contenedor.
- **center**: Alinea elementos en el centro del contenedor.
- **space-between**: Muestra elementos con la misma distancia entre ellos.
- **space-around**: Muestra elementos con la misma separación alrededor de ellos.

Por ejemplo, **justify-content: flex-end;** moverá la rana a la derecha.

```
1 #pond {  
2   pantalla: flex;  
3   justify-content: flex-end;  
4 }  
5  
6
```





## LECTURA DE DATOS



*Al escribir un programa debemos realizar lectura de los datos ingresados por teclado.*



```
6 package cide_entrada_datos_buffered_reader;
7 import java.io.BufferedReader;
8 import java.io.IOException;
9 import java.io.InputStreamReader;
10 /**
11 *
12 * @author Harol
13 */
14 // CIDE TIPOS DE DATOS
15 // FUNDAMENTOS DE PROGRAMACION
16 public class CIDE_ENTRADA_DATOS_BUFFERED_READER {
17
18     /**
19      * @param args the command line arguments
20     */
21     public static void main(String[] args) throws IOException {
22         // TODO code application logic here
23         //Ingrese datos usando BufferedReader
24         BufferedReader reader =
25             new BufferedReader(new InputStreamReader(System.in));
26         // Leyendo datos usando readLine
27         String name = reader.readLine();
28
29         // Imprimir la linea de lectura
30         System.out.println(name);
31     }
32
33 }
```

# CLASE BUFFERED READER

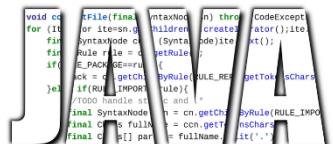


Este es el método clásico de Java para leer datos de entrada, introducido en JDK 1.0. Este método se usa envolviendo `System.in` (flujo de entrada estándar) en un `InputStreamReader` que está envuelto en un `BufferedReader`, podemos leer la entrada del usuario en la línea de comando



```
package cide_entrada_datos_clase_escaner;
import java.util.Scanner;
/**
 *
 * @author Harol
 */
// CIDE ENTRADA DE DATOS
// FUNDAMENTOS DE PROGRAMACION
public class CIDE_ENTRADA_DATOS_CLASE_ESCANER {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        // Usando Scanner para obtener información del usuario
        Scanner in = new Scanner(System.in);
        String s = in.nextLine();
        System.out.println("Usted ingresó la cadena: "+s);
        int a = in.nextInt();
        System.out.println("Usted ingresó un entero: "+a);
        float b = in.nextFloat();
        System.out.println("Usted ingresó un float: "+b);
    }
}
```



# CLASE ESCANER



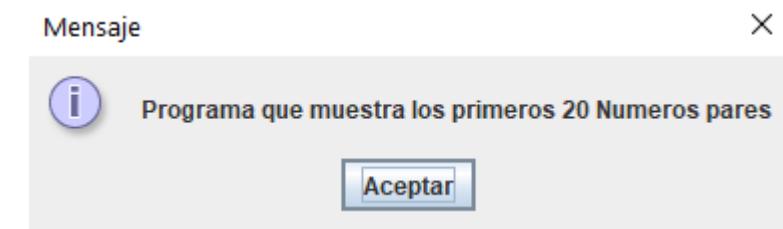
Este es probablemente el método más preferido para tomar datos de entrada. El objetivo principal de la clase Scanner es analizar los tipos primitivos y las cadenas con expresiones regulares, sin embargo, también se puede usar para leer las entradas del usuario en la línea de comandos.



```
6 package cide_entrada_datos_joption_pane;
7 import javax.swing.*;
8 /**
9 * @author Harol
10 */
11
12 public class CIDE_ENTRADA_DATOS_JOPTION_PANE {
13
14     /**
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         // TODO code application logic here
19         JOptionPane.showMessageDialog(null, "Programa que muestra los primeros 20 Numeros pares");
20         int i=0,ci;
21         for (int c=0;c<=38;c++) {
22             i = i + 2;
23             c = i;
24             JOptionPane.showMessageDialog(null, c);
25         }
26         JOptionPane.showMessageDialog(null,"CIDE Fundamentos Programacion Harol Torres");
27     }
...}
```

## JOPTION\_PANE

Opción diferente a utilizar la consola.  
Muestra un cuadro de dialogo para la interacción entre el usuario y el programa desarrollado.





# RETO + PRESENTACIÓN DE DATOS DE ENTRADA

ENTREGA	
1	pseudocodigo
2	diagrama de flujo
3	prueba de escritorio
4	codigo Java
5	Evidencia

CLASE BUFFERED READER

JOPTION\_PANE

CLASE ESCANER



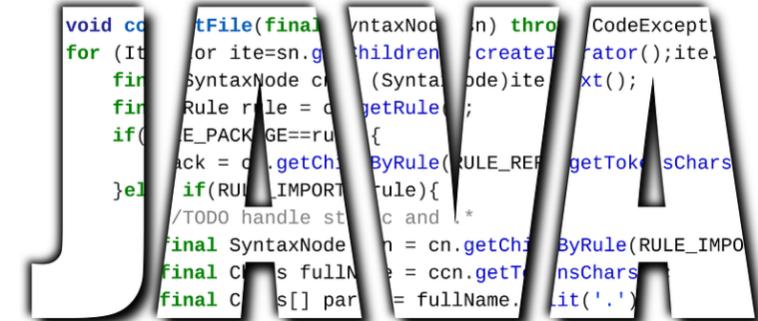


*La toma de decisiones en la programación es similar a la toma de decisiones en la vida real.*

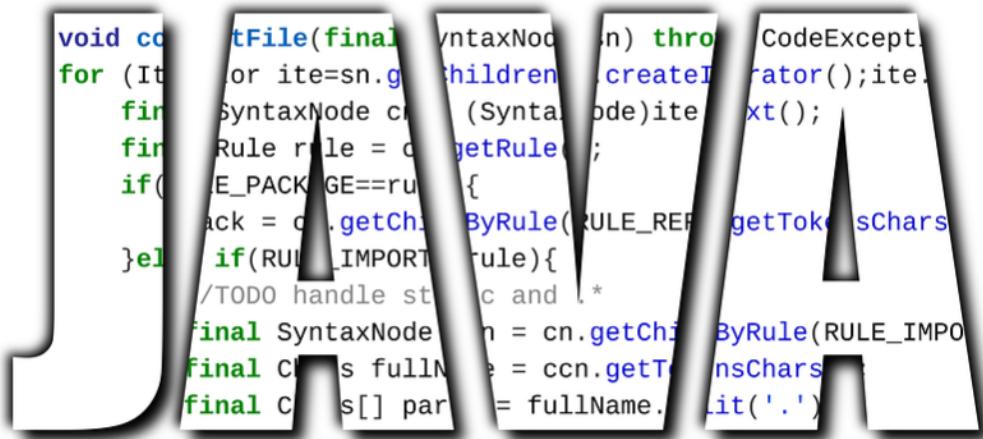
*En la programación también enfrentamos algunas situaciones en las que queremos que se ejecute cierto bloque de código cuando se cumple alguna condición.*

*Un lenguaje de programación utiliza instrucciones de control para controlar el flujo de ejecución del programa en función de ciertas condiciones. Estos se utilizan para hacer que el flujo de ejecución avance y se ramifique en función de los cambios en el estado de un programa.*

# ESTRUCTURAS CONDICIONALES JAVA



# ESTRUCTURAS CONDICIONALES JAVA



## *Declaraciones de selección de Java:*

- *if*
  - *if-else*
  - *nested-if*
  - *if-else-if*
  - *switch-case*
  - *jump – break, continue, return*





# IF CONDICIONALES JAVA

```
if (condición)
{
    // Declaraciones para ejecutar si
    // la condición es verdadera
}
```



*La sentencia if es la declaración de toma de decisiones más simple.*

*Se usa para decidir si una determinada declaración o bloque de enunciados se ejecutará o no; es decir, si una determinada condición es verdadera (true), se ejecutará un bloque de enunciado y, de ser falsa (false), no.*





```
if (condición)
{
    // Declaraciones para ejecutar si
    // la condición es verdadera
}
```

# IF CONDICIONALES JAVA



*La sentencia if es la declaración de toma de decisiones más simple.*

*Se usa para decidir si una determinada declaración o bloque de enunciados se ejecutará o no; es decir, si una determinada condición es verdadera (true), se ejecutará un bloque de enunciado y, de ser falsa (false), no.*





```
if (condición)
{
    // Ejecuta este bloque si
    // la condición es verdadera
}
else
{
    // Ejecuta este bloque si
    // la condición es falsa
}
```

# IF - ELSE CONDICIONALES JAVA



*La declaración if solo nos dice que, si una condición es verdadera ejecutará un bloque de instrucciones y si la condición es falsa, no lo hará. Pero, ¿y si queremos hacer otra cosa cuando la condición sea falsa? Aquí viene la declaración else. Podemos usar la instrucción else con la instrucción if para ejecutar un bloque de código cuando la condición es falsa.*





```
if (condicion1)
{
    // Ejecuta cuando condicion1 es verdadero
    if (condition2)
    {
        // Ejecuta cuando condicion2 es verdadero
    }
}
```

# IF - ANIDADO CONDICIONALES JAVA



*Una if anidado (nested-if) es una declaración if que se deriva de otro if o else. Las declaraciones if anidadas significan una instrucción if dentro de una declaración if. Sí, Java nos permite anidar las declaraciones if con if, es decir, podemos colocar una instrucción if dentro de otra instrucción if.*





```
if (condicion1)
{
    // Ejecuta cuando condicion1 es verdadero
    if (condition2)
    {
        // Ejecuta cuando condicion2 es verdadero
    }
}
```

# IF - ANIDADO CONDICIONALES JAVA



*Una if anidado (nested-if) es una declaración if que se deriva de otro if o else. Las declaraciones if anidadas significan una instrucción if dentro de una declaración if. Sí, Java nos permite anidar las declaraciones if con if, es decir, podemos colocar una instrucción if dentro de otra instrucción if.*





```
if (condición)
    declaración;
else if (condición)
    declaración;
.
.
declaración
else ;
```

# ESCALERA IF - ELSE CONDICIONALES JAVA



Un usuario puede decidir entre múltiples opciones. Las sentencias `if` se ejecutan desde arriba hacia abajo. Tan pronto como una de las condiciones que controlan el `if` sea verdadera, se ejecuta la instrucción asociada con ese `if`, y el resto de la escalera se pasa por alto. Si ninguna de las condiciones es verdadera, se ejecutará la sentencia final `else`.





```
switch (expresión)
{
    case valor1:
        declaracion1;
        break;
    case value2:
        declaracion2;
        break;
    .
    .
    case valorN:
        declaracionN;
        break;
    default:
        declaracionDefault;
}
```

# SWITCH CASE CONDICIONALES JAVA



La instrucción `switch` es una declaración de bifurcación de múltiples vías (selección múltiple). Proporciona una forma sencilla de enviar la ejecución a diferentes partes del código en función del valor de la expresión.





# JUMP CONDICIONALES JAVA



Java admite tres declaraciones de salto: *break*, *continue* y *return*. Estas tres declaraciones transfieren el control a otra parte del programa.



```
// Programa Java para ilustrar usando
// break para salir del bucle
class BreakLoopDemo
{
    public static void main(String args[])
    {
        // Inicialmente, el bucle está configurado para ejecutarse desde 0-9
        for (int i = 0; i < 10; i++)
        {
            // termina el bucle cuando llega a 5.
            if (i == 5)
                break;

            System.out.println("i: " + i);
        }
        System.out.println("Bucle completo.");
    }
}
```

# BREAK CONDICIONALES JAVA



*En Java, break se utiliza principalmente para:*

*Terminar una secuencia en una instrucción switch (discutida arriba).*

*Para salir de un bucle*

*Como una forma “civilizada” de goto.*

*Usar break para salir de un bucle*

*Utilizando el break, podemos forzar la terminación inmediata de un bucle, evitando la expresión condicional y cualquier código restante en el cuerpo del bucle.*





```
// Programa Java para ilustrar usando
// continue en una declaración if
class ContinueDemo
{
    public static void main(String args[])
    {
        for (int i = 0; i < 10; i++)
        {
            // Si el número es par
            // omitir y continuar
            if (i%2 == 0)
                continue;

            // Si el número es impar, imprimalo
            System.out.print(i + " ");
        }
    }
}
```

# CONTINUE CONDICIONALES JAVA



es útil forzar una iteración temprana de un bucle. Es decir, es posible que desee continuar ejecutando el bucle, pero deje de procesar el resto del código (en su cuerpo) para esta iteración en particular. Esto es, en efecto, un *goto* pasando del cuerpo del bucle, al final del bucle. La instrucción *continue* realiza tal acción.





```
// Programa Java para ilustrar usando return
class Return
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Antes de return.");

        if (t)
            return;

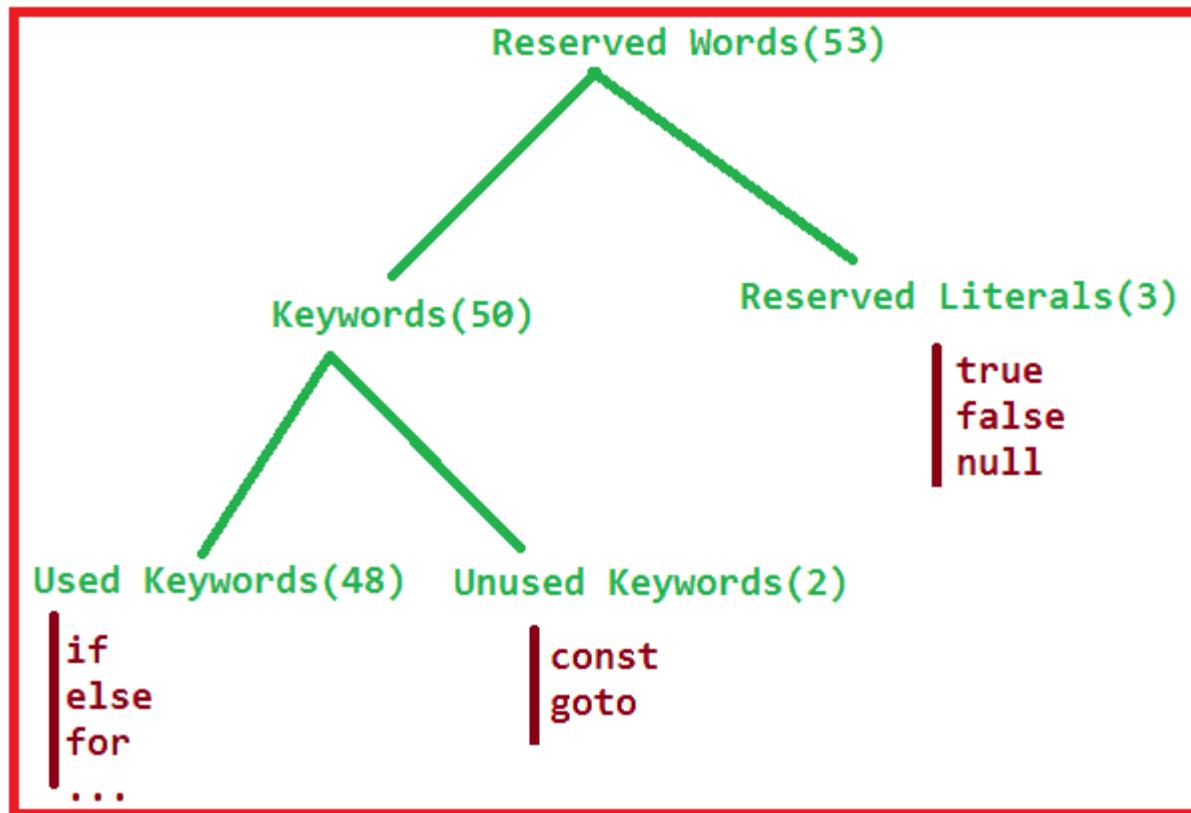
        // El compilador eludirá todas las declaraciones
        // después de return
        System.out.println("Esto no se ejecutará.");
    }
}
```

# RETURN CONDICIONALES JAVA



*La declaración RETURN se usa para regresar explícitamente de un método. Es decir, hace que un control de programa se transfiera nuevamente a quién llama del método.*





# PALABRAS RESERVADAS EN JAVA

*Las palabras clave definen funcionalidades y las literales definen un valor.*





# PALABRAS RESERVADAS EN JAVA

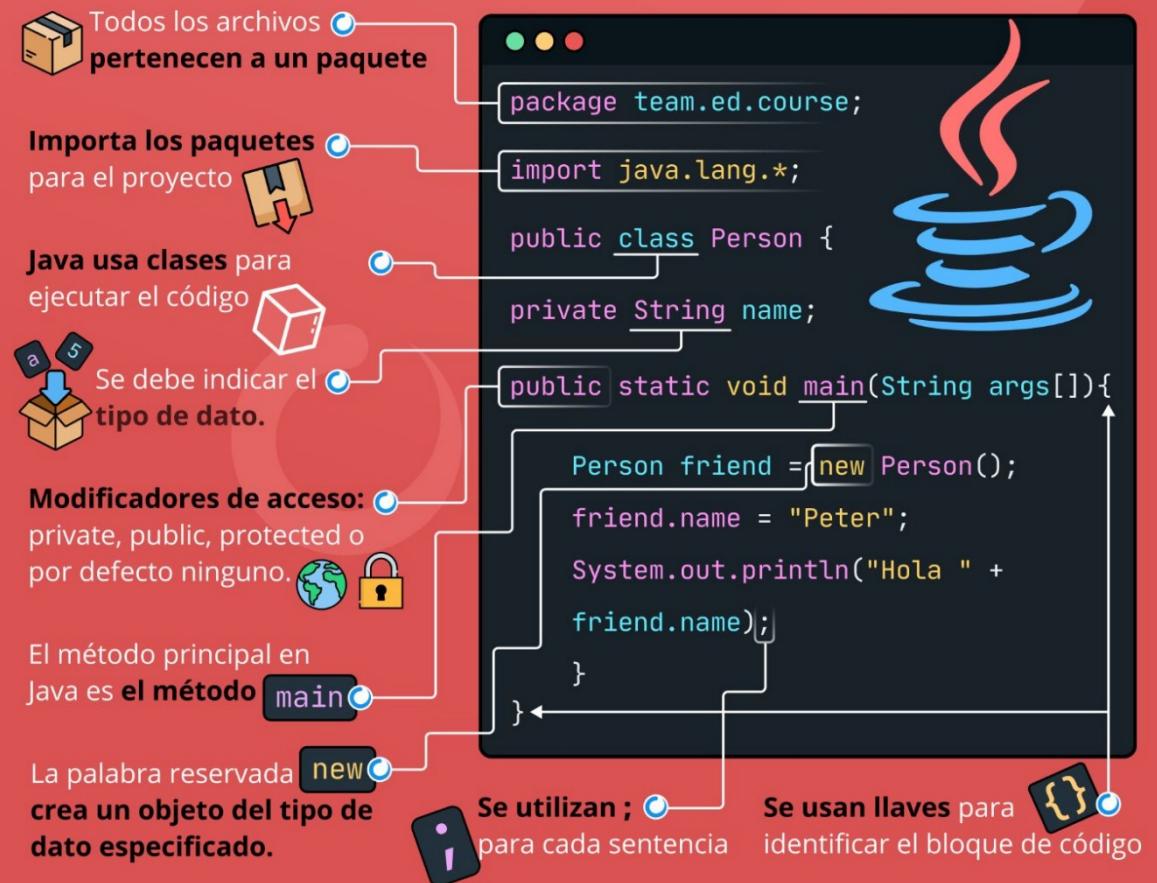
Tabla de Palabras Claves Reservadas en Java

abstract	assert	boolean	break	byte	case	catch	char	class	const	continue
default	do	double	else	enum	extends	false	final	finally	float	for
goto	if	implements	import	instanceof	int	interface	long	native	new	null
package	private	protected	public	return	short	static	strictfp	String	super	switch
synchronized	this	throw	throws	transient	true	try	void	volatile	while	





# SINTAXIS BÁSICA DE JAVA



Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/java](http://ed.team/java)

 EDteam



# 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones creada para resolver una necesidad específica.**



## 1 GOOGLE GUAVA



### Mejora del flujo de trabajo

destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

## 2 RXJAVA



Crea aplicaciones para **reaccionar** a los flujos de datos en tiempo real.

## 3 MPANDROIDCHART



Crea **gráficos** de líneas, barras, radares y burbujas para Android.

## 4 FASTJSON



Convierte **objetos** Java en su representación JSON y viceversa.

## 5 MOCKITO



Librería de código abierto para **simular pruebas unitarias**.

## 6 JUNIT



Esta es la librería más **usada para pruebas**.

Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/cursos/java](http://ed.team/cursos/java)



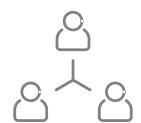
# CONCLUSIONES

.





**¿ PREGUNTAS ?**



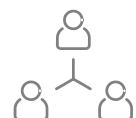
# Gracias



Politécnico  
Internacional



**“Cualquier tecnología  
suficientemente  
avanzada es  
equivalente a la  
magia.”**



Arthur C. Clarke – Escritor y Científico

# ANTES DE EMPEZAR



# AGENDA

- Retroalimentación - Clase Anterior*
- Actividades Pendientes*
- Comentarios Java*
- Ejemplos*
- Ciclos*
- Ejemplos*
- Preguntas*
- Conclusiones*
- Conclusiones*





1 Sábados  
2 Break

8 AM a 12 M - 4 Horas  
9:30 AM a 09:45 AM

## ESPACIOS DE FORMACIÓN

# A TENER EN CUENTA



1. Cerrar los micrófonos y desactivar las cámaras.



2. Solicitar la palabra mediante el chat.



3. Preguntar

FUNDAMENTOS DE PROGRAMACIÓN



## CONTENIDOS

1. Algoritmos
2. Elementos lógicos de un algoritmo
3. Técnicas para la formulación de algoritmos. DFD, pseudocódigo, prueba de escritorio.
4. Introducción al Lenguaje de programación C#
5. Historia, evolución y futuro del lenguaje de programación C#
6. Introducción a la sintaxis de C#
7. Tipos de datos en C#
8. Operadores en C#
9. Estructuras de control en C#
10. Arreglos en C#
11. Manejo de excepciones en C#
12. Colecciones en C# (Espacio de nombres System.Collections)
13. LinQ con C#

# CONTENIDOS



## LECTURA DE DATOS

- CLASE BUFFERED READER
- CLASE ESCANER
- JOPTION\_PANE

## ESTRUCTURAS CONDICIONALES

- IF
- IF - ELSE
- IF ANIDADO
- ECALERA IF
- SWITCH CASE
- JUMP
- BREAK
- CONTINUE
- RETURN





# RETROALIMENTACIÓN: CLASE ANTERIOR

ESTUDIANTES PENDIENTES POR PRESENTACION  
EJERCICIO





# INFORME FINAL

## CRITERIOS DE ACEPTACIÓN

1. PORTADA
  2. TABLA DE CONTENIDO
  3. PRESENTACION DEL PROBLEMA
  4. PSEUDOCODIGO
  5. DIAGRAMA DE FLUJO
  6. PRUEBA DE ESCRITORIO
  7. CODIGO JAVA (COMENTARIOS)
  8. COMPIILACION Y GENERACION DE .JAR
  9. VIDEO PUBLICADO EN YOUTUBE CON LA EXPLICACION – MINIMO 7 MINUTOS.
  10. PUBLICACION GITHUB / PROYECTO
  11. CONCLUSIONES
  12. BIBLIOGRAFIA (NORMA APA)
- Arial 12 + Formato PDF





# ENTREGA TALLERES JAVA I - II

**Evaluable**

1. PSEUDOCODIGO
2. DIAGRAMA DE FLUJO
3. PRUEBA DE ESCRITORIO
4. CODIGO JAVA (COMENTARIOS)



# RETO



Sobre nosotros    Padres    Impacto    Mis clases    Mi Cuenta ▾    español

## CIDE\_FUNDAMENTOS [editar la configuración de la clase](#)

FUNDAMENTOS\_PROGRAMACION

Resumen de la clase

Idioma : JavaScript

Estudiantes : 0

Tiempo de juego de nivel medio :

Tiempo total de juego :

Niveles promedio completados :

Niveles totales completados :

Creado : 14/8/2020

Agregar estudiantes :

**NightShirtMilk**

Copiar código de clase

Los estudiantes pueden unirse a su clase usando este Código de clase. No se requiere una dirección de correo electrónico al crear una cuenta de estudiante con este código de clase.

<https://codecombat.com/stude>

Copiar URL de clase

También puede publicar esta URL de clase única en una página web compartida.



# RETO

## FLEXBOX FROGGY

Nivel 1 de 24

Bienvenido a Flexbox Froggy, un juego donde ayudarás a Froggy ya sus amigos escribiendo código CSS. Guía a esta rana hacia la hoja de lirio en la derecha, usando la propiedad **justify-content**, la cual alinea elementos horizontalmente y acepta los siguientes valores:

- **flex-start**: Alinea elementos al lado izquierdo del contenedor.
- **flex-end**: Alinea elementos al lado derecho del contenedor.
- **center**: Alinea elementos en el centro del contenedor.
- **space-between**: Muestra elementos con la misma distancia entre ellos.
- **space-around**: Muestra elementos con la misma separación alrededor de ellos.

Por ejemplo, **justify-content: flex-end;** moverá la rana a la derecha.

```
1 #pond {  
2   pantalla: flex;  
3   justify-content: flex-end;  
4 }  
5  
6
```





## COMENTARIOS JAVA



*En un programa, los comentarios forman parte en hacer que el programa se vuelva más legible mediante el uso de los detalles del código haciendo que el mantenimiento sea más fácil y pueda encontrarse errores fácilmente. Los comentarios son ignorados por el compilador al compilar un código*



## TIPOS DE COMENTARIOS



En Java hay tres tipos de comentarios:

- Comentarios de una sola línea.
- Comentarios de múltiples líneas.
- Comentarios de la documentación.

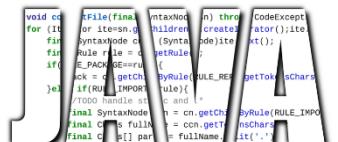


Sintaxis:

```
//Comentarios aqui ( El texto en esta linea solo se considera comentario )
```

Ejemplo:

```
//Programa Java que muestra comentarios de una sola linea
class Scomment
{
    public static void main(String args[])
    {
        // Comentario de una sola linea aquí
        System.out.println("Comentario de una sola linea arriba");
    }
}
```



# COMENTARIOS 1 LINEA



Un programador de nivel principiante utiliza comúnmente comentarios de una sola línea para describir la funcionalidad del código. Son los comentarios más fáciles de escribir.



```
/*El comentario comienza
continúa
continúa
.
.
.
El comentario termina*/
```

Ejemplo:

```
//Programa Java para mostrar comentarios de varias líneas
class Scomment
{
    public static void main(String args[])
    {
        System.out.println("Comentarios de varias líneas a continuación");
        /*Línea de comentario 1
         Línea de comentario 2
         Línea de comentario 3*/
    }
}
```

# COMENTARIOS VARIAS LINEAS



Para describir un método completo en un código o un fragmento complejo, los comentarios de una sola línea pueden ser tediosos de escribir, ya que tenemos que dar '//' en cada línea. Entonces, para evitar repetir '//' en cada línea, podemos utilizar la sintaxis de los comentarios de varias líneas.





The screenshot shows a web browser displaying the Java API documentation for the `Scanner` class. The URL is [docs.oracle.com/javase/7/docs/api/java/util/Scanner.html](https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html). The page title is "Comentarios en Java: Tipos y Ejer". The navigation bar includes links for "Aplicaciones", "Trabajo", "Personal", "Certificados", "Educacion", "Trabajo\_Search", "UD", and "Linux". The main content area shows the `java.util` package and the **Escáner de clase** section. It details the `Scanner` class, which extends `Object` and implements `Iterator<String>` and `Closable`. The class is described as a simple text scanner that can analyze primitive types and strings using regular expressions. Examples are provided for reading from `System.in` and from a file named `myNumbers`.

```
Scanner sc = nuevo escáner (System.in);
int i = sc.nextInt ();
```

```
Scanner sc = new Scanner (nuevo archivo ("myNumbers"));
while (sc.hasNextLong ()) {
    long aLong = sc.nextLong ();
}
```

<https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html>

# COMENTARIOS EN LA DOCUMENTACIÓN



*Este tipo de comentarios se utilizan generalmente al escribir código para un paquete de proyecto/software, ya que ayuda a generar una página de documentación para referencia, que puede usarse para obtener información sobre los métodos presentes, sus parámetros, etc.*





# COMENTARIOS EN LA DOCUMENTACIÓN

ETIQUETA	DESCRIPCIÓN	SINTAXIS
@author	Agrega el autor de una clase.	@author nombre
{@code}	Muestra texto en la fuente del código sin interpretar el texto como marcado HTML o etiquetas javadoc anidadas.	{@code texto}
{@docRoot}	Representa la ruta relativa al directorio raíz del documento generado desde cualquier página generada.	{@docRoot}
@deprecated	Agrega un comentario que indica que esta API ya no se debe usar.	@deprecated texto-obsoleto
@exception	Agrega un subtítulo “Throws” a la documentación generada, con el nombre de clase y el texto de descripción.	@exception descripción del nombre-clase
{@inheritDoc}	Hereda un comentario de la clase heredable más cercana o la interfaz implementable.	Hereda un comentario de la superclase inmediata.
{@link}	Inserta un enlace con la etiqueta de texto visible que apunta a la documentación del paquete, clase o nombre de miembro especificado de una clase referenciada.	{@linkplain package.class#etiqueta-miembro}



# COMENTARIOS EN LA DOCUMENTACIÓN

{@linkplain}	Idéntico a {@link}, excepto que la etiqueta del enlace se muestra en texto plano que en la fuente del código.	{@linkplain package.class#etiqueta-miembro}
@param	Agrega un parámetro con el nombre de parámetro especificado seguido de la descripción especificada en la sección "Parámetros".	@param descripción del nombre-parámetro
@return	Agrega una sección "Returns" con el texto de descripción.	@return descripción
@see	Agrega un encabezado "See Also" ("Ver también") con un enlace o entrada de texto que apunta a la referencia.	@see referencia
@serial	Usado en el comentario del documento para un campo serializable predeterminado.	@serial campo-Descripción   incluir   excluir
@serialData	Documenta los datos escritos por los métodos writeObject() o writeExternal().	@serialData descripción-datos
@serialField	Documenta un componente ObjectStreamField.	@serialField campo-nombre campo-tipo campo-descripción



# COMENTARIOS EN LA DOCUMENTACIÓN

{@linkplain}	Idéntico a {@link}, excepto que la etiqueta del enlace se muestra en texto plano que en la fuente del código.	{@linkplain package.class#etiqueta-miembro}
@param	Agrega un parámetro con el nombre de parámetro especificado seguido de la descripción especificada en la sección "Parámetros".	@param descripción del nombre-parámetro
@return	Agrega una sección "Returns" con el texto de descripción.	@return descripción
@see	Agrega un encabezado "See Also" ("Ver también") con un enlace o entrada de texto que apunta a la referencia.	@see referencia
@serial	Usado en el comentario del documento para un campo serializable predeterminado.	@serial campo-Descripción   incluir   excluir
@serialData	Documenta los datos escritos por los métodos writeObject() o writeExternal().	@serialData descripción-datos
@serialField	Documenta un componente ObjectStreamField.	@serialField campo-nombre campo-tipo campo-descripción





# COMENTARIOS EN LA DOCUMENTACIÓN

@since	Indica a partir de que versión de la API fue incluida la clase o método.	@since release
@throws	Las etiquetas @throws y @exception son sinónimos.	@throws descripción del nombre de la clase
{@value}	Cuando {@value} se usa en el comentario del doc de un campo estático, muestra el valor de esa constante.	{@value package.class#campo}
@version	Agrega un subtítulo de "Versión" con el texto de versión especificado a los documentos generados cuando se usa la opción -version.	@version texto-version





# COMENTARIOS EN LA DOCUMENTACIÓN

The screenshot shows the NetBeans IDE interface with the generated documentation for the `FindAvg` project. The left pane displays the class hierarchy under "All Classes". The main pane shows the JavaDoc for the `findAvg` method and the `main` method. The `findAvg` method documentation includes parameters `numA`, `numB`, and `numC`, and a note about calculating the average of three integers. The `main` method documentation includes parameters `args` and a note about calling the `findAvg` method.

```
public int findAvg(int numA,  
                   int numB,  
                   int numC)  
  
Este metodo se usa para encontrar un promedio de tres enteros.  
  
Parameters:  
numA - Este es el primer parametro para encontrar el metodo Avg  
numB - Este es el segundo parametro para encontrar el metodo Avg  
numC - Este es el tercer parametro para encontrar el metodo Avg  
  
public static void main(java.lang.String[] args)  
  
Este es el metodo principal que hace uso del metodo findAvg.  
  
Parameters:  
args - sin usar.
```

The screenshot shows the NetBeans IDE interface during the JavaDoc build process. The top window shows the Java code with Javadoc comments. The bottom window, titled "Output - FindAvg (javadoc)", displays the build logs. It shows the execution of Javadoc, loading the source file, constructing the information, building the tree for packages and classes, generating the index, and finally generating the help documentation. A warning message is visible in the log regarding a missing @return annotation in the `findAvg` method. The build is successful.

```
/*
 * <h1>Encuentra un promedio de tres numeros!</h1>
 * El programa FindAvg implementa una aplicacion que
 * simplemente calcula el promedio de tres enteros y los imprime
 * a la salida en la pantalla.
 *
 * @author Harol
 * @version 1.0
 * @since 2020-25-09
 * CIDE EVENTO COMENTARIOS
 * FUNDAMENTOS DE PROGRAMACION
 */
public class FindAvg {
    /**
     * Este metodo se usa para encontrar un promedio de tres enteros.
     */
}
```

Javadoc execution  
Loading source file C:\Users\Harol\Documents\NetBeansProjects\FindAvg\src\findavg\FindAvg.java...  
Constructing Javadoc information...  
Standard Doclet version 1.8.0\_261  
Building tree for all the packages and classes...  
C:\Users\Harol\Documents\NetBeansProjects\FindAvg\src\findavg\FindAvg.java:32: warning: no @return  
 public int findAvg(int numA, int numB, int numC)  
Building index for all the packages and classes...  
Building index for all classes...  
Generating C:\Users\Harol\Documents\NetBeansProjects\FindAvg\dist\javadoc\help-doc.html...  
1 warning  
Browsing: file:/C:/Users/Harol/Documents/NetBeansProjects/FindAvg/dist/javadoc/index.html  
javadoc:  
BUILD SUCCESSFUL (total time: 1 second)

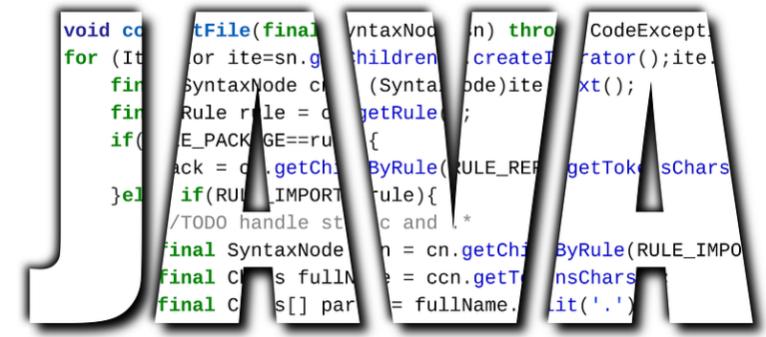




*Un bucle while es una sentencia de control de flujo que permite que el código se ejecute repetidamente en función de una condición booleana dada. El bucle while se puede considerar como una instrucción if repetitiva.*

# BUCKLES WHILE

# JAVA





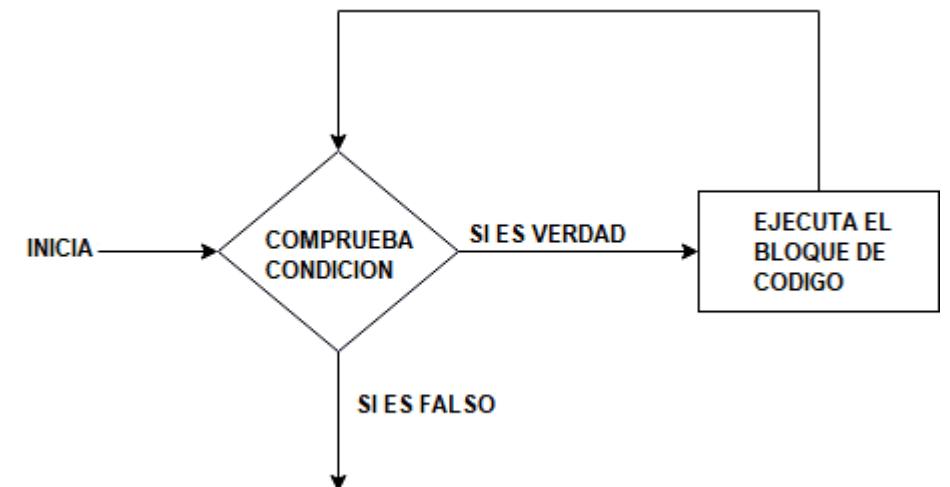
*El WHILE comienza con la verificación de la condición. Si se evalúa como verdadero, las instrucciones del cuerpo del bucle se ejecutan; de lo contrario, se ejecuta la primera instrucción que le sigue al bucle. Por esta razón, también se llama bucle de Control de entrada.*

*Una vez que la condición se evalúa como verdadera, se ejecutan las instrucciones en el cuerpo del bucle.*

*Normalmente, las declaraciones contienen un valor de actualización para la variable que se procesa para la siguiente iteración.*

*Cuando la condición se vuelve falsa, el ciclo finaliza y marca el final de su ciclo de vida.*

# BUCKLES WHILE JAVA





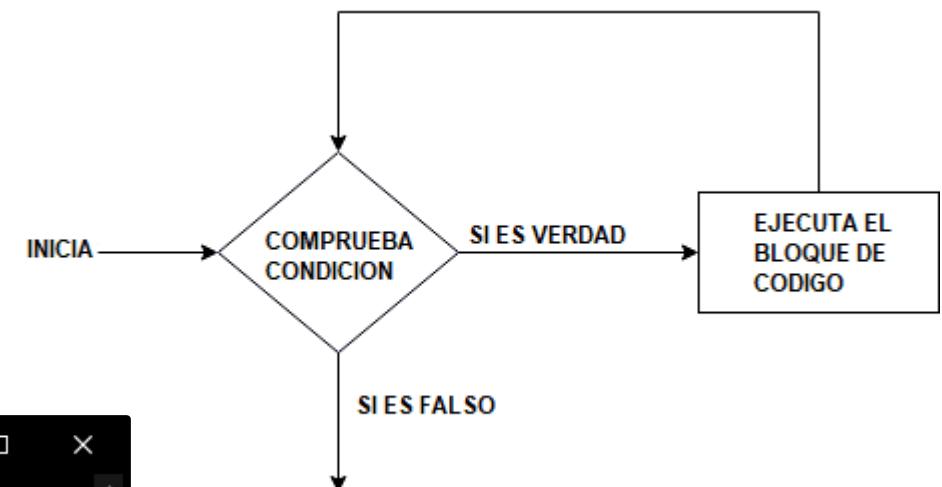
```
CIDE_CICLO WHILE - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config> Start Page FindAvg.java CIDE_CICLO WHILE
Source History < > < > < >
1  /*
2   * To change this license
3   * To change this template
4   * and open the template in
5   */
6 package cide_ciclo_while;
7
8 /**
9 */

INICIA → DIAMANTE: COMPRUEBA CONDICION
SI ES VERDAD → Caja: EJECUTA EL BLOQUE DE CODIGO
SI ES FALSO → Salida: ↓
```

```
C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO WHILE\dist>java -jar CIDE_CICLO WHILE.jar
Valor de x: 1
Valor de x: 2
Valor de x: 3
Valor de x: 4

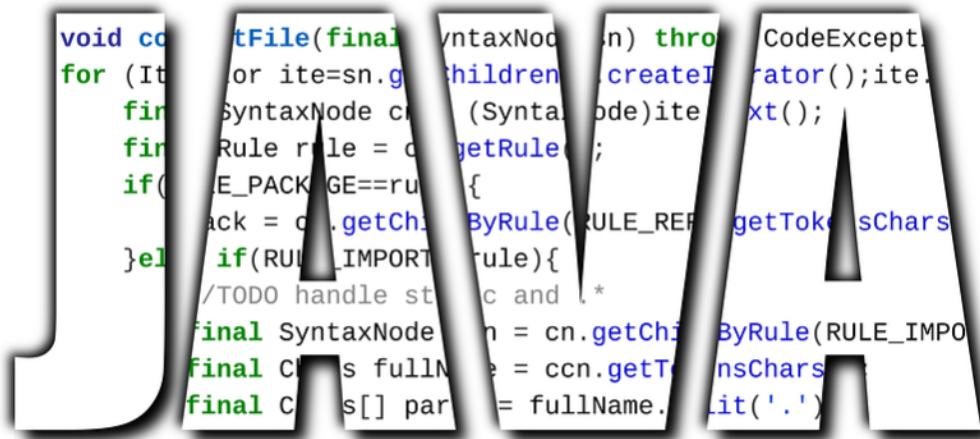
C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO WHILE\dist>
```

# BUCLES WHILE JAVA





# BUCLES FOR JAVA



A large Java logo is formed by several lines of Java code. The letters are stylized and overlap each other to create the shape of the letter 'J'.

```
void co
for (It
    fin
    fin
    if(
        E_PACK
    }el
        if(RU
        /TODO han
        final Sy
        final Cl
        final C
            tFile(final SyntaxNode cn) throws CodeExcepti
                or ite=sn.getChil
                    children.createIterator();ite.
                        SyntaxNode c
                        (SyntaxNode)ite
                        Rule rule = c
                        if(C
                            pack = c
                            if(RU
                                rule){
                                    /TODO handle st
                                    final SyntaxNode cn = cn.getChil
                                    final Class fullN
                                    final Class[] par
                        CodeExcepti
                            or ite=sn.getChil
                                children.createIterator();ite.
                                    SyntaxNode c
                                    (SyntaxNode)ite
                                    Rule rule = c
                                    if(C
                                        pack = c
                                        if(RU
                                            rule){
                                                /TODO han
                                                final SyntaxNode cn = cn.getChil
                                                final Class fullN
                                                final Class[] par

```

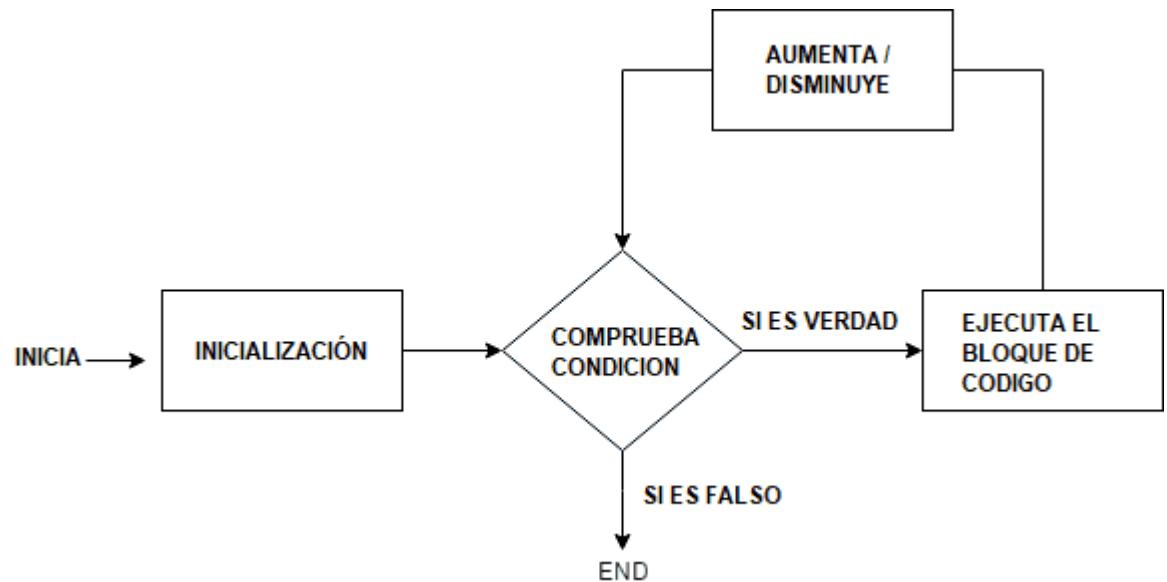


El bucle **FOR** proporciona una forma concisa de escribir la estructura de bucle. A diferencia de un ciclo **WHILE**, una sentencia **FOR** consume la inicialización, la condición y el incremento/decremento en una línea, proporcionando así una estructura de bucle más corta y fácil de depurar.





# BUCLLES FOR JAVA



**condición de inicialización:** Aquí, inicializamos la variable en uso. Marca el inicio de un ciclo For. Se puede usar una variable ya declarada o se puede declarar una variable, solo local para el bucle.



**Condición de prueba:** se usa para probar la condición de salida de un bucle. Debe devolver un valor booleano. También es un bucle de control de entrada cuando se verifica la condición antes de la ejecución de las instrucciones de bucle.

**Ejecución de instrucción:** una vez que la condición se evalúa como verdadera, se ejecutan las instrucciones en el cuerpo del bucle.

**Incremento/Decremento:** se usa para actualizar la variable para la siguiente iteración.

**Terminación de bucle:** cuando la condición se vuelve falsa, el bucle termina marcando el final de su ciclo de vida.





# BUCLES FOR JAVA

```
* @author Harol
*/
// CIDE CICLO FOR
// FUNDAMENTOS DE PROGRAMACION
public class CIDE_CICLO_FOR {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        // bucle for comienza cuando x=2
        // y corre hasta x <=4
        for (int x = 2; x <= 4; x++) {
            System.out.println("Valor de x: " + x);
        }
    }
}
```

```
C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO_FOR\dist>java -jar CIDE_CICLO_FOR.jar
Valor de x: 2
Valor de x: 3
Valor de x: 4

C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO_FOR\dist>
```





# BUCLLES FOR JAVA

```
Output - CIDE_CICLO_FOR (clean.jar) X
ant -f C:\\\\Users\\\\Harol\\\\Documents\\\\NetBeansProjects\\\\CIDE_CICLO_FOR -Dnb.internal.action.name=rebuild clean jar
init:
deps-clean:
Updating property file: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build\\built-clean.properties
Deleting directory C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build
clean:
init:
deps-jar:
Created dir: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build
Updating property file: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build\\built-jar.properties
Created dir: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build\\classes
Created dir: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build\\empty
Created dir: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build\\generated-sources\\ap-source-output
Compiling 1 source file to C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build\\classes
compile:
Created dir: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\dist
Copying 1 file to C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\build
Nothing to copy.
Building jar: C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\dist\\CIDE_CICLO_FOR.jar
To run this application from the command line without Ant, try:
java -jar "C:\\Users\\Harol\\Documents\\NetBeansProjects\\CIDE_CICLO_FOR\\dist\\CIDE_CICLO_FOR.jar"
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```



# BUCKLES FOR EACH JAVA

```
void co tFile(final SyntaxNode cn) throws CodeExcepti  
for (It or ite=cn.getChil dren().createIter ator();ite.  
fin SyntaxNode c (SyntaxNode)ite  
fin Rule rule = c getRule();  
if(E_PACK AGE==rule){  
ack = c .getCh ByRule(RULE_REF) getTok sChars  
}el if(RULE_IMPORT){  
/TODO handle st c and /*  
final SyntaxNode n = cn.getCh ByRule(RULE_IMPO  
final Class fullN e = ccn.getTok nsChars  
final Class[] par = fullName. lit('.')
```



Java también incluye otra versión del bucle `for` introducido en Java 5. La mejora del bucle `for` proporciona una forma más sencilla de iterar a través de los elementos de una colección o matriz. Es inflexible y debe usarse solo cuando existe la necesidad de recorrer los elementos de forma secuencial sin conocer el índice del elemento procesado actualmente.





# BUCLES FOR EACH JAVA

```
for (Elemento T:Colección de obj/array)
{
    declaraciones
}
```

```
C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO_FOR_EACH\dist>java -jar CIDE_CICLO_FOR_EACH.jar
Harol
Torres
CIDE

C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO_FOR_EACH\dist>
```

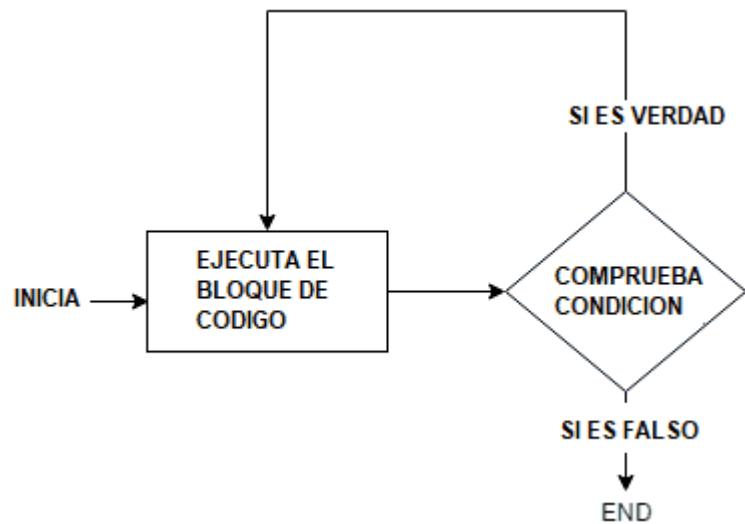


```
/*
 * @author Harol
 */
// CIDE  CICLO FOR EACH
// FUNDAMENTOS DE PROGRAMACION
public class CIDE_CICLO_FOR_EACH {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        String array[] = {"Harol", "Torres", "CIDE"};

        //mejorado para for
        for (String x:array)
        {
            System.out.println(x);
        }

        /*bucle for para la misma función
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}
```

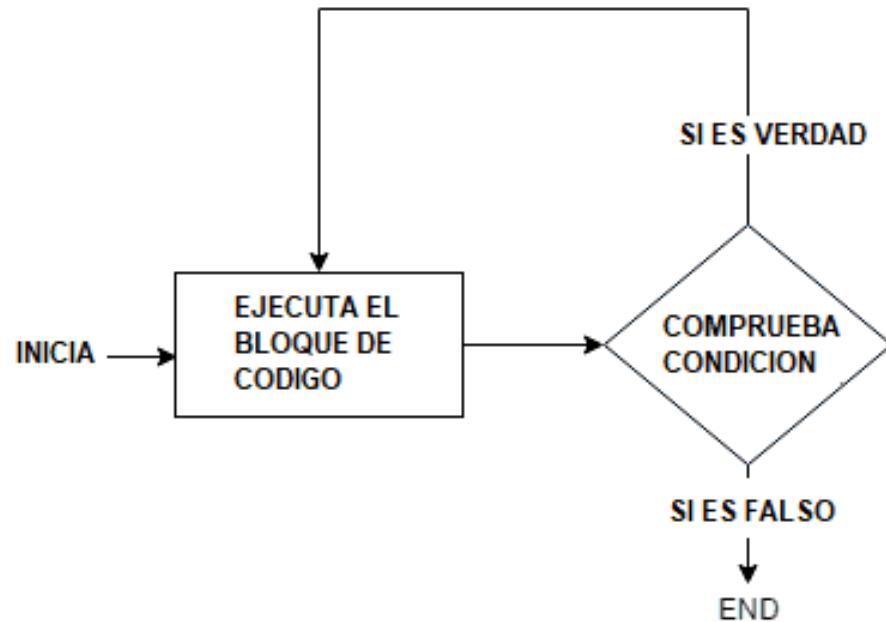


# BUCLLES DO WHILE JAVA



*El bucle do while es similar al while con la única diferencia de que comprueba la condición después de ejecutar las instrucciones, y por lo tanto es un ejemplo de Exit Control Loop (Salir del bloque de control).*





# BUCLES DO WHILE JAVA



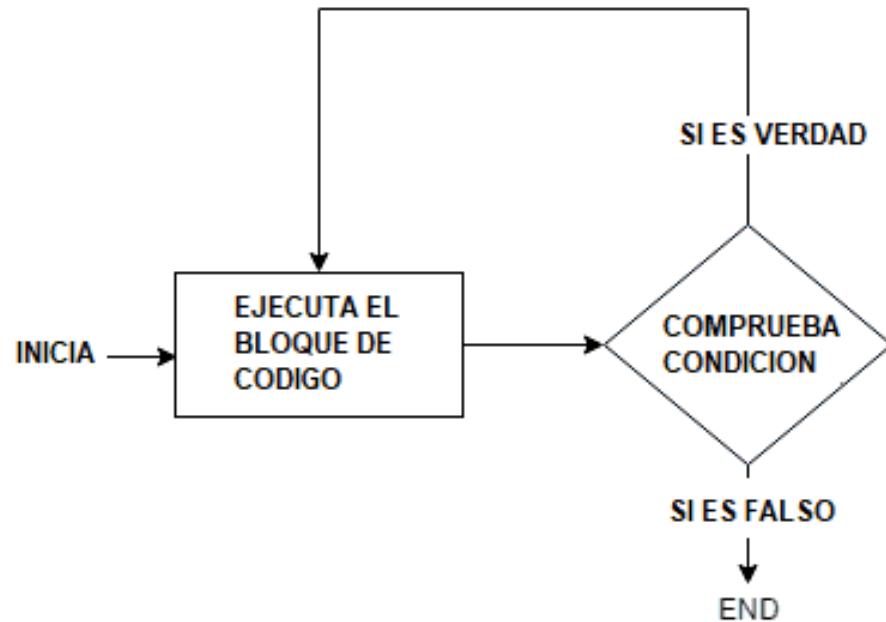
*El bucle do while comienza con la ejecución de la(s) declaración(es). No hay verificación de ninguna condición la primera vez.*

*Después de la ejecución de los enunciados, y la actualización del valor de la variable, la condición se verifica para el valor verdadero o falso. Si se evalúa como verdadero, comienza la siguiente iteración del ciclo.*

*Cuando la condición se vuelve falsa, el ciclo finaliza y marca el final de su ciclo de vida.*

*Es importante tener en cuenta que el bucle do-while ejecutará sus declaraciones al menos una vez antes de que se verifique cualquier condición, y por lo tanto es un ejemplo de bucle de control de salida.*





# BUCLES DO WHILE JAVA



*El bucle do while comienza con la ejecución de la(s) declaración(es). No hay verificación de ninguna condición la primera vez.*

*Después de la ejecución de los enunciados, y la actualización del valor de la variable, la condición se verifica para el valor verdadero o falso. Si se evalúa como verdadero, comienza la siguiente iteración del ciclo.*

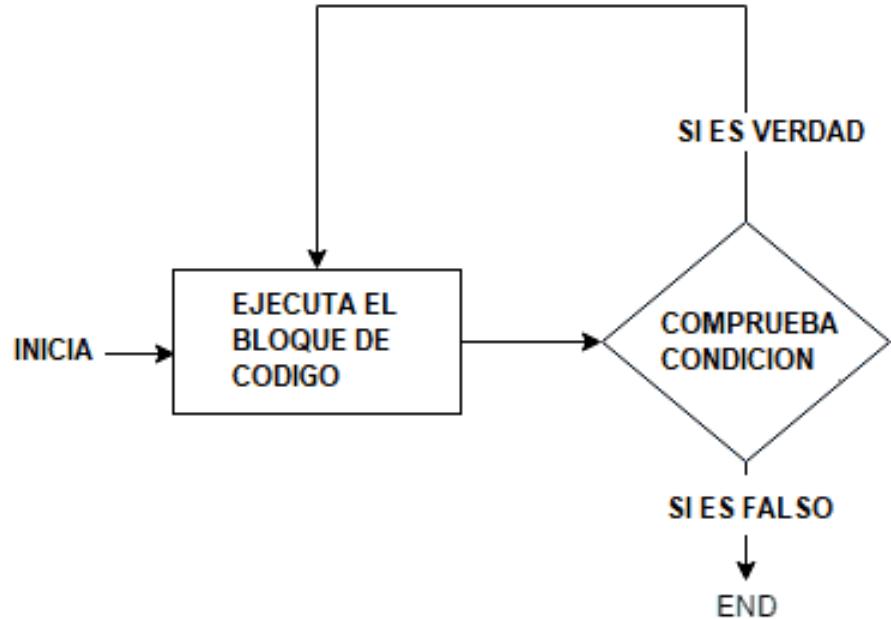
*Cuando la condición se vuelve falsa, el ciclo finaliza y marca el final de su ciclo de vida.*

*Es importante tener en cuenta que el bucle do-while ejecutará sus declaraciones al menos una vez antes de que se verifique cualquier condición, y por lo tanto es un ejemplo de bucle de control de salida.*





# BUCLES DO WHILE JAVA



```
public static void main(String[] args) {
    // TODO code application logic here
    int x = 21;
    do
    {
        //El código dentro del do se imprime incluso
        //si la condición es falsa
        System.out.println("Valor de x :" + x);
        x++;
    }
    while (x < 20);
}
```

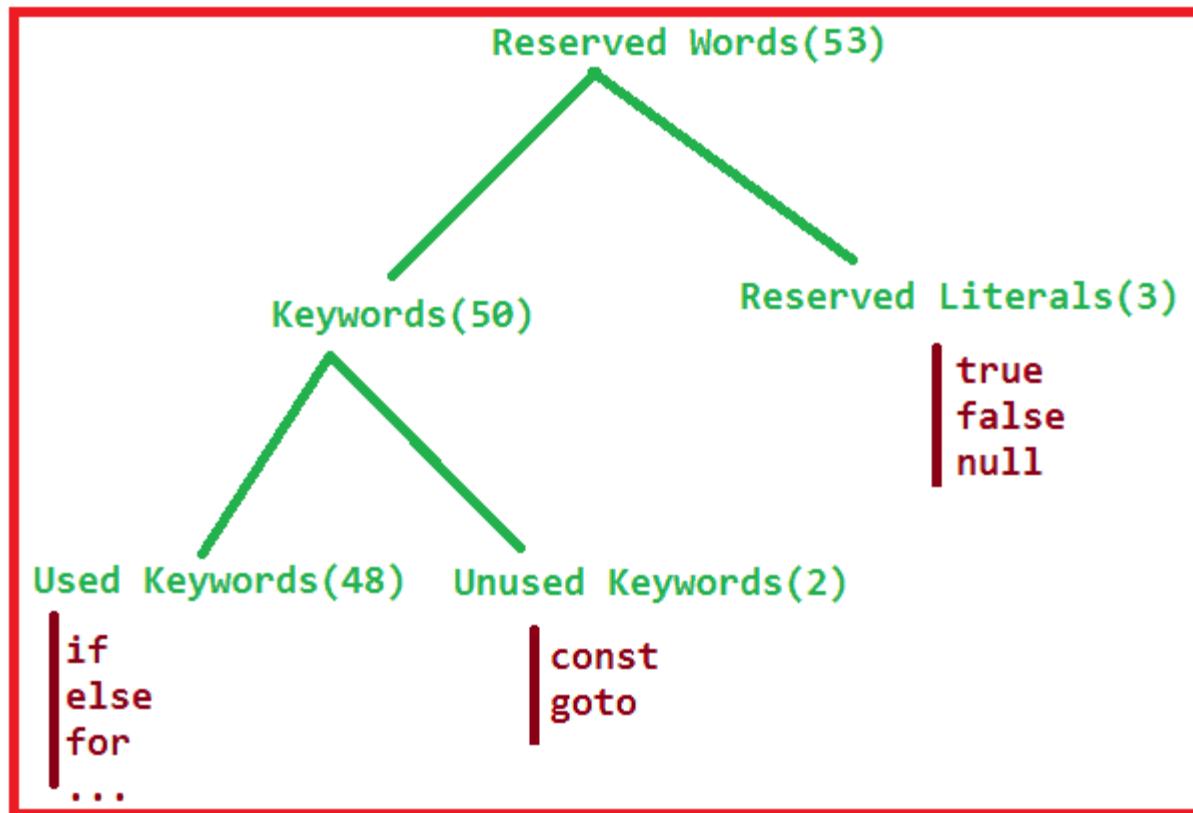
cide\_cdo\_do\_while.CIDE\_CICLO\_DO\_WHILE >> main >> do ... while (x < 20) >>

out - CIDE_CICLO_DO_WHILE (run) X
run: Valor de x :21 BUILD SUCCESSFUL (total time: 0 seconds)

```
C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO_DO_WHILE\dist>java -jar CIDE_CICLO_DO_WHILE.jar
Valor de x :21

C:\Users\Harol\Documents\NetBeansProjects\CIDE_CICLO_DO_WHILE\dist>
```





# PALABRAS RESERVADAS EN JAVA

*Las palabras clave definen funcionalidades y las literales definen un valor.*





# PALABRAS RESERVADAS EN JAVA

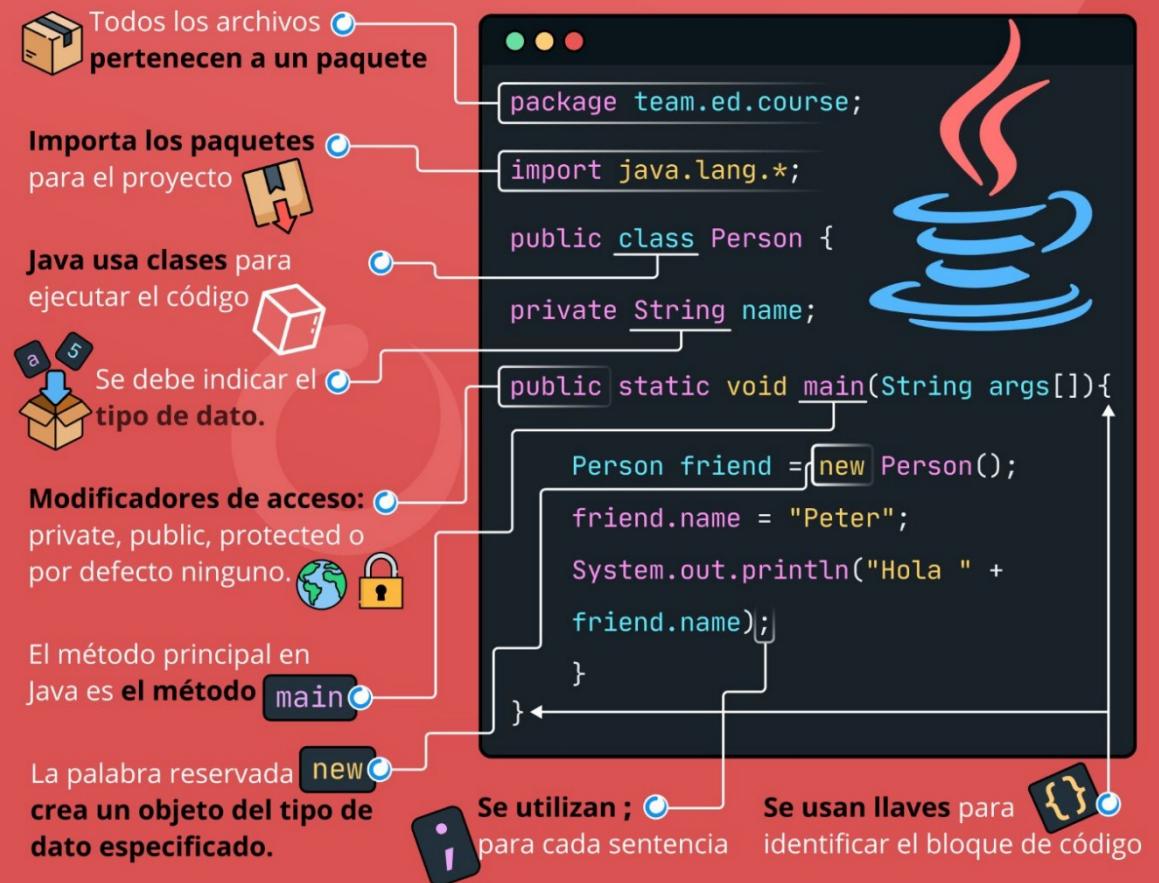
Tabla de Palabras Claves Reservadas en Java

abstract	assert	boolean	break	byte	case	catch	char	class	const	continue
default	do	double	else	enum	extends	false	final	finally	float	for
goto	if	implements	import	instanceof	int	interface	long	native	new	null
package	private	protected	public	return	short	static	strictfp	String	super	switch
synchronized	this	throw	throws	transient	true	try	void	volatile	while	





# SINTAXIS BÁSICA DE JAVA



Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/java](http://ed.team/java)

 EDteam



# 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones creada para resolver una necesidad específica.**



## 1 GOOGLE GUAVA



### Mejora del flujo de trabajo

destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

## 2 RXJAVA



Crea aplicaciones para **reaccionar** a los flujos de datos en tiempo real.

## 3 MPANDROIDCHART



Crea **gráficos** de líneas, barras, radares y burbujas para Android.

## 4 FASTJSON



Convierte **objetos** Java en su representación JSON y viceversa.

## 5 MOCKITO



Librería de código abierto para **simular pruebas unitarias**.

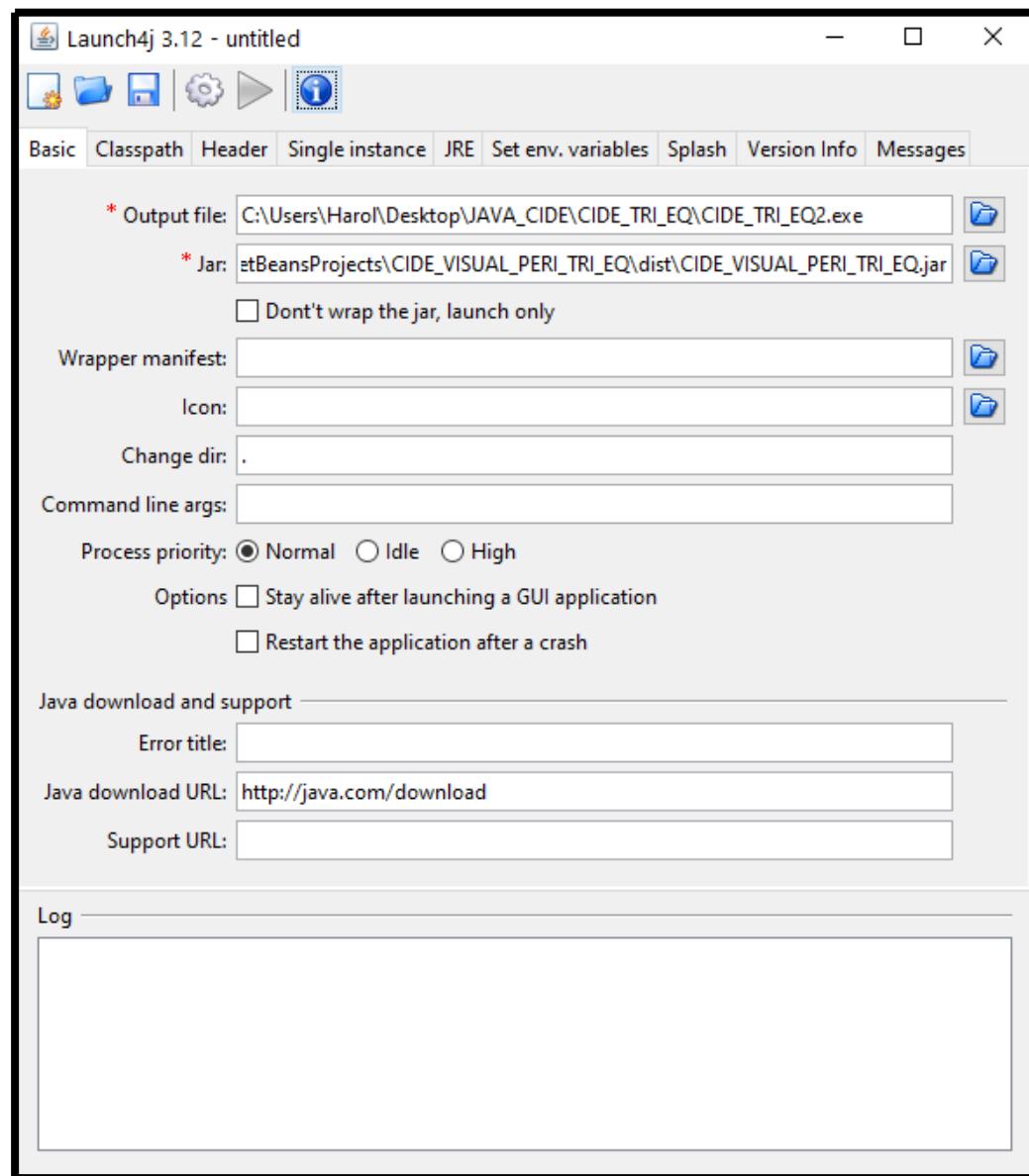


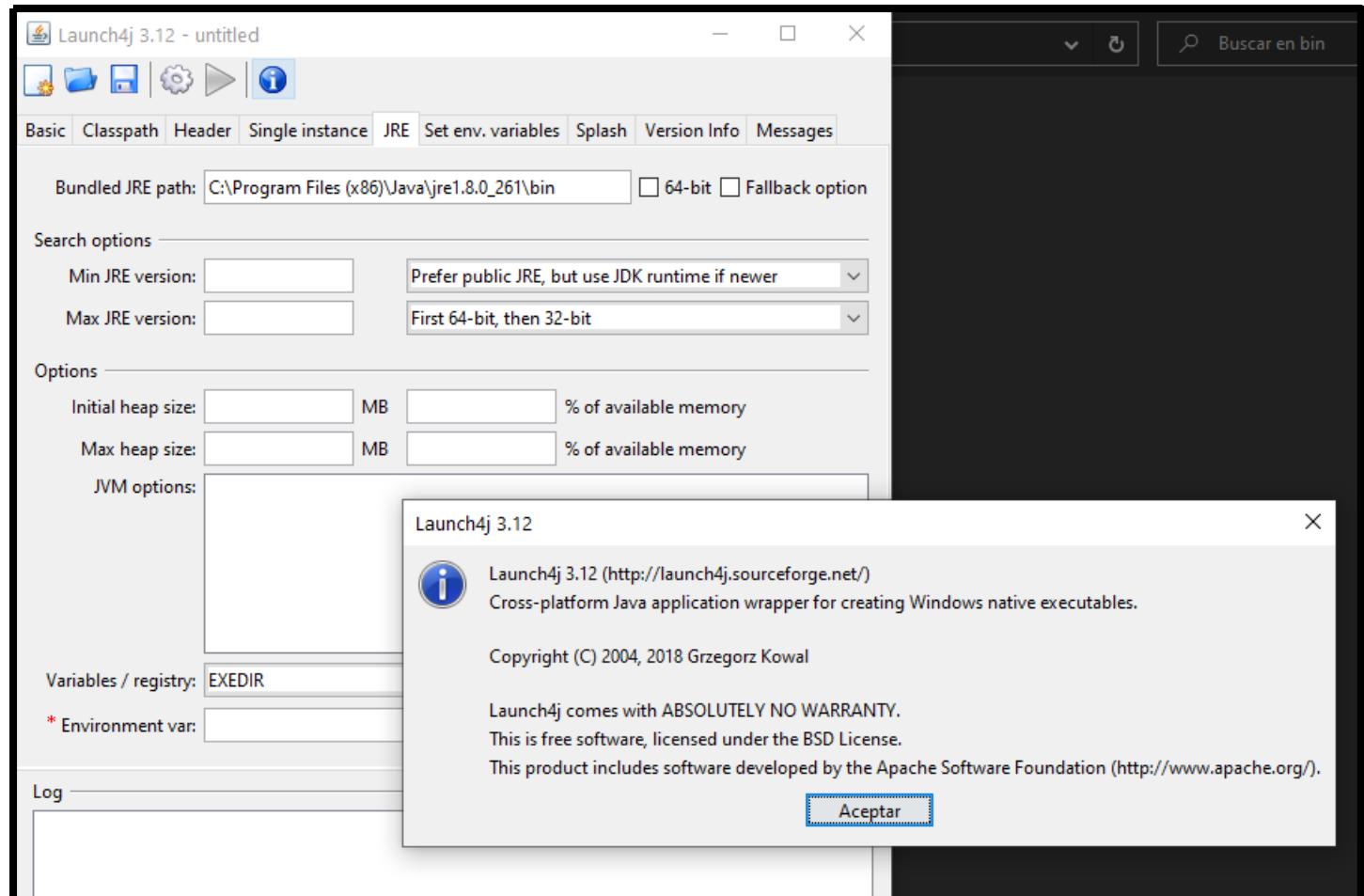
Esta es la librería más **usada para pruebas**.

Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/cursos/java](http://ed.team/cursos/java)









# ENTREGA INFORME

## CRITERIOS DE ACEPTACIÓN

1. PORTADA
2. TABLA DE CONTENIDO
3. PRESENTACION DEL PROBLEMA
4. PSEUDOCODIGO
5. DIAGRAMA DE FLUJO
6. PRUEBA DE ESCRITORIO
7. CODIGO JAVA (COMENTARIOS)
8. COMPILACION Y GENERACION DE .JAR
9. VIDEO PUBLICADO EN YOUTUBE CON LA EXPLICACION – MINIMO 7 MINUTOS.
10. PUBLICACION GITHUB / PROYECTO
11. CONCLUSIONES
12. BIBLIOGRAFIA (NORMA APA)



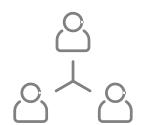
# CONCLUSIONES

.





**¿ PREGUNTAS ?**



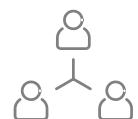
# Gracias



Politécnico  
Internacional



**“Celebra cada  
pequeña victoria.”**



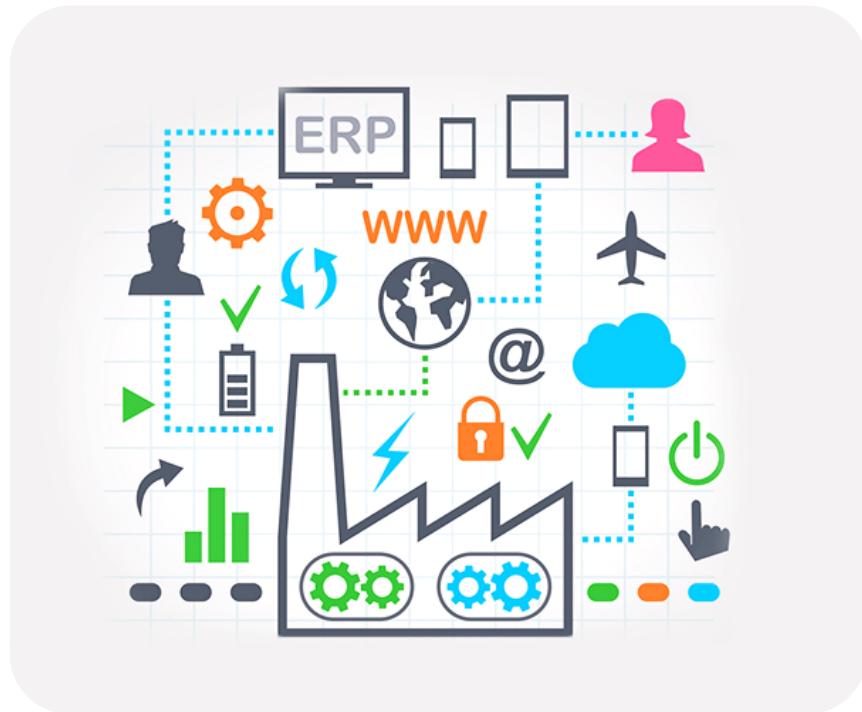
**ANTES DE  
EMPEZAR**



# AGENDA

- Retroalimentación - Clase Anterior*
- Actividades Pendientes*
- Informe Final*
- Presentación informe Final*
- Presentación 18 ejercicios / Notas*
- Notas / Preguntas*
- Conclusiones*





1 Sábados  
2 Break

8 AM a 12 M - 4 Horas  
9:30 AM a 09:45 AM

## ESPACIOS DE FORMACIÓN

# A TENER EN CUENTA



1. Cerrar los micrófonos y desactivar las cámaras.



2. Solicitar la palabra mediante el chat.



3. Preguntar



## CONTENIDOS

1. Algoritmos
2. Elementos lógicos de un algoritmo
3. Técnicas para la formulación de algoritmos. DFD, pseudocódigo, prueba de escritorio.
4. Introducción al Lenguaje de programación C#
5. Historia, evolución y futuro del lenguaje de programación C#
6. Introducción a la sintaxis de C#
7. Tipos de datos en C#
8. Operadores en C#
9. Estructuras de control en C#
10. Arreglos en C#
11. Manejo de excepciones en C#
12. Colecciones en C# (Espacio de nombres System.Collections)
13. LinQ con C#

# CONTENIDOS



# RETROALIMENTACIÓN: CLASE ANTERIOR

- Comentarios Java
- Ejemplos
- Ciclos
- Ejemplos





# RETROALIMENTACIÓN: CLASE ANTERIOR

ESTUDIANTES PENDIENTES POR PRESENTACION  
EJERCICIO





# INFORME FINAL

## CRITERIOS DE ACEPTACIÓN

1. PORTADA
  2. TABLA DE CONTENIDO
  3. PRESENTACION DEL PROBLEMA
  4. PSEUDOCODIGO
  5. DIAGRAMA DE FLUJO
  6. PRUEBA DE ESCRITORIO
  7. CODIGO JAVA (COMENTARIOS)
  8. COMPIILACION Y GENERACION DE .JAR
  9. VIDEO PUBLICADO EN YOUTUBE CON LA EXPLICACION – MINIMO 7 MINUTOS.
  10. PUBLICACION GITHUB / PROYECTO
  11. CONCLUSIONES
  12. BIBLIOGRAFIA (NORMA APA)
- Arial 12 + Formato PDF





# ENTREGA TALLERES JAVA I - II

**Evaluable**

1. PSEUDOCODIGO
2. DIAGRAMA DE FLUJO
3. PRUEBA DE ESCRITORIO
4. CODIGO JAVA (COMENTARIOS)



# RETO



Sobre nosotros    Padres    Impacto    Mis clases    Mi Cuenta ▾    español

## CIDE\_FUNDAMENTOS [editar la configuración de la clase](#)

FUNDAMENTOS\_PROGRAMACION

Resumen de la clase

Idioma : JavaScript

Estudiantes : 0

Tiempo de juego de nivel medio :

Tiempo total de juego :

Niveles promedio completados :

Niveles totales completados :

Creado : 14/8/2020

Agregar estudiantes :

**NightShirtMilk**

Copiar código de clase

Los estudiantes pueden unirse a su clase usando este Código de clase. No se requiere una dirección de correo electrónico al crear una cuenta de estudiante con este código de clase.

<https://codecombat.com/stude>

Copiar URL de clase

También puede publicar esta URL de clase única en una página web compartida.



# RETO

## FLEXBOX FROGGY

Nivel 1 de 24

Bienvenido a Flexbox Froggy, un juego donde ayudarás a Froggy ya sus amigos escribiendo código CSS. Guía a esta rana hacia la hoja de lirio en la derecha, usando la propiedad **justify-content**, la cual alinea elementos horizontalmente y acepta los siguientes valores:

- **flex-start**: Alinea elementos al lado izquierdo del contenedor.
- **flex-end**: Alinea elementos al lado derecho del contenedor.
- **center**: Alinea elementos en el centro del contenedor.
- **space-between**: Muestra elementos con la misma distancia entre ellos.
- **space-around**: Muestra elementos con la misma separación alrededor de ellos.

Por ejemplo, **justify-content: flex-end;** moverá la rana a la derecha.

```
1 #pond {  
2   pantalla: flex;  
3   justify-content: flex-end;  
4 }  
5  
6
```

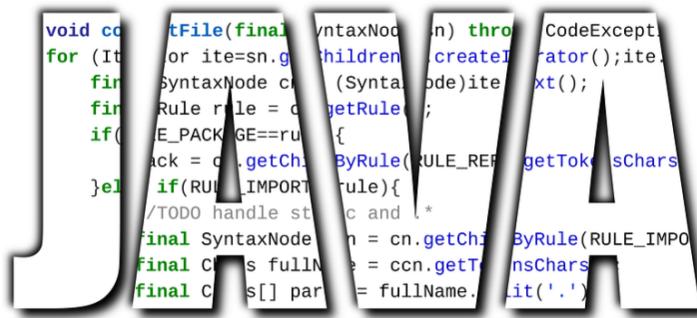




**CLASE SCANNER**



Scanner es una clase en el Paquete **JAVA.UTIL** utilizada para obtener la entrada de los tipos primitivos como int, double etc. y también **STRING**. Es la forma más fácil de leer datos en un programa Java, aunque no es muy eficiente si se quiere un método de entrada para escenarios con validación.



## IMPORTANTE

Para crear un objeto de clase `Scanner`, normalmente pasamos el objeto predefinido `System.in`, que representa el flujo de entrada estándar. Podemos pasar un objeto de clase `File` si queremos leer la entrada de un archivo.

Para leer valores numéricos de un determinado tipo de datos `XYZ`, la función que se utilizará es `Nextxyz()`. Por ejemplo, para leer un valor de tipo `short`, podemos usar `Nextshort()`.

Para leer cadenas (`STRINGS`), usamos `Nextline()`.

Para leer un solo carácter, se usa `next().charAt(0)`. La función `next()` devuelve el siguiente token/palabra en la entrada como cadena y la función `charAt (0)` devuelve el primer carácter de esa cadena.



## CÓDIGO

The screenshot shows an IDE interface with several tabs at the top: ...ava, CIDE\_CICLO\_DO WHILE.java, CIDE\_VISUAL\_PERI\_EQ.java, and CIDE\_ENTRADA\_SCANNER.java. The CIDE\_ENTRADA\_SCANNER.java tab is active, displaying the following Java code:

```
35     long mobileNo = sc.nextLong();
36     double average = sc.nextDouble();
37
38     // Imprima los valores para verificar si la entrada
39     // fue obtenida correctamente.
40     System.out.println("Nombre Estudiante CIDE: "+name);
41     System.out.println("Género Estudiante CIDE: "+gender);
42     System.out.println("Edad Estudiante CIDE: "+age);
43     System.out.println("Teléfono Estudiante CIDE: "+mobileNo);
44     System.out.println("Promedio Estudiante CIDE: "+average);
45
46 }
47 }
```

Below the code editor, the status bar shows the file name cide\_entrada\_scanner.CIDE\_ENTRADA\_SCANNER and the main method. The output window below shows the program's execution:

Output - CIDE\_ENTRADA\_SCANNER (run) ×

```
run:
harol
masculino
35
22222
5
Nombre Estudiante CIDE: harol
Género Estudiante CIDE: m
Edad Estudiante CIDE: 35
Teléfono Estudiante CIDE: 22222
Promedio Estudiante CIDE: 5.0
BUILD SUCCESSFUL (total time: 24 seconds)
```



## **HASNEXTXYZ()**

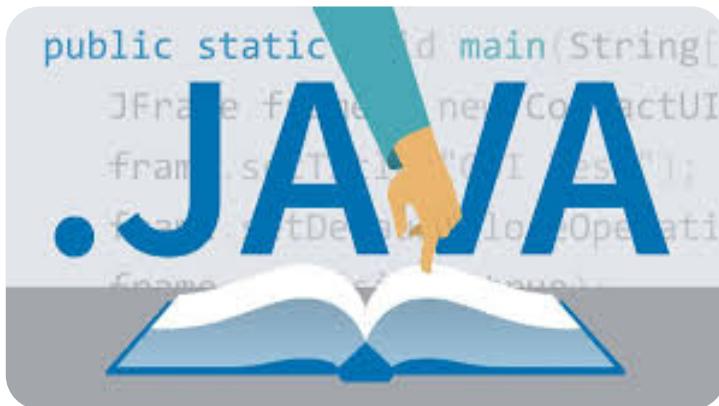
A veces, debemos verificar si el siguiente valor que leemos es de cierto tipo o si la entrada ha finalizado (se encontró el marcador EOF).

Luego, se verifica si la entrada del Scanner es del tipo que queremos con la ayuda de las funciones **hasNextXYZ()** donde *XYZ* es del tipo que nos interesa. La función devuelve true (verdadero) si el scanner tiene un *token* de ese tipo, de lo contrario es false (falso).

En el siguiente código, se utiliza **hasNextInt()**. Para verificar una cadena, usamos **hasNextLine()**. De manera similar, para verificar si hay un solo carácter, se usa **hasNext().CharAt(0)**.



## CÓDIGO



The screenshot shows an IDE interface with two tabs: "CIDE\_ENTRADA\_SCANNER.java" and "CIDE\_ENTRADA\_SCANNER\_HASTNETXYZ.java". The "CIDE\_ENTRADA\_SCANNER\_HASTNETXYZ.java" tab is active, displaying the following Java code:

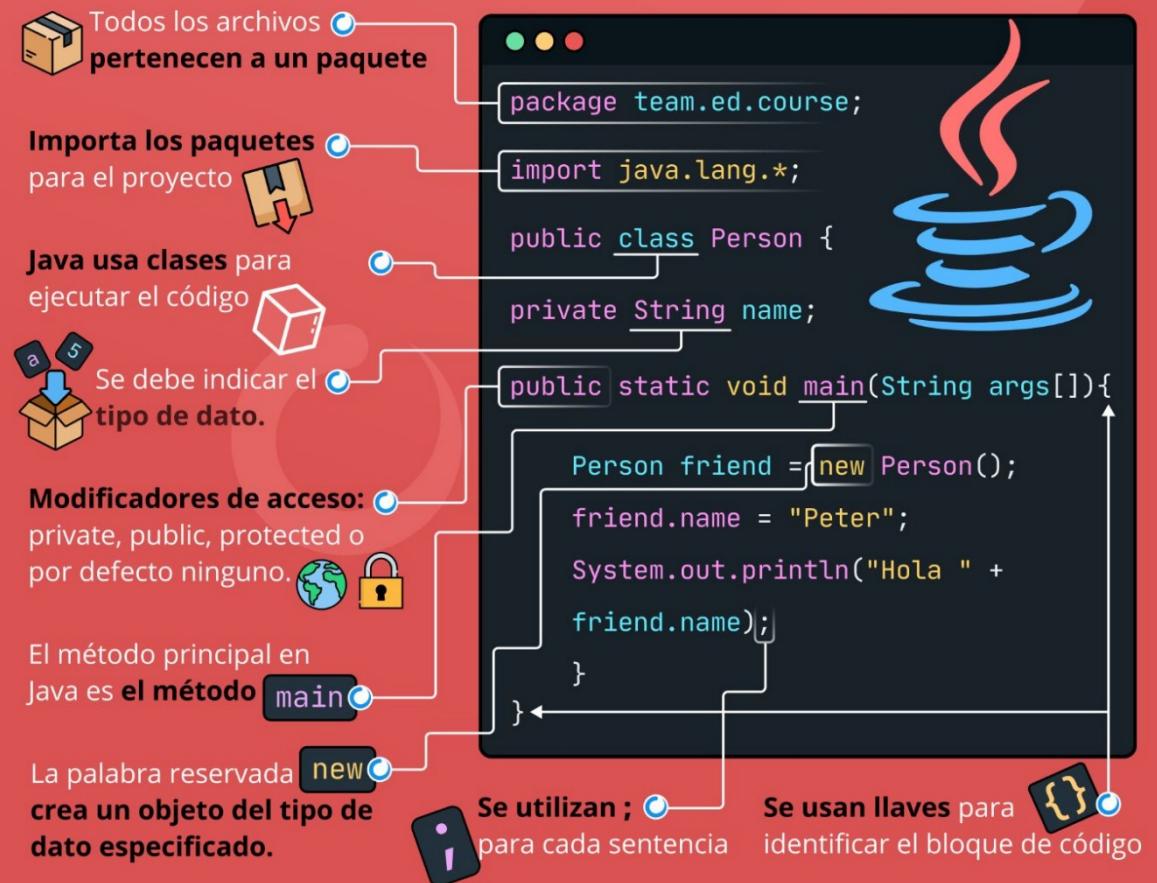
```
10  * @author Harol
11  */
12  // CIDE ENTRADA DE DATOS
13  // FUNDAMENTOS DE PROGRAMACION
14  public class CIDE_ENTRADA_SCANNER_HASTNETXYZ {
15
16      /**
17      * @param args the command line arguments
18      */
19      public static void main(String[] args) {
20          // TODO code application logic here
21
22          // Declarar un objeto e inicializar con
23          // el objeto de entrada estndar predefinido
24          Scanner sc = new Scanner(System.in);
25
26          // Inicializar la suma y el recuento
27          // de los elementos de entrada
28          int sum = 0, count = 0;
29      }
}
```

The "Output - CIDE\_ENTRADA\_SCANNER\_HASTNETXYZ (run)" panel shows the following results:

```
run:
1
2
3.0
La media es: 1
BUILD SUCCESSFUL (total time: 3 seconds)
```



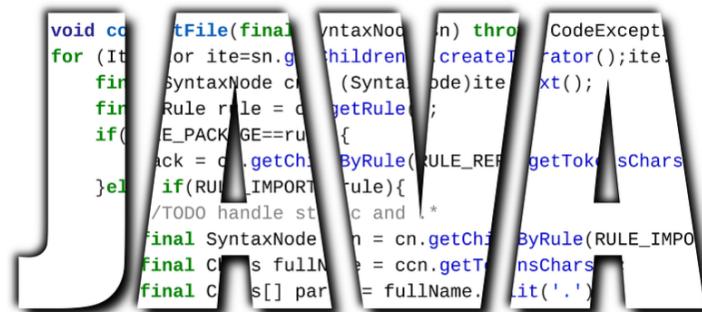
# SINTAXIS BÁSICA DE JAVA



Aprende Java desde cero hasta nivel Jedi en:

 [ed.team/java](http://ed.team/java)

 EDteam



# 6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones** creada para resolver una necesidad específica.



## 1 GOOGLE GUAVA



### Mejora del flujo de trabajo

destinado a tareas comunes. (E/S, caché, primitivas, entre otros).

## 2 RXJAVA



Crea aplicaciones para reaccionar a los flujos de datos en tiempo real.

## 3 MPANDROIDCHART



Crea gráficos de líneas, barras, radares y burbujas para Android.

## 4 FASTJSON



Convierte objetos Java en su representación JSON y viceversa.

## 5 MOCKITO



Librería de código abierto para simular pruebas unitarias.

## 6 JUNIT

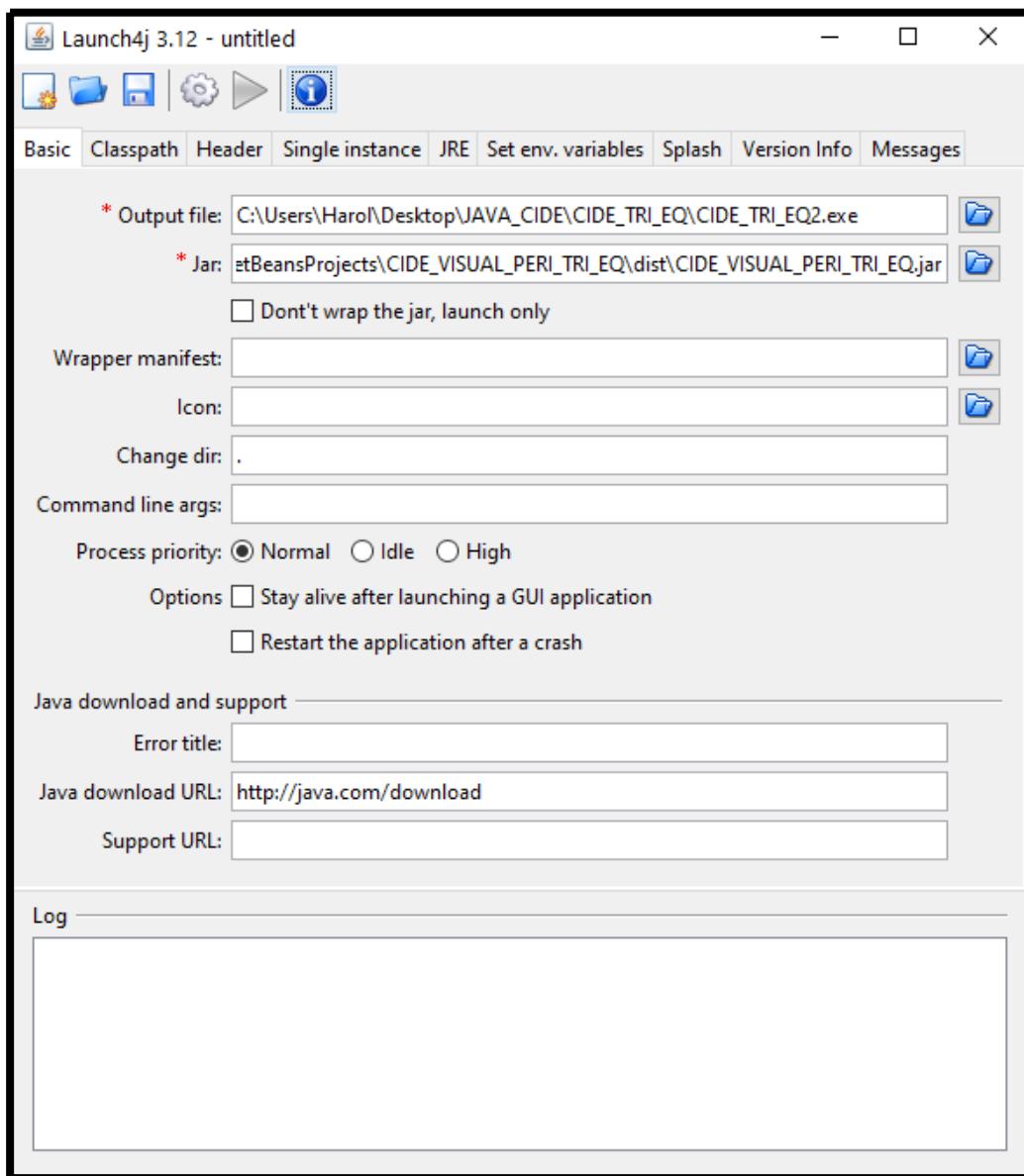


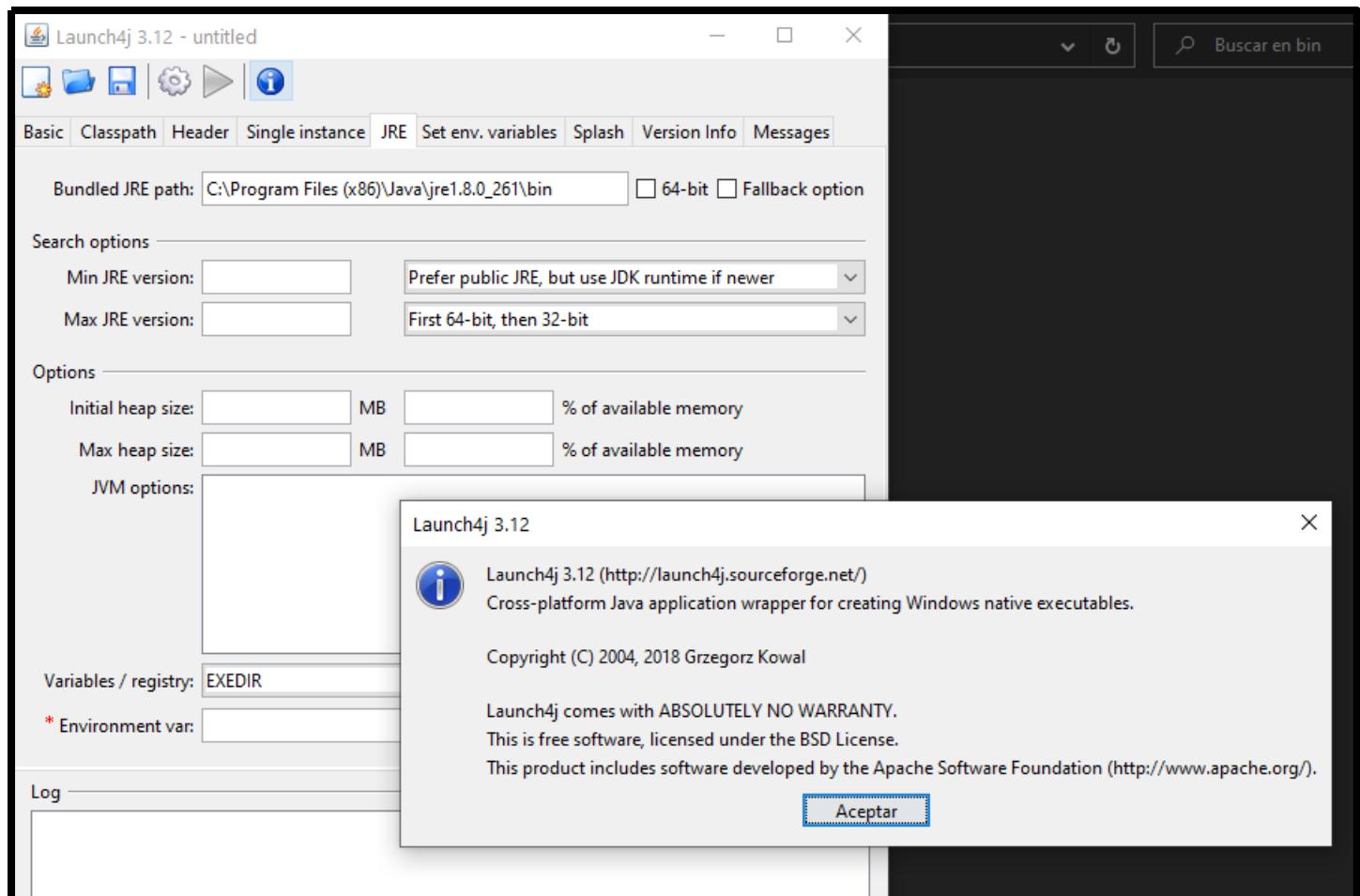
Esta es la librería más usada para pruebas.

Aprende Java desde cero hasta nivel Jedi en:

[ed.team/cursos/java](http://ed.team/cursos/java)





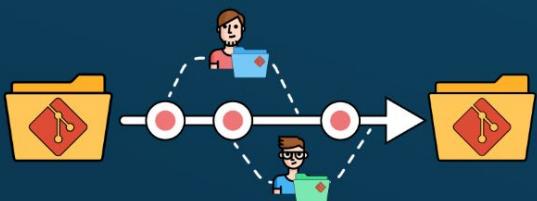




# ¿SABES QUÉ ES GIT?



Es un sistema de **control de versiones**. Lleva un registro de todos los **cambios y avances** de tu proyecto.



## GIT TRABAJA CON RAMAS

Ayuda a que **varias personas trabajen** en un mismo proyecto y pueden realizar **modificaciones sin afectar a los demás** archivos. Una vez que estén listos los cambios se **fusionan con la rama principal**.

Prof. Beto Quiroga



Todo desarrollador sin importar el lenguaje **debe dominar Git**.

v1 v2 v3

Funciona como una **máquina de tiempo**, puedes ir al pasado de tu código o volver al presente.



**Github** es un servicio que te ayuda a **almacenar tu proyecto en la nube**, además existen otros servicios como **Gitlab o Bitbucket**.

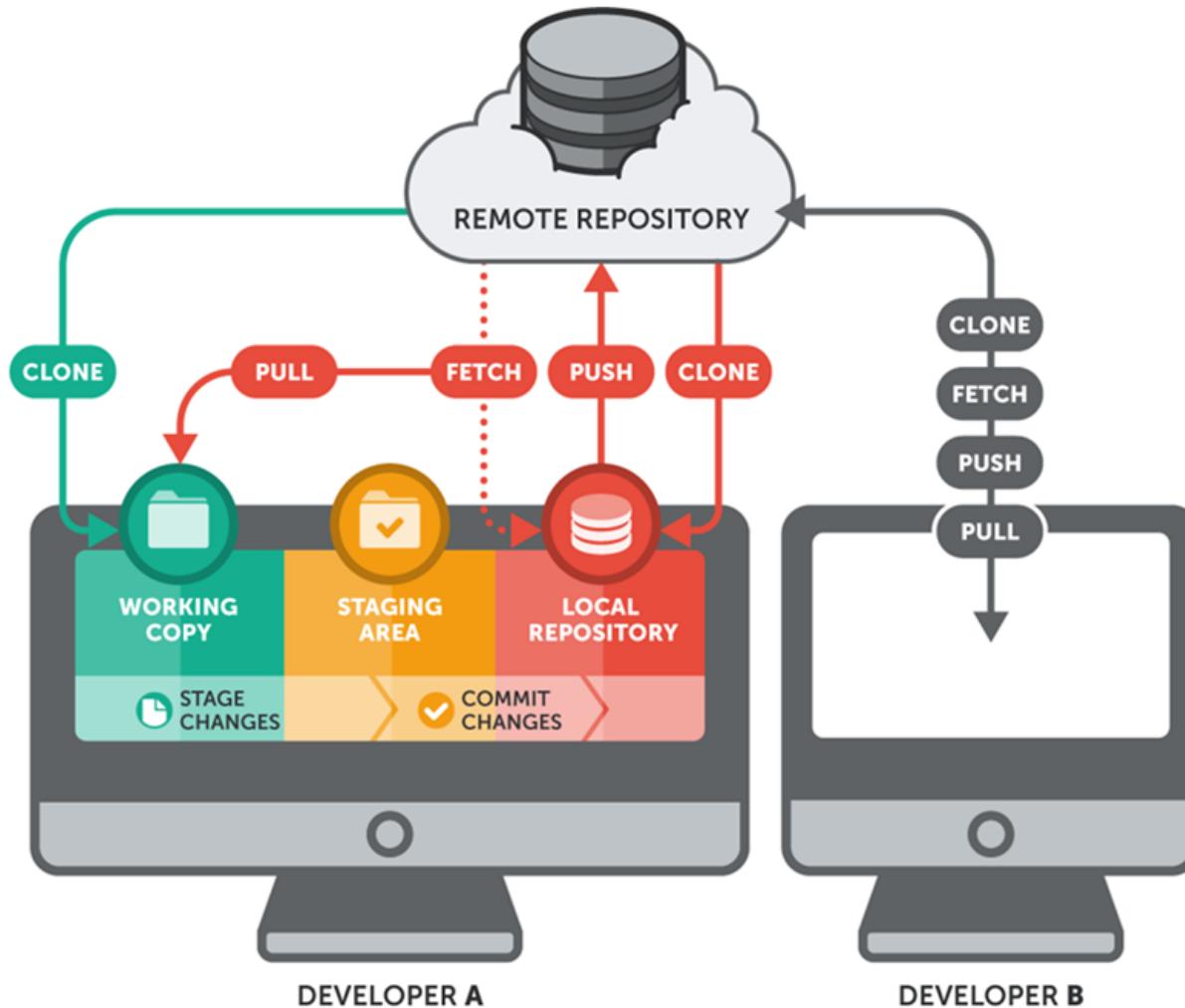


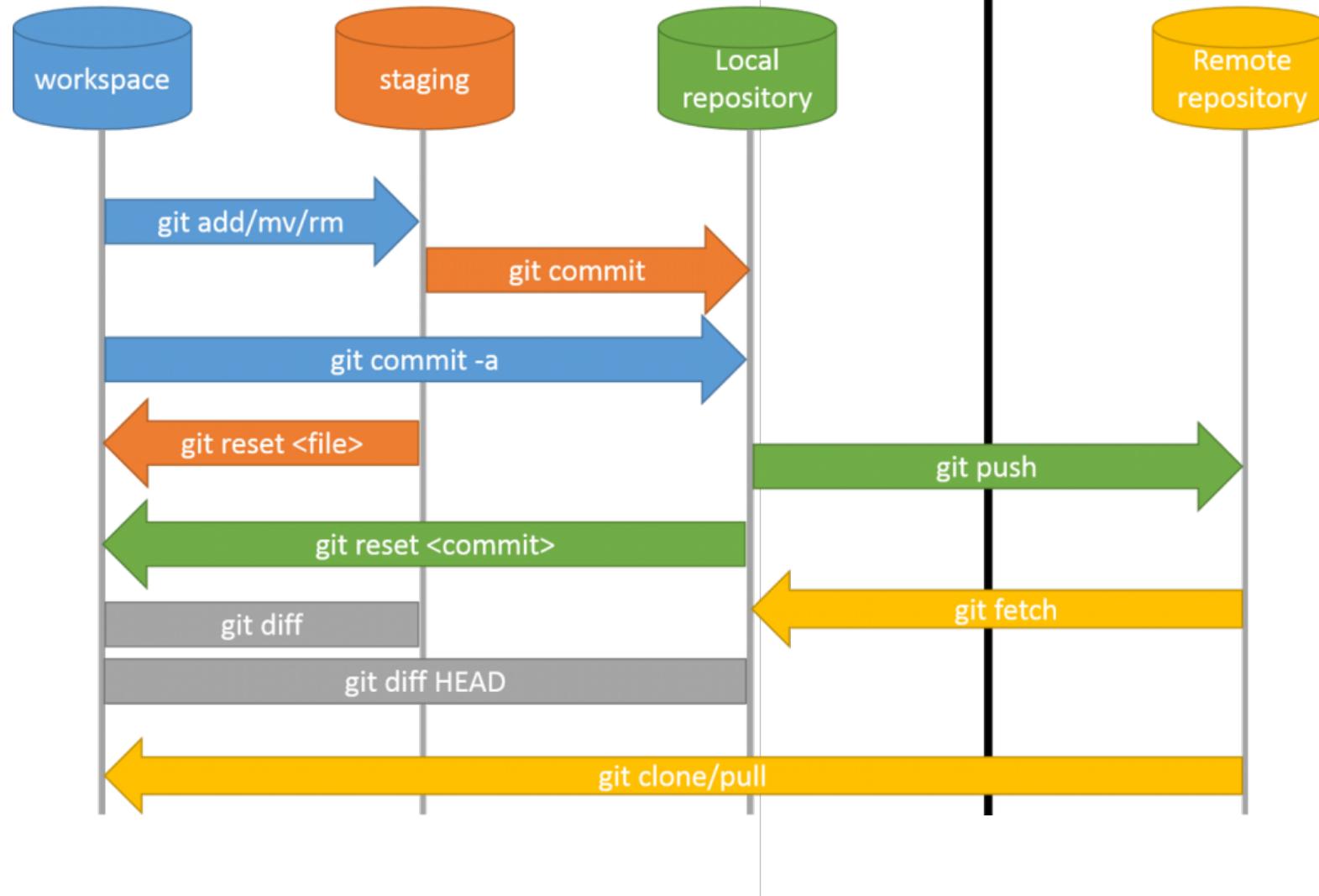
Aprende a trabajar en equipo con Git en:

> [ed.team/cursos/git-workflow](https://ed.team/cursos/git-workflow)



EDteam







# COMANDOS BÁSICOS DE git



**GIT** es un sistema de control de versiones que nos ayuda a llevar el historial completo de modificaciones de un proyecto.

Todo desarrollador sin importar el lenguaje **debe dominar Git**.  
**Prof. Beto Quiroga**

**GIT INIT**



Inicia un nuevo repositorio.

**GIT ADD**



Añade un archivo a la zona de montaje. **git add \*** añade uno o más archivos a la zona de montaje.

**GIT LOG**



Se utiliza para listar el **historial** de versiones de la rama actual.

**GIT RESET**



Descompone el **archivo**, pero conserva el contenido del mismo.

**GIT CLONE**



Clona un repositorio existente.

**GIT CONFIG**



Establece el nombre del autor, el correo y demás **parámetros** que Git utiliza por defecto.

**GIT STATUS**



Enumera todos los archivos que deben ser confirmados.

**GIT DIFF**



Muestra las **diferencias** de archivo que aún no se ponen en escena.

¿Qué comandos faltó? Haremos la 2da parte con tus sugerencias

 [ed.team/cursos/git](http://ed.team/cursos/git)





# COMANDOS BÁSICOS DE git (PARTE 2)

Si quieras trabajar en el mundo del desarrollo de software, es obligatorio que sepas GIT.

**GIT HELP**



Muestra información para saber el uso de cada comando de git.

**GIT CLEAN**



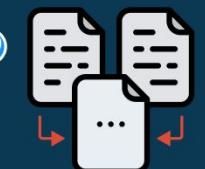
Elimina archivos no deseados de un repositorio.

**GIT BRANCH**



Muestra la lista de ramas que existen en un repositorio.

**GIT MERGE**



Fusiona dos o más ramas.

**GIT PULL**



Descarga y actualiza los cambios realizados desde un repositorio remoto a tu repositorio local.

**GIT PUSH**



Sube los archivos a un repositorio remoto.

**GIT CHECKOUT**



Sirve para moverse de una rama a otra y regresar en el tiempo.



**GIT REMOTE**

Crea, visualiza y elimina conexiones a otros repositorios.

Si eres programador es **obligatorio** aprender GIT. Domínalo en:

 [ed.team/cursos/git](https://ed.team/cursos/git)





# TÉRMINOS DE git QUE DEBES CONOCER

## REPOSITORY/REPO



Base de datos donde se almacena el historial del código.

## BRANCH / RAMA

Entorno o espacio de trabajo independiente en Git.

## COMMIT



Registro de uno o varios cambios hechos en el repositorio.

## MASTER



Rama principal de un repositorio.

## STAGE



Lista de archivos que se usarán para un commit.

## CHECKOUT



Es la acción de moverse entre diferentes ramas.

## FORK



Copia de un repositorio.

## MERGE



Combina dos o más ramas en una.

Si eres programador es **obligatorio** aprender GIT. Domínalo en:

 [ed.team/cursos/git](http://ed.team/cursos/git)



# EL MENSAJE DE COMMIT IDEAL EN GIT

The diagram shows a terminal window with a commit message:

```
$ git commit
Crea un nuevo componente para el countdown
* Reemplaza el componente que generó conflictos
* Visible solo para usuarios premium
Resuelve issue: #193"
```

Numbered callouts explain each part:

- 1 TÍTULO**: Resumen breve de los cambios realizados. No debes usar más de 50 caracteres.
- 2 CUERPO (OPCIONAL)**: Explicación más detallada que aporta contexto e información adicional.
- 3**: Puedes usar varias frases o crear listas para explicar.
- 4**: Menciones de los issues afectados o relacionados.

El mensaje del commit explica la razón detrás de los cambios.

Beto Quiroga  
COO EDteam

Encuentra más consejos como este y domina Git en:  
`> ed.team/cursos/git`



# GITHUB VS GITLAB

GitHub	GitLab
Máximo de 3 colaboradores gratis.	No hay límite de colaboradores.
Ofrece <b>hasta 500 MB</b> gratis por repositorio.	El límite gratis por repositorio es de <b>10 GB</b> .
El código se puede descargar como archivo.	Permite <b>incluir adjuntos en un issue</b> o problema.
Opción de <b>lectura o escritura por repositorio</b> .	Puedes <b>proporcionar acceso</b> a determinadas partes de un proyecto.
Supera los <b>35 millones</b> proyectos.	Más de <b>100 mil</b> proyectos.
Fue comprada por Microsoft <b>¿Bueno o malo? ¿Qué opinas?</b>	Opera bajo <b>licencia de código abierto</b> .

Aprende gestionar tus repositorios en GitHub y GitLab en:  
-> [ed.team/cursos/git-workflow](https://ed.team/cursos/git-workflow)



git

The screenshot shows a Git commit interface. On the left, a sidebar lists 'Changes' (8 changed files) with checkboxes next to each file: 'Config...\\APACHE+PHP+MYSQL.txt', 'Configuracion...E\_MYSQL\_PHP.pdf', 'Diagrama\_Arquitecto - Linux.pdf', 'Diagrama\_Arquitecto\_Focus.pdf', 'OVA\_FOCUS.pdf', 'OVA\_FOCUS.pptx', 'Proyecto\\Proyecto\_Diagrama\_gan', and 'Proyecto\\Proyecto\_Diagrama\_pdf'. Below this is a 'Summary (required)' input field and a 'Description' input field with a plus icon. At the bottom is a blue 'Commit to master' button. The main area displays a diff view for the file 'Configuracion\_Instalacion\\APACHE+PHP+MYSQL.txt'. The diff shows additions (lines 1-28) and deletions (line 0). The additions include configuration for Apache, MySQL, PHP, and UFW, along with user details and installation steps for Apache and Firewall configuration.

```
@@ -0,0 +1,108 @@
1 +*****
2 +*****
3 +*****
4 + HAROL HERNAN TORRES NEUTA
5 + UDISTRITAL
6 + CURSO LINUX
7 + 2020
8 +*****
9 +*****
10 +*****
11 +*****
12 +INSTALAR APACHE
13 +*****
14 +
15 +sudo apt-get update
16 +sudo apt-get install apache2
17 +sudo passwd root
18 +su root
19 +
20 +*****
21 +CONFIGURAR FW
22 +*****
23 +
24 +sudo ufw status
25 +sudo ufw enable
26 +sudo ufw allow 80
27 +sudo ufw allow 445
28 +sudo ufw app list
29 +sudo ufw allow in "Apache Full"
```



# ENTREGA INFORME

## CRITERIOS DE ACEPTACIÓN

1. PORTADA
2. TABLA DE CONTENIDO
3. PRESENACION DEL PROBLEMA
4. PSEUDOCODIGO
5. DIAGRAMA DE FLUJO
6. PRUEBA DE ESCRITORIO
7. CODIGO JAVA (COMENTARIOS)
8. COMPILACION Y GENERACION DE .JAR
9. VIDEO PUBLICADO EN YOUTUBE CON LA EXPLICACION – MINIMO 7 MINUTOS.
10. PUBLICACION GITHUB / PROYECTO
11. CONCLUSIONES
12. BIBLIOGRAFIA (NORMA APA)



# CONCLUSIONES

.



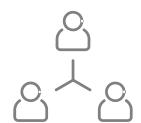


**¿ PREGUNTAS ?**



# RETROALIMENTACIÓN CLASE

**<https://forms.gle/7Ny6jwVKCYvXRqao9>**



# Gracias