

# Desarrollo Web Integrado

Sesión 1

“Introducción. Arquitectura Java”



Universidad  
Tecnológica  
del Perú

Comentemos sobre lo que acontecerá



¿Que veremos durante el desarrollo del curso?

*Sílabo del Curso*

Levanta la mano para participar por audio o chat



Universidad  
Tecnológica  
del Perú

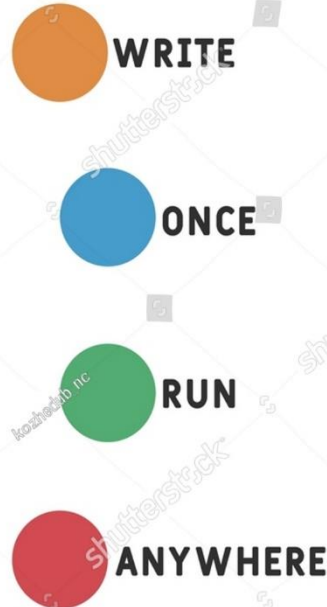
# Inicio



Fuente: Java



Fuente: Shutterstock



¿Qué idea les transmite  
la imagen sobre Java?



Levanta la mano para participar por audio o chat

## Logro de la sesión:



Universidad  
Tecnológica  
del Perú

Al finalizar la **sesión**, el estudiante **identifica** los componentes de la Arquitectura de Java



Desaprende lo que te limita

# Temario de la sesión

- Introducción
- Arquitectura de Java



**Desaprende** lo que te limita

**¿Por qué crees que será importante aprender el tema el día de hoy?**

**Levanta la mano para participar por audio o chat**

## Tema: Introducción y Arquitectura de Java

- Origen
- Evolución
- WORA
- Componentes de Java

# Origen de Java

- **Contexto:** En 1990, Sun Microsystems lanzó un proyecto llamado **Proyecto Green**, dirigido por James Gosling y su equipo.



**James Gosling**  
Patrick Naughton  
Mike Sheridan  
Bill Joy  
Andy Bechtolsheim  
Wayne Rosing

Proyecto encaminado a la realización de un lenguaje de programación orientado a objetos

- **Lenguaje Oak:** El primer lenguaje creado en este proyecto se llamó **Oak**. Se diseñó para dispositivos electrónicos y estaba pensado para ser independiente de la plataforma.
- **Cambio de enfoque:** Aunque Oak no tuvo éxito en dispositivos electrónicos, el equipo se dio cuenta de su potencial para la web emergente.



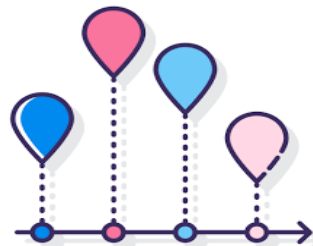
# Evolución de Java

## Lanzamiento de Java (1995):

- **Renombre y Filosofía:** En 1995, Oak se renombró como **Java**. Su filosofía "Write Once, Run Anywhere" (WORA) prometía que el código escrito en Java podría ejecutarse en cualquier sistema que tuviera una Java Virtual Machine (JVM).
- **Applets:** Java se hizo popular rápidamente en la web gracias a los applets, pequeñas aplicaciones incrustadas en páginas web.

## Java 2 (1998):

- **División en Ediciones:** Java 2 introdujo tres ediciones principales:
  - **J2SE (Java 2 Standard Edition):** Para aplicaciones de escritorio y servidor.
  - **J2EE (Java 2 Enterprise Edition):** Para aplicaciones empresariales.
  - **J2ME (Java 2 Micro Edition):** Para dispositivos móviles y embebidos.
- **Impacto en la Industria:** Se convirtió en un estándar para aplicaciones web y empresariales.



## Era de Oracle (2010-presente):

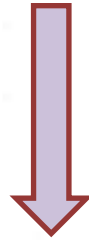
- **Adquisición de Sun Microsystems:** En 2010, Oracle adquirió Sun Microsystems y se hizo cargo de Java.
- **Lanzamientos Rápidos:** Oracle implementó un ciclo de lanzamientos rápidos, con nuevas versiones cada 6 meses, introduciendo características como **expresiones lambda** y el **Streams API**.
- **Java 11 y posteriores:** Java 11 es una versión LTS (Long-Term Support) que ha sido ampliamente adoptada, con mejoras continuas en rendimiento y características.

# WORA

"Write Once, Run Anywhere"

*Escribe una vez, ejecuta  
en cualquier lugar*

Un programa escrito para un sistema operativo como Windows, por ejemplo, necesitaba ser reescrito o significativamente modificado para ejecutarse en otro sistema operativo como macOS o Linux



**Java fue diseñado desde el principio con la idea de resolver este problema de portabilidad**



**Código  
Fuente**



**Código  
Intermedio**



**Código  
Máquina**

*Código independiente  
de la máquina física*

*Código construido para  
una **máquina virtual***

**Máquina  
Virtual**

## Libraries

Bibliotecas de código  
compilado (BYECODE)

## Tools

javac: Compilador

java: Interprete  
entre otros

# Componentes de Java

Design Time

Java

Java Development Kit

Libraries  
Tools

**JDK**

Java Virtual Machine  
**JVM**

Run Time

Java Runtime Environment

**JRE**

Java Virtual Machine  
**JVM**

Código  
Fuente

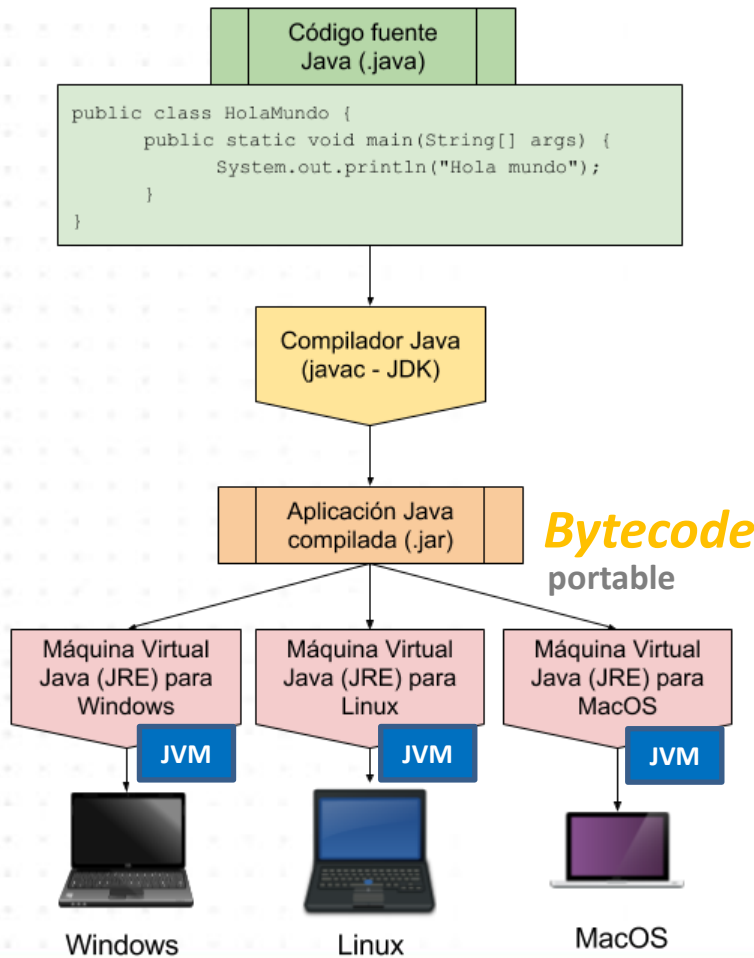
Código  
Intermedio

Código  
Máquina

Código  
Intermedio

Código  
Máquina

# Cómo funciona WORA





***PORTABILIDAD***



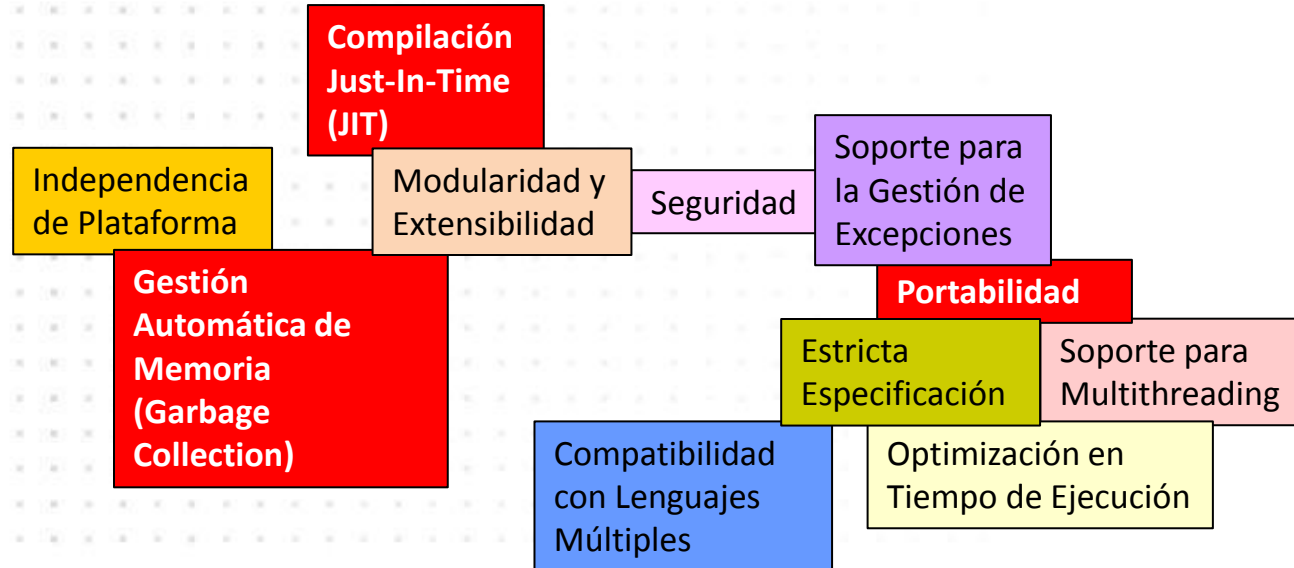
Valoraciones de  
**WORA**

***RENDIMIENTO***



# y la JVM ?

Java Virtual Machine



# Cómo funciona JIT

```
public class JITExample {  
    public static void main(String[] args) {  
        JITExample example = new JITExample();  
        long result = 0;  
  
        // Este bucle ejecuta el método sum en un millón de iteraciones  
        for (int i = 0; i < 1_000_000; i++) {  
            result += example.sum(i, i + 1);  
        }  
  
        System.out.println("Resultado: " + result);  
    }  
  
    public int sum(int a, int b) {  
        return a + b;  
    }  
}
```

Cuando ejecutas este programa, la JVM comienza interpretando el bytecode línea por línea.

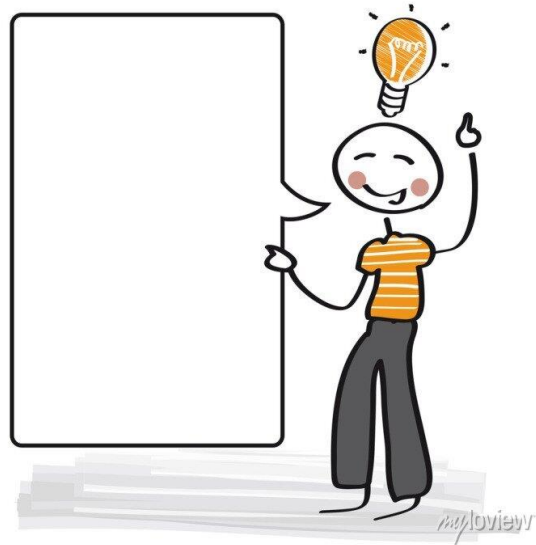
A medida que el bucle continúa ejecutándose, la JVM observa que el método `sum` es llamado repetidamente, un millón de veces en este caso, y lo marca como un **hotspot**

La JVM decide que este método sería un buen candidato para la compilación Just-In-Time y activa el JIT compiler para compilar el método `sum` a código máquina nativo específico para la plataforma en la que se ejecuta el programa.

Durante esta compilación, el JIT puede aplicar optimizaciones como el **inlining**, insertando el código del método `sum` directamente en el lugar donde se llama dentro del bucle, eliminando la sobrecarga de la llamada al método

Una vez que el JIT ha compilado el método, la JVM usa este código nativo optimizado para ejecutar el resto del bucle. Esto significa que las siguientes 999,999 iteraciones del bucle serán mucho más rápidas porque se están ejecutando en código nativo en lugar de ser interpretadas.





# Test

*Portabilidad*

# ¿Porque las aplicaciones de Java son portables?

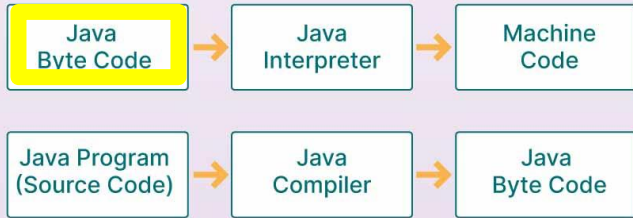


**Veamos ...**

# ¿En que consisten las aplicaciones Java?



## Difference Between Machine Code and Byte Code



*Una aplicación de Java consiste de una  
secuencia de instrucciones ByteCode*

# ByteCode

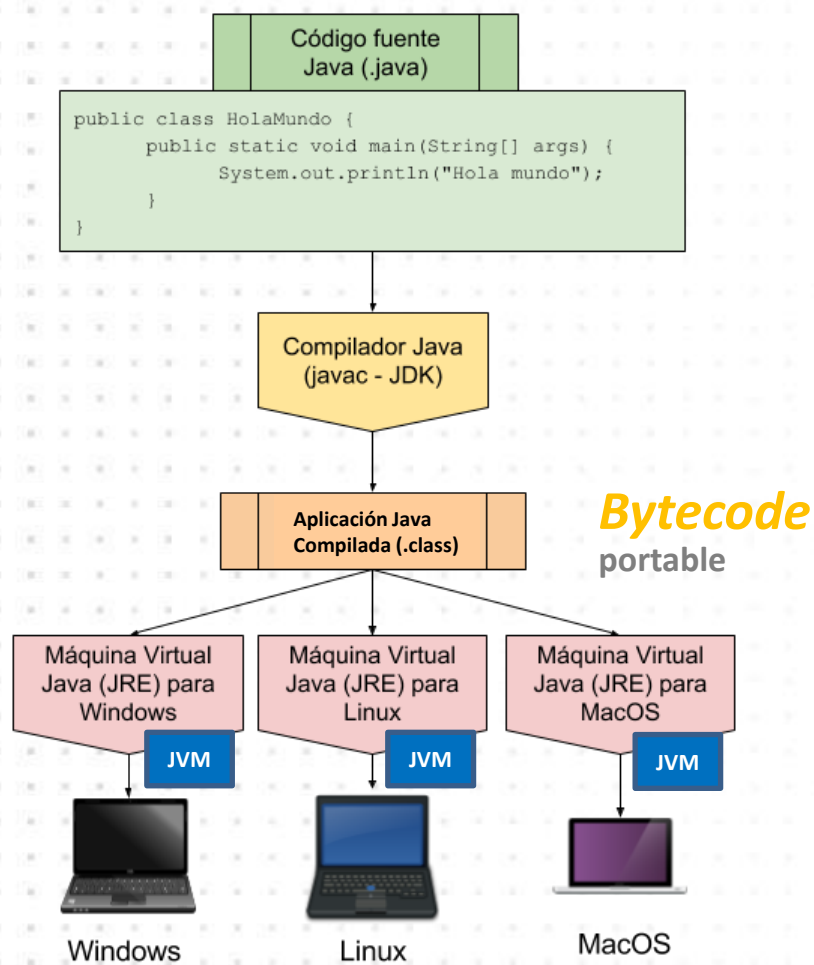
*Código  
intermedio*

*Java: .class*

- Cada código de operación tiene una longitud de un byte
- Cada instrucción tiene un código de operación entre 0 y 255 seguido de parámetros tales como los registros o las direcciones de memoria

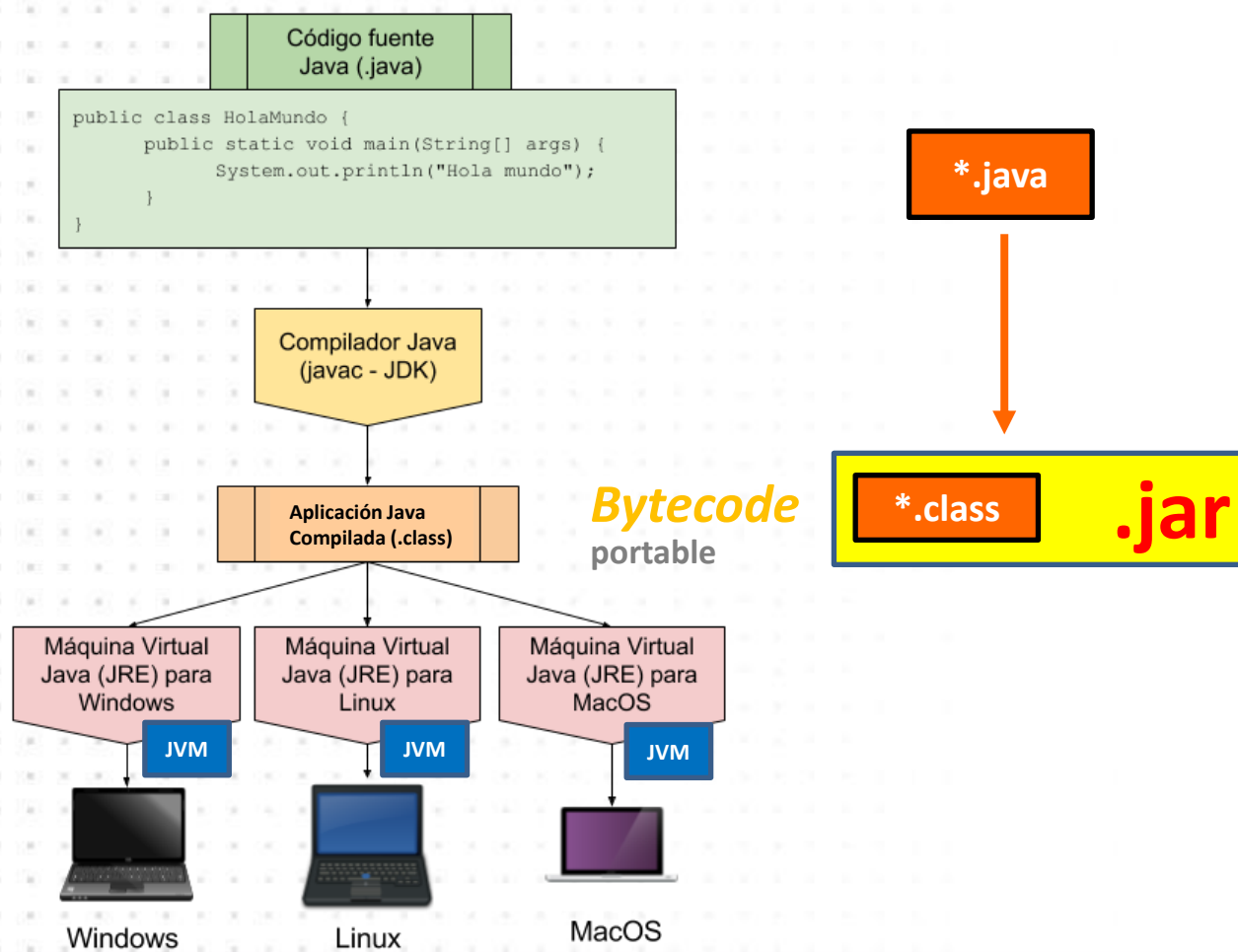
# ¿Cómo se construyen las instrucciones ByteCode para una aplicación Java?





**\*.java**

**\*.class**



MyApp.jar

+-- META-INF/

+-- MANIFEST.MF

+-- com/

+-- example/

+-- app/

+-- Main.class

+-- Utils.class

+-- config/

+-- app.properties

+-- images/

+-- logo.png

# Java **AR**chive



Universidad  
Tecnológica  
del Perú

Un archivo JAR (Java ARchive) es un archivo comprimido que contiene todos los componentes necesarios para ejecutar una aplicación Java

**Clases:** Contienen el código compilado.

**MANIFEST.MF:** Metadatos sobre el JAR, incluyendo el punto de entrada.

**Recursos:** Archivos adicionales necesarios por la aplicación.

**META-INF:** Contiene `MANIFEST.MF` y posibles firmas digitales.

MANIFEST.MF

Manifest-Version: 1.0

Main-Class: com.example.app.Main



## Trabajo en equipo

### Indicaciones

*Generar las instrucciones bytecode para un código fuente escrito en el lenguaje de programación Java*

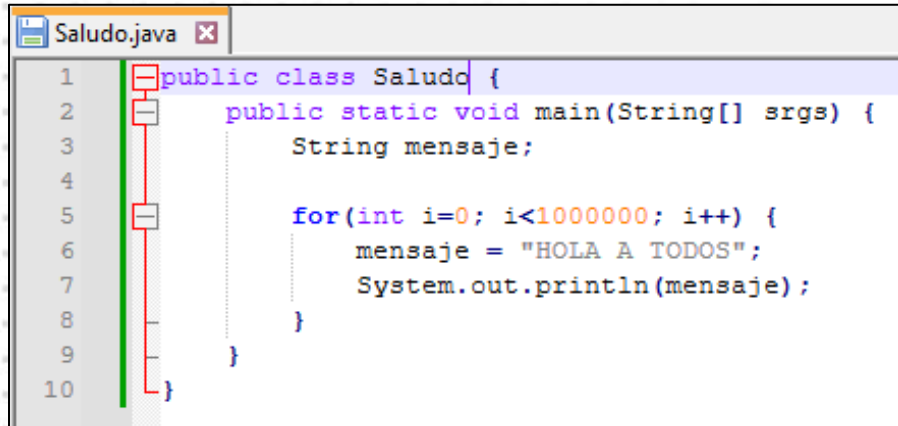


# Practiquemos

Tiempo: 15 minutos

Terminado el tiempo, cada grupo regresa a la sala principal y expone

# ¿Cómo generamos la secuencia de instrucciones ByteCode en un archivo .class?



```
1 public class Saludo {
2     public static void main(String[] srgs) {
3         String mensaje;
4
5         for(int i=0; i<10000000; i++) {
6             mensaje = "HOLA A TODOS";
7             System.out.println(mensaje);
8         }
9     }
10 }
```

javac Saludo.java

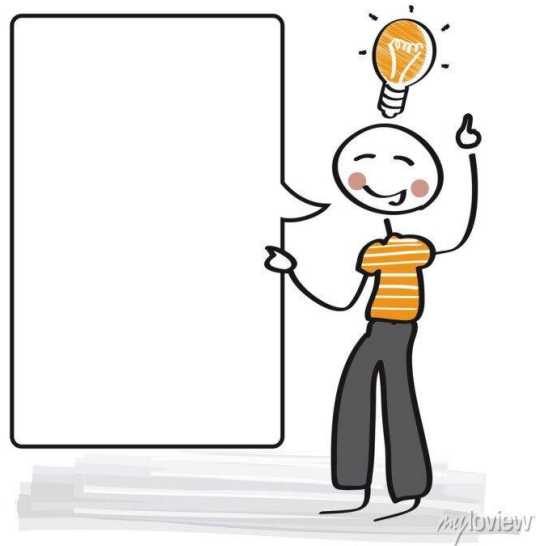


# ¿Cómo generamos la secuencia de instrucciones ByteCode en un archivo .jar?

```
Saludo.java x
1 public class Saludo {
2     public static void main(String[] args) {
3         String mensaje;
4
5         for(int i=0; i<1000000; i++) {
6             mensaje = "HOLA A TODOS";
7             System.out.println(mensaje);
8         }
9     }
10 }
```



```
jar cvmf MANIFIEST.MF Saludo.jar Saludo.class
```



# Test

*Manejo de Memoria*

Archivo: Dato.java

```
public class Dato {  
    private int A;  
    private int B;  
  
    public int getA() {  
        return A;  
    }  
  
    public void setA(int A) {  
        this.A = A;  
    }  
  
    public int getB() {  
        return B;  
    }  
  
    public void setB(int B) {  
        this.B = B;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Dato d1 = new Dato();  
        d1.setA(10);  
        d1.setB(20);  
  
        Dato d2 = new Dato();  
        d2.setA(30);  
        d2.setB(40);  
  
        Dato d3 = new Dato();  
        d3.setA(50);  
        d3.setB(60);  
  
        d1 = d3;  
        d3.setB(45);  
        System.out.println(Integer.toString(d2.getA() + d1.getB()));  
    }  
}
```



Universidad  
Tecnológica  
del Perú



# ¿Qué valor imprime?

Veamos ...

# ¿Cómo organiza Java la memoria?



principalmente

...

# STACK and HEAP

1. Llamadas a funciones  
(Call functions -> Call Stack)
2. Variables de tipo básico
3. Variables de tipo referencia  
(Referencias a objetos)

1. Objetos



# ¿Cómo maneja Java la memoria?

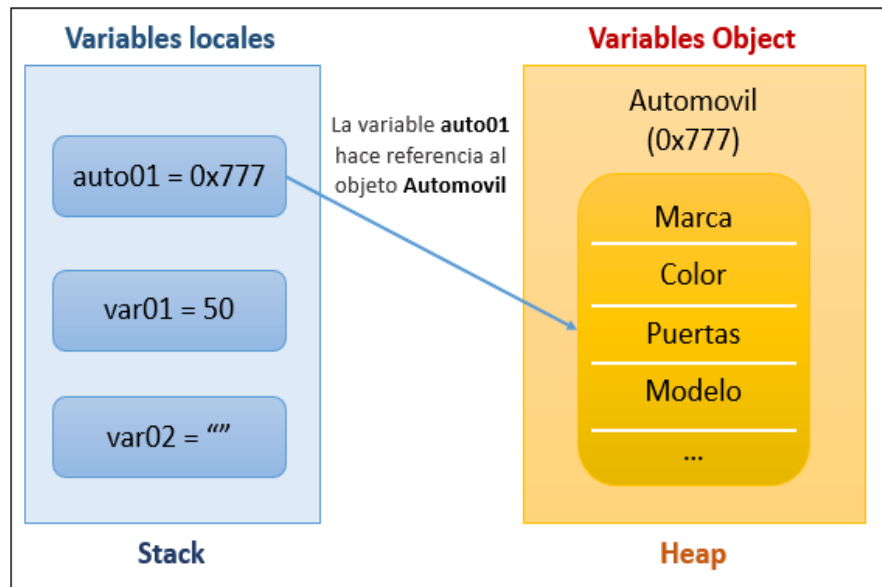
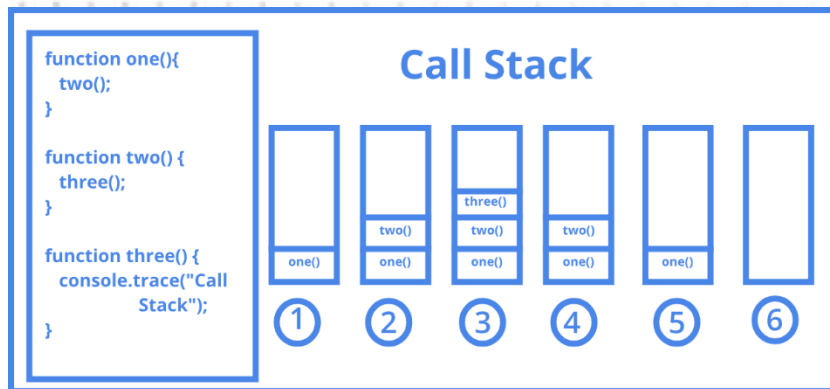




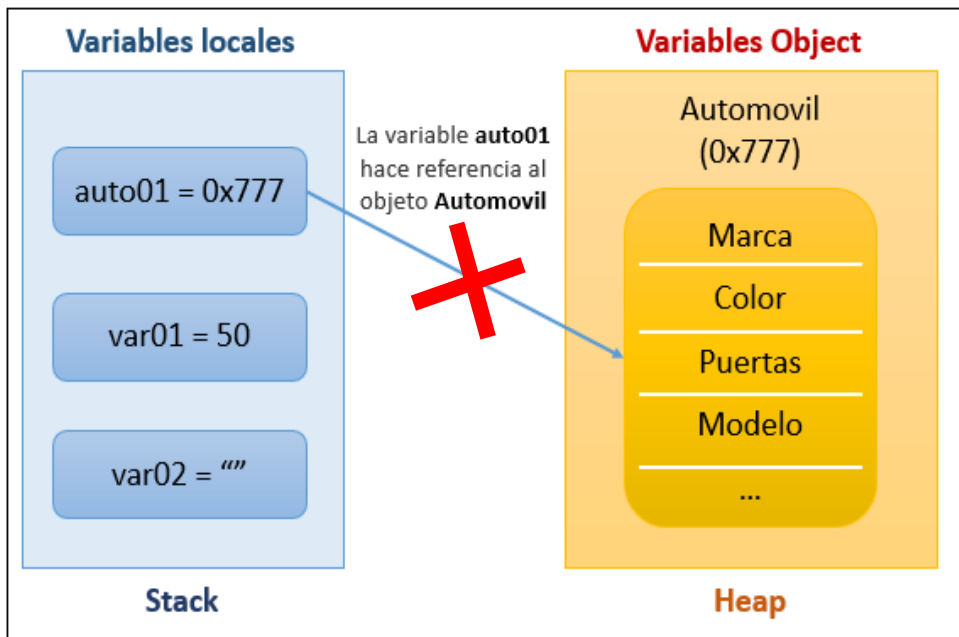
# Variables

```
Automovil auto01 = new Automovil();  
int var01 = 50;  
String var02 = "";
```

## Funciones



```
Automovil auto01 = new Automovil();  
int var01 = 50;  
String var02 = "";
```



**auto01 = null;**

***Objeto no  
referenciado***

# ¿Cómo visualizamos la memoria consumida?



# VisualVM

VisualVM 2.1.9

File Applications View Tools Window Help

40,3/78,5MB

Applications ×

- Local
  - VisualVM
    - Saludo (pid 8780)
      - [heapdump] 06:32:19 AM
  - Remote
    - VM Coredumps
    - JFR Snapshots
    - Snapshots

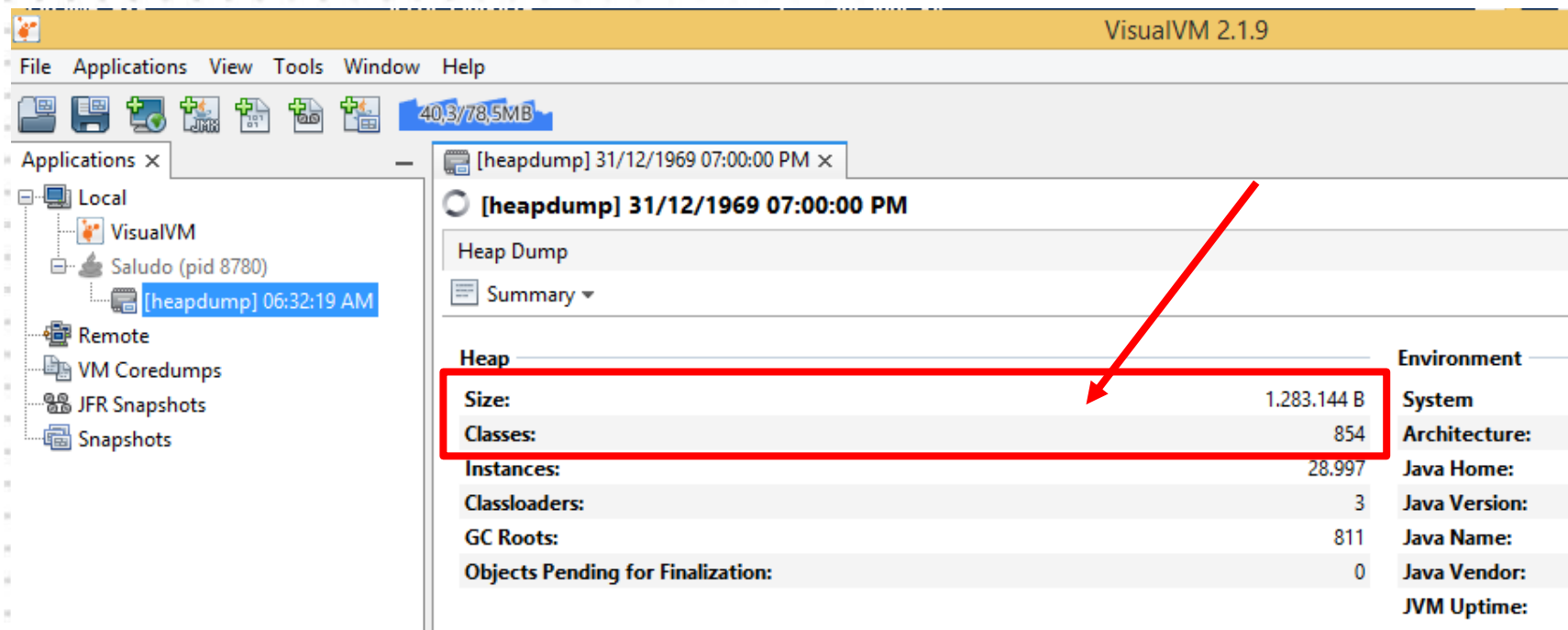
[heapdump] 31/12/1969 07:00:00 PM ×

**[heapdump] 31/12/1969 07:00:00 PM**

Heap Dump

Summary ▾

Heap	Environment
Size: 1.283.144 B	System
Classes: 854	Architecture:
Instances: 28.997	Java Home:
Classloaders: 3	Java Version:
GC Roots: 811	Java Name:
Objects Pending for Finalization: 0	Java Vendor:
	JVM Uptime:



# ¿Cómo libera Java la memoria consumida por los objetos?



Libera memoria ocupada por  
objetos que ya no son accesibles o  
necesarios, evitando así fugas de  
memoria y mejorando la eficiencia



```
auto01 = null;
```

***Objeto no  
referenciado***

**GC**

**Garbage Collector**

*Sistema de gestión  
automática de memoria*

***OutOfMemoryError***

## Indicaciones

*Simular el consumo de memoria de una aplicación Java*



# Practiquemos

Tiempo: 15 minutos

Terminado el tiempo, cada grupo regresa a la sala principal y expone

### Archivo: Dato.java

```
public class Dato {  
    private int A;  
    private int B;  
  
    public int getA() {  
        return A;  
    }  
  
    public void setA(int A) {  
        this.A = A;  
    }  
  
    public int getB() {  
        return B;  
    }  
  
    public void setB(int B) {  
        this.B = B;  
    }  
}
```

### Archivo: Test.java

```
public class Test {  
    public static void main(String[] args) {  
        Dato d1 = new Dato();  
        d1.setA(10);  
        d1.setB(20);  
  
        Dato d2 = new Dato();  
        d2.setA(30);  
        d2.setB(40);  
  
        Dato d3 = new Dato();  
        d3.setA(50);  
        d3.setB(60);  
  
        d1 = d2;  
        System.out.println(Integer.toString(d1.getA() + d1.getB()));  
  
        d3 = d1;  
        d2 = d3;  
        System.out.println(Integer.toString(d1.getA() + d2.getA()));  
  
        d1 = d3;  
        d3.setB(60);  
        System.out.println(Integer.toString(d2.getA() + d1.getB()));  
  
        d1.setA(200);  
        System.out.println(Integer.toString(d3.getA() + d3.getB()));  
    }  
}
```

1

2

3

4



Universidad  
Tecnológica  
del Perú

¿Qué valor  
imprime en  
la línea  
1, 2, 3 y 4?





# ¿Qué cantidad de memoria consume del heap?

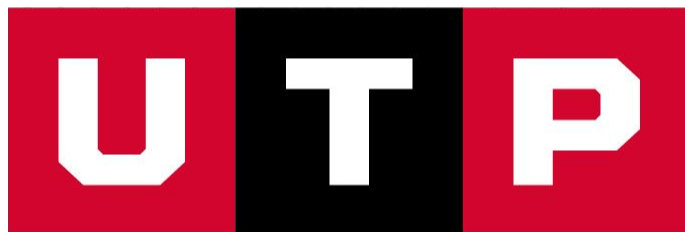
```
Saludo.java x
1 public class Saludo {
2     public static void main(String[] args) {
3         String mensaje;
4
5         for(int i=0; i<1000000; i++) {
6             mensaje = "HOLA A TODOS";
7             System.out.println(mensaje);
8         }
9     }
10 }
```



## ¿Qué aprendiste el día de hoy?

- ¿En que consiste la filosofía WORA?
- ¿Cuáles son los componentes de la plataforma Java?
- ¿Qué es el bytecode?
- ¿Qué es la JVM?
- ¿Cómo funciona la compilación JIT?
- ¿Que almacena Java en la memoria HEAP?

Levanta la mano para participar por audio o chat



**Universidad  
Tecnológica  
del Perú**