CICLO 2

[FORMACIÓN POR CICLOS]

Programación Básica JAVA

JAVA

Ingeni@ Soluciones TIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Contactos

- Soporte: soportemisiontic@udea.edu.co
- Equipo de Permanencia: permanenciamisiontic@udea.edu.co
- Docente: gomezmunera@gmail.com

Formador	John Anderson Gómez Munera	gomezmunera@gmail.com
Tutor	Ánderson Barrientos Parra	anderson.barrientos@udea.edu.co

Recursos

- Moodle

https://lms.misiontic2022udea.com/

http://pythontutor.com/visualize.html#mode=edit

Contenido

- Programa del curso
- Jerarquía de datos
- Tipos de lenguajes
- Generalidades de JAVA
- JDK, JRE y JVM
- Entorno de desarrollo: Netbeans
- Tecnologías de software

Contenido

- Configuración del IDE
- Estructura de un programa estándar en JAVA
- Impresión de línea de texto
- EJERCICIOS

- Variables
- Prefijos y posfijos
- Casting de datos con *parse*
- Import y Entradas de texto
- EJERCICIOS

Contenido

- Comparación de datos numéricos
- Comparación de cadenas de caracteres
- Ciclo condicional if
- Ciclo for
- While
- Ciclo Do While
- EJERCICIOS

- Vectores
- Matrices
- Llenar vector con ciclos
- Llenar matriz con ciclos
- EJERCICIOS

Programa

Semana	Núcleos temáticos	Horas de los encuentros sincrónicos	Horas de trabajo independiente	Total
Semana 1	Introducción al lenguaje Java y su sintaxis	3	7	10
	Sentencias de control de flujo	2	5	7
	Vectores y matrices	2	5	7

Programa

Semana 2	Introducción a la programación orientada a objetos	2	5	7
	Clases, objetos, métodos y atributos	2	5	7
	Diagramas de clases en UML 2.5	3	7	10

Programa

Semana 3	Herencia e interfaces	2	5	7
	Polimorfismo	2	5	7
	Colecciones en Java	3	7	10

Programa

Semana 4	Introducción a las interfaces gráficas en Java	3	7	10
	Más de interfaces gráficas en Java usando Swing	4	10	14

Programa

Semana 5	Introducción a las pruebas unitarias con Junit	4	10	14
	Introducción a la persistencia mediante bases de datos	3	7	10

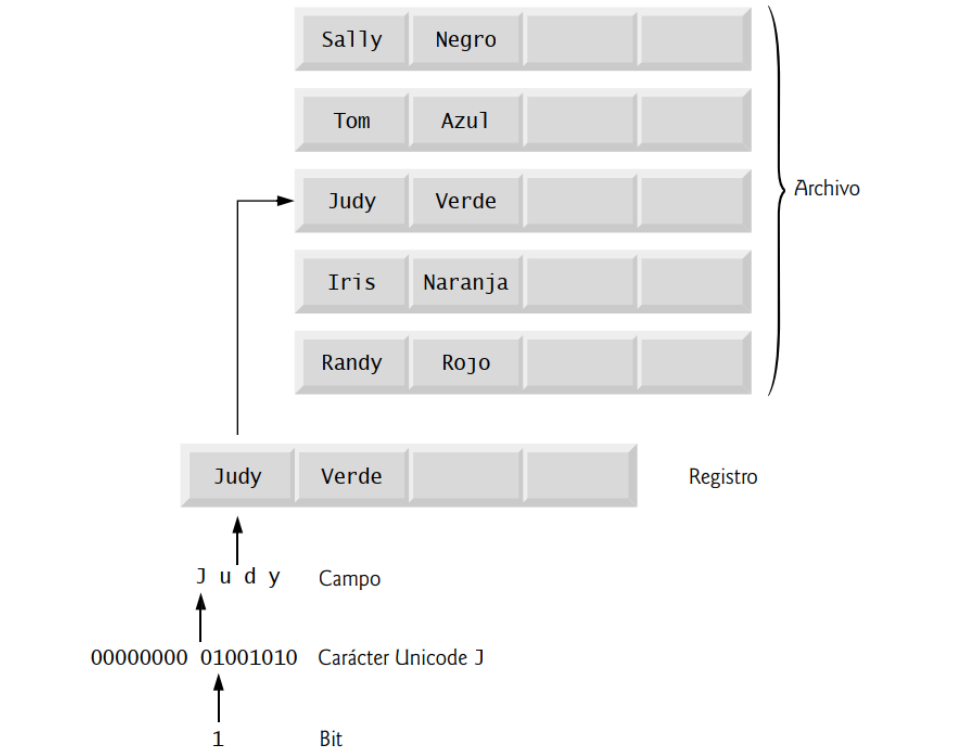
Programa

Semana 6	Diseño básico de bases de datos	2	5	7
	Bases de datos relacionales (con SQL)	5	12	19

Programa

Semana 7	Más de bases de datos relacionales	2	5	7
	Conexión a bases de datos con JDBC (I)	3	7	10
	Introducción a patrones multi-capa usando Modelo-Vista-Controlador (MVC)	2	5	7

Jerarquía de Datos



Jerarquía de datos.

Unidades de Almacenamiento

Unidades de Medidas de Almacenamiento

Medida	Simbologia	Equivalencia	Equivalente en Bytes
byte	b	8 bits	1 byte
kilobyte	Kb	1024 bytes	1 024 bytes
megabyte	MB	1024 KB	1 048 576 bytes
gigabyte	GB	1024 MB	1 073 741 824 bytes
terabyte	TB	1024 GB	1 099 511 627 776 bytes
Petabyte	PB	1024 TB	1 125 899 906 842 624 bytes
Exabyte	EB	1024 PB	1 152 921 504 606 846 976 bytes
Zetabyte	ZB	1024 EB	1 180 591 620 717 411 303 424 bytes
Yottabyte	YB	1024 ZB	1 208 925 819 614 629 174 706 176 bytes
Brontobyte	BB	1024 YB	1 237 940 039 285 380 274 899 124 224 bytes
Geopbyte	GB	1024 BB	1 267 650 600 228 229 401 496 703 205 376 bytes

Lenguaje de alto nivel, ensamblador y de máquina

```
sueldoBruto = sueldoBase + sueldoExtra
```

```
load        sueldobase
add         sueldoextra
store       sueldobruto
```

```
+1300042774
+1400593419
+1200274027
```

Generalidades de Java

Sun Microsystems patrocinó en 1991 un proyecto interno de investigación corporativa dirigido por James Gosling, que dio como resultado un lenguaje de programación orientado a objetos y ***basado en C++,*** al que Sun llamó **Java**.

Un objetivo clave de Java es poder escribir programas que se ejecuten en una gran variedad de sistemas computacionales y dispositivos controlados por computadora. A esto se le conoce algunas veces como “escribir una vez, ejecutar en cualquier parte”

Generalidades de Java

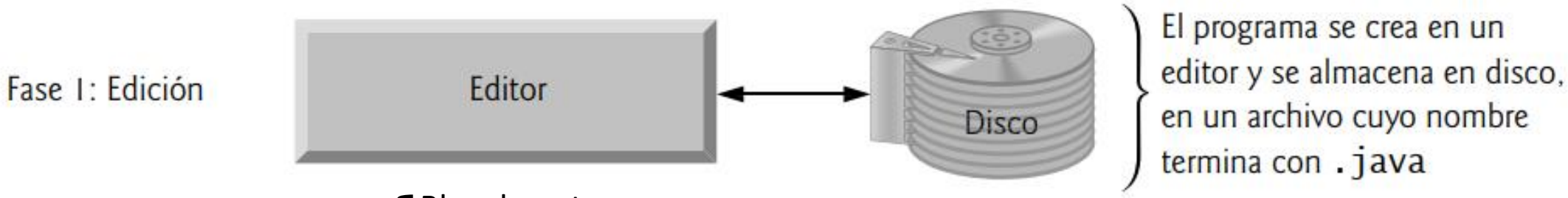
En 1993 Sun vio el potencial de usar Java para agregar contenido dinámico, como interactividad y animaciones, a las **páginas Web**. Java atrajo la atención de la comunidad de negocios debido al fenomenal interés en el servicio Web.

En la actualidad, Java se utiliza para desarrollar aplicaciones empresariales a gran escala, para mejorar la funcionalidad de los servidores Web, para proporcionar aplicaciones para los dispositivos de uso doméstico (celulares, televisores) y para muchos otros propósitos.

Java también es el lenguaje clave para desarrollar aplicaciones para teléfonos inteligentes y tabletas de Android. En 2010, Oracle adquirió Sun Microsystems.

Típico entorno de desarrollo en Java

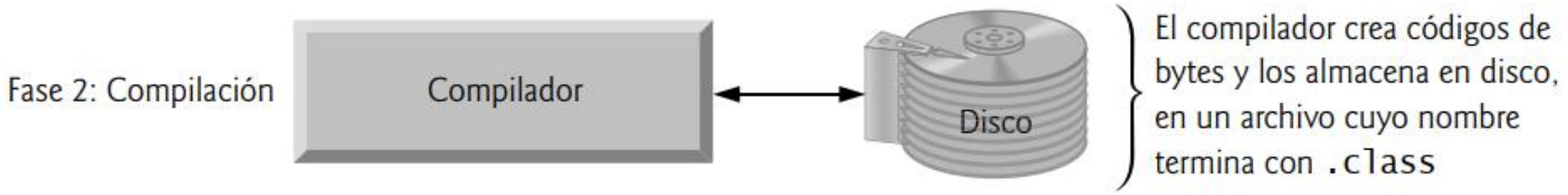
Los pasos típicos utilizados para crear y ejecutar una aplicación en Java. Por lo general hay cinco fases: edición, compilación, carga, verificación y ejecución.



- Básicos
  - Bloc de notas
  - Emacs
  - Notepad++
- Entornos de desarrollo
  - Netbeans
  - Eclipse
  - IntelliJ IDEA

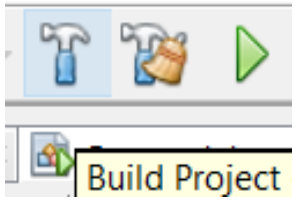
Típico entorno de desarrollo en Java

Los pasos típicos utilizados para crear y ejecutar una aplicación en Java. Por lo general hay cinco fases: edición, compilación, carga, verificación y ejecución.



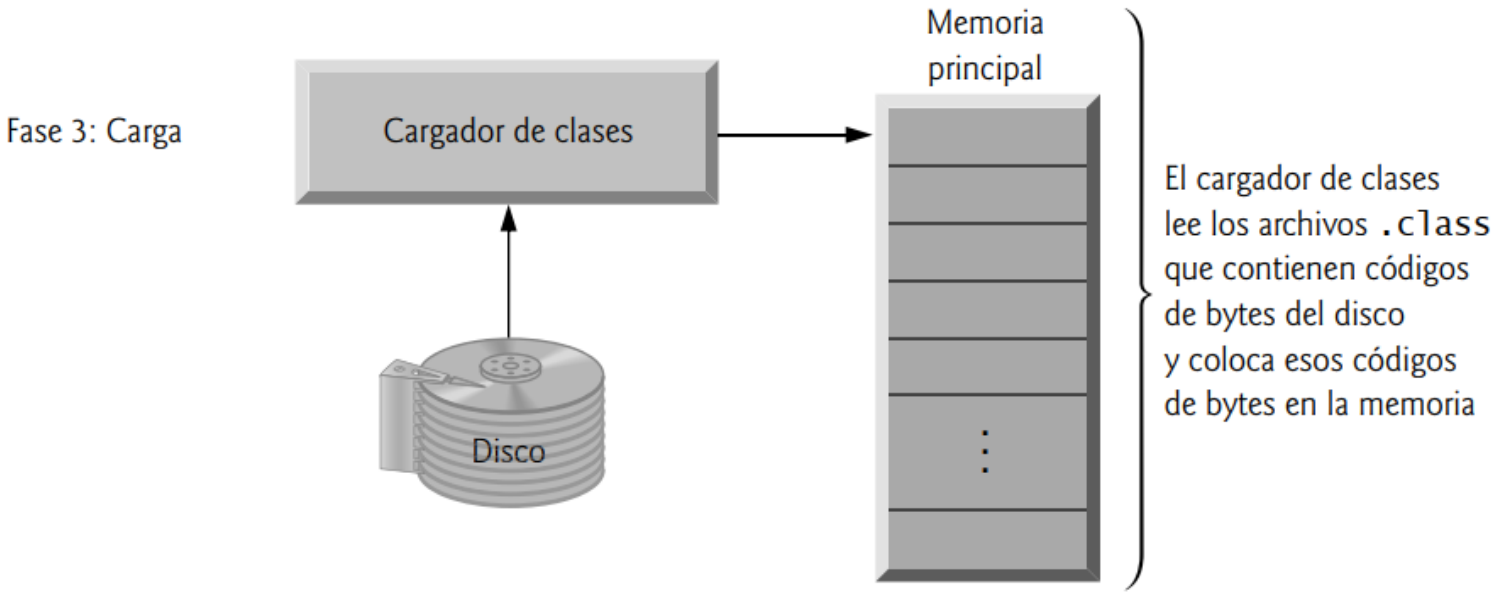
Editor básico { javac programa.java

Entornos de desarrollo {



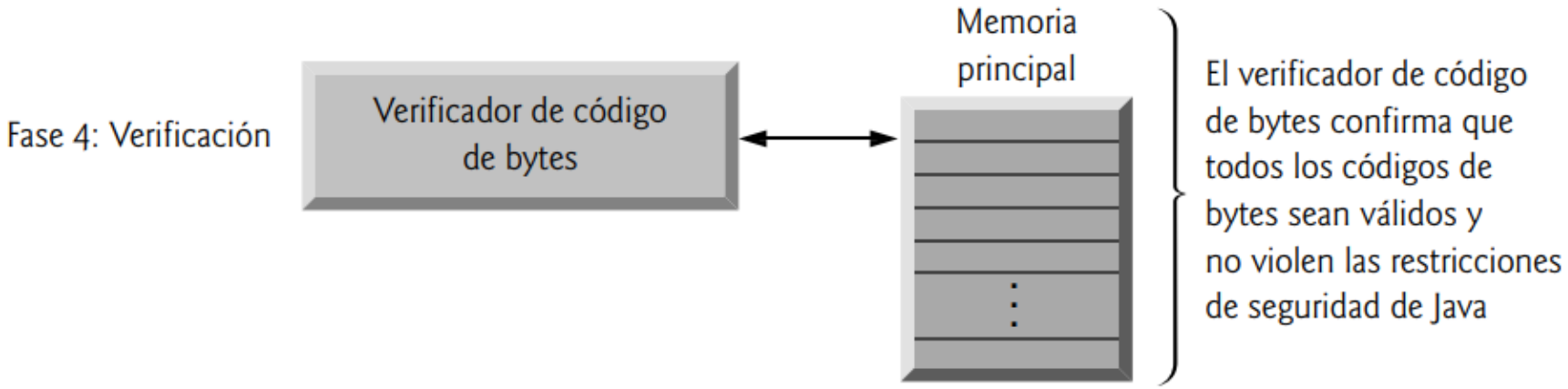
Típico entorno de desarrollo en Java

La JVM coloca el programa en memoria para ejecutarlo; a esto se le conoce como carga. El cargador de clases también carga cualquiera de los archivos .class que su programa utilice, y que sean proporcionados por Java desde un disco en su sistema o a través de una red



Típico entorno de desarrollo en Java

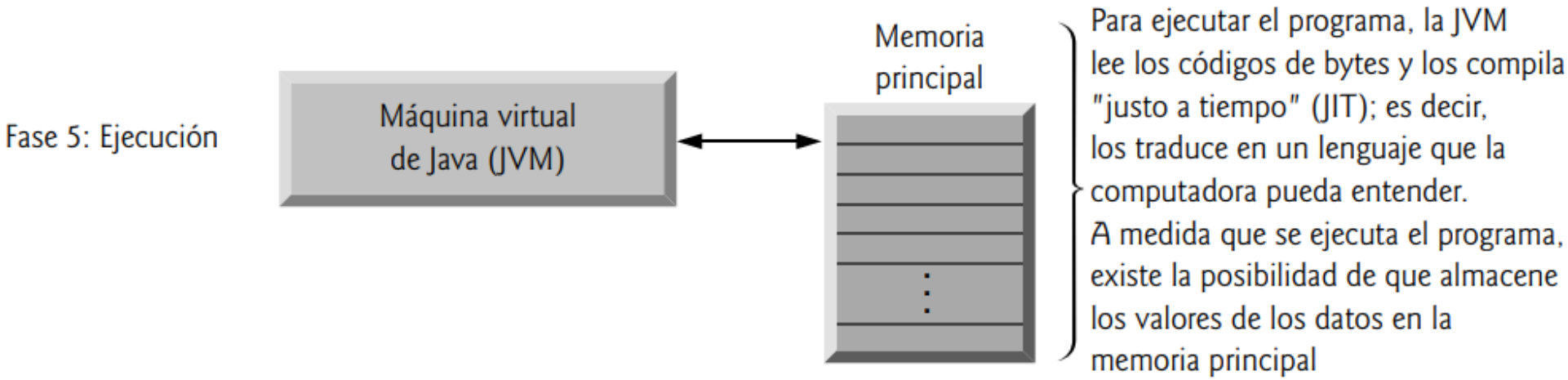
A medida que se cargan las clases, el ***verificador de códigos*** de bytes examina sus códigos de bytes para asegurar que sean válidos y que no violen las restricciones de seguridad de Java.



Java implementa una estrecha seguridad para asegurar que los programas en Java que llegan a través de la red no dañen sus archivos o su sistema

Típico entorno de desarrollo en Java

La JVM ejecuta los códigos de bytes del programa, realizando así las acciones especificadas por el mismo .



En las primeras versiones de Java, la JVM era tan sólo un intérprete de códigos de bytes de Java. Esto hacía que la mayoría de los programas se ejecutaran con lentitud, ya que la JVM tenía que interpretar y ejecutar un código de bytes a la vez.

Típico entorno de desarrollo en Java

Por lo general, las JVM actuales ejecutan códigos de bytes mediante una combinación de la interpretación y la denominada compilación ***justo a tiempo (JIT).***

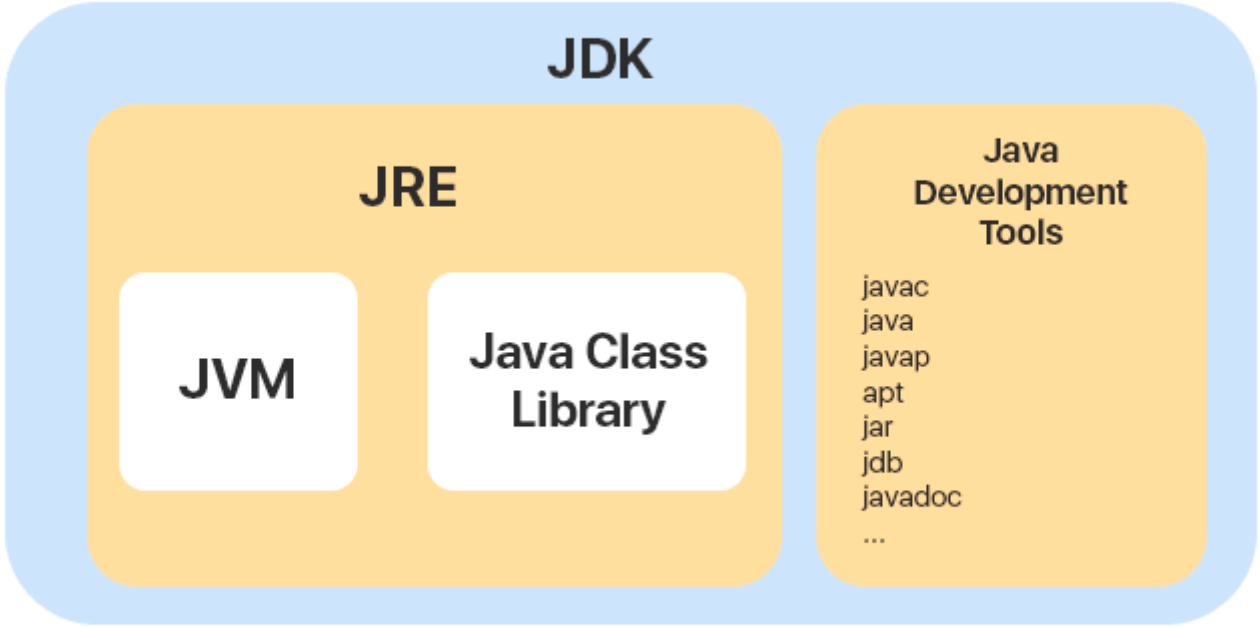
En este proceso, la JVM analiza los códigos de bytes a medida que se interpretan, en busca de puntos activos (***partes de los códigos de bytes que se ejecutan con frecuencia***). Para estas partes, un compilador justo a tiempo (JIT) traduce los códigos de bytes al lenguaje máquina correspondiente de la computadora.

Cuando la JVM ***vuelve*** a encontrar estas partes compiladas, ***se ejecuta el código en lenguaje máquina, que es más rápido***.

Por ende, los programas en Java en realidad ***pasan por dos fases de compilación***: una en la cual el código fuente se traduce a ***código de bytes*** (portabilidad) y otra en la que, durante la ejecución los ***códigos de bytes se traducen en lenguaje máquina*** para la computadora actual en la que se ejecuta el programa.

JAVA:

JDK, JRE, JVM



JDK: Java Development Kit

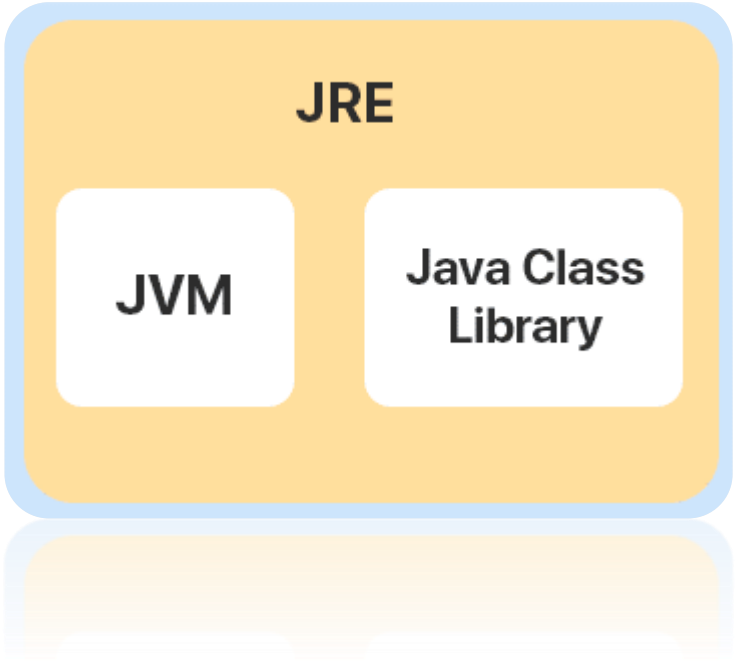
Es el kit de desarrollo para JAVA, es un conjunto de herramientas que permiten a los programadores crear programas en lenguaje JAVA

Los programas más importantes que se incluyen:

appletviewer.exe:	es un visor de applets para generar sus vistas previas, ya que un applet carece de método main y no se puede ejecutar con el programa java.
javac.exe:	es el compilador de Java.
java.exe:	es el intérprete de Java.
javadoc.exe:	genera la documentación de las clases Java de un programa

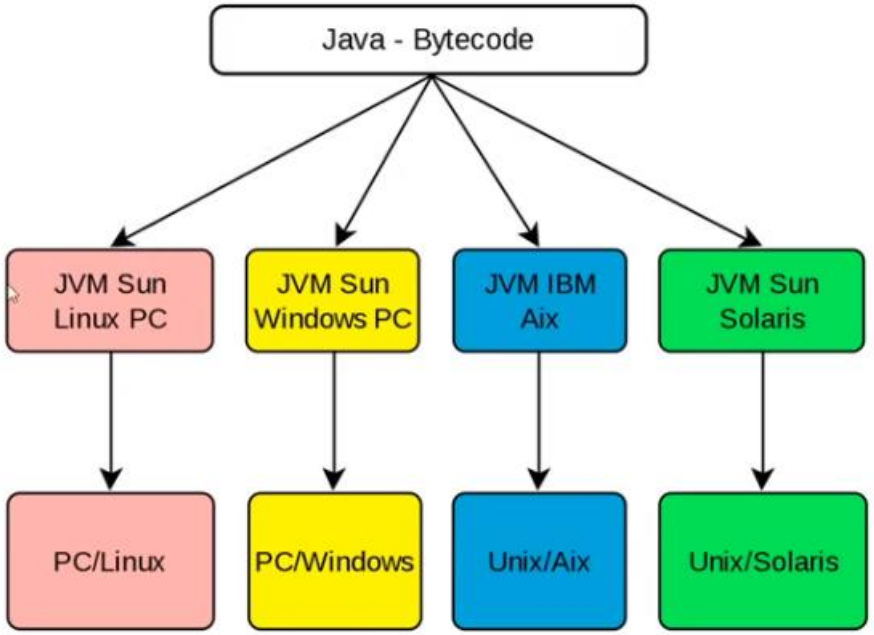
JRE: Java Runtime Environment

El JRE (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java.



JVM: Java Virtual Machine

Una máquina virtual Java (en inglés Java Virtual Machine, JVM) es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.



Tecnologías de Software

- Desarrollo ágil de software
- Refactorización
- Patrones de diseño
- LAMP
- SaaS
- PaaS
- Iaas
- Computación en la nube
- SDK (Kits de desarrollo de software)

Versiones

- Alfa
- Beta
- Candidatos para liberación
- Liberación de versión final
- Beta permanente

Quizizz

https://quizizz.com/join?gc=00656070

Parte 2

Estructura básica de un programa en Java

```
1  // Fig. 2.1: Bienvenido1.java
2  // Programa para imprimir texto.
3
4  public class Bienvenido1
5  {
6     // el método main empieza la ejecución de la aplicación en Java
7     public static void main(String[] args)
8     {
9        System.out.println("Bienvenido a la programacion en Java!");
10    } // fin del método main
11 } // fin de la clase Bienvenido1
```

Estructura básica de un programa en Java

Los **paréntesis** después del identificador main indican que éste es un bloque de construcción del programa, al cual se le llama **método**.

Las declaraciones de clases en Java por lo general contienen uno o más métodos.

En una aplicación en Java, **sólo uno de esos métodos *debe* llamarse main** y hay que definirlo como se muestra en la línea 7; de no ser así, la Máquina Virtual de Java (JVM) no ejecutará la aplicación.

Los métodos pueden realizar tareas y devolver información una vez que éstas hayan concluido. Mas adelante hablaremos sobre static.

La palabra clave **void** indica que este método ***no*** devolverá ningún tipo de información. Más adelante veremos cómo puede un método devolver información.

En la línea 7, las palabras String[] args entre paréntesis son una parte requerida de la declaración del método main; hablaremos sobre esto mas adelante.

Variables y tipos de datos

Una **variable** es un espacio en memoria, especificado a la hora de escribir un programa en cualquier lenguaje de programación, que tiene un nombre, y que puede tener una información conocida o desconocida, denominada valor.

En Java, además, una variable tiene un tipo, que determina el tipo de información que puede incluir, el rango de valores que puede adoptar y el tamaño que la variable ocupa en la memoria principal.

Variables

En **JAVA** para usar una variable, es necesario primero declararla, es decir, asignar un tipo y un nombre de variable para asignar un espacio en memoria.

```
int NOMBRE_VARIABLE;
```

Java case_sentitive

Java es **sensible a mayúsculas y minúsculas**; es decir, las letras mayusculas y minúsculas son distintas, por lo que **valor, VALOR** y **Valor** son distintos identificadores (pero todos son validos).

Consideraciones:

- El nombre de una variable puede ser un identificador alfanumérico que empiece por una letra, el signo $ o guion bajo (_). **No se permiten espacios en blanco**.
- Se recomiendan palabras auto-descriptivas que no sean una palabra reservada del lenguaje (que se pueden conocer en la referencia de Java).
- Para el nombrado de variables, se recomienda la notación camel-case, es decir:

  Si el nombre consta de una sola palabra, dicha palabra se escribe toda en minúsculas, si son dos o más palabras, se escribe la primera palabra toda en minúsculas, y las palabras subsecuentes, en minúscula, pero con la primera letra en mayúscula, así:

  frecuencia, telefono, direccionResidencia, cambioDireccion.

Datos Primitivos

Tipo	Tamaño en bits	Valores	Estándar
`boolean`		`true` o `false`	
[*Nota:* una representación `boolean` es específica para la Máquina virtual de Java en cada plataforma].			
`char`	16	`'\u0000'` a `'\uFFFF'` (0 a 65535)	(ISO, conjunto de caracteres Unicode)
`byte`	8	–128 a +127 (-2^7 a $2^7 - 1$)	
`short`	16	–32,768 a +32,767 (-2^{15} a $2^{15} - 1$)	
`int`	32	–2,147,483,648 a +2,147,483,647 (-2^{31} a $2^{31} - 1$)	
`long`	64	–9,223,372,036,854,775,808 a +9,223,372,036,854,775,807 (-2^{63} a $2^{63} - 1$)	
`float`	32	*Rango negativo:* –3.4028234663852886E+38 a –1.40129846432481707e–45 *Rango positivo:* 1.40129846432481707e–45 a 3.4028234663852886E+38	(IEEE 754, punto flotante)
`double`	64	*Rango negativo:* –1.7976931348623157E+308 a –4.94065645841246544e–324 *Rango positivo:* 4.94065645841246544e–324 a 1.7976931348623157E+308	(IEEE 754, punto flotante)

Tipado

Java usa tipado fuerte y estático:

- Fuerte: No se pueden mezclar tipos de datos
  - Int a = 2;
  - String b = “0”;
  - c = a + b; // PRODUCE UN ERROR

- Estático: Es igual durante toda la ejecución del programa
  - Int a = 2 ; // no puede cambiar después a String

Método Print

Quizizz

https://quizizz.com/join?gc=17171142

Post - pre operadores

Operador	Uso	Descripción
++	op++	Incrementa op en 1; se evalúa al valor anterior al incremento (post incremento)
++	++op	Incrementa op en 1; se evalúa al valor posterior al incremento (pre incremento)
--	op--	Decrementa op en 1; se evalúa al valor anterior al incremento (post decremento)
--	--op	Decrementa op en 1; se evalúa al valor posterior al incremento (pre decremento)

Casting y parse

El **casting** consiste en la conversión de tipos de datos similares (compatibles) entre sí, generalmente a través de la herencia.

```
int a= (int)(5.65);
```

El **parsing** consiste en analizar el formato de una sentencia de texto, y obtener información si el formato es correcto

```
//parse: analiza y convierte si es posibles sino devuelve error
int b = Integer.parseInt("2500");
System.out.println(b);

// si no son compatible produce in error:
int c = Integer.parseInt("hola");
System.out.println(b);
```

Palabras reservadas

Palabras clave en Java				
abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	final	finally	float	for
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	transient	try
void	volatile	while		
Palabras clave que no se utilizan actualmente				
const	goto			

Operadores Lógicos

Operador	Descripción	Asociatividad
++	unario de postincremento	de derecha a izquierda
--	unario de postdecremento	
++	unario de preincremento	de derecha a izquierda
--	unario de predecremento	
+	unario de suma	
-	unario de resta	
!	unario de negación lógica	
~	unario de complemento a nivel de bits	
(*tipo*)	unario de conversión	
*	multiplicación	de izquierda a derecha
/	división	
%	residuo	

Operadores Lógicos

+	suma o concatenación de cadenas	de izquierda a derecha
-	resta	
<<	desplazamiento a la izquierda	de izquierda a derecha
>>	desplazamiento a la derecha con signo	
>>>	desplazamiento a la derecha sin signo	
<	menor que	de izquierda a derecha
<=	menor o igual que	
>	mayor que	
>=	mayor o igual que	
instanceof	comparación de tipos	
==	es igual que	de izquierda a derecha
!=	no es igual que	
&	AND a nivel de bits AND lógico booleano	de izquierda a derecha
^	OR excluyente a nivel de bits OR excluyente lógico booleano	de izquierda a derecha

Operadores Lógicos

Operador	Descripción	Asociatividad
|	OR incluyente a nivel de bits OR incluyente lógico booleano	de izquierda a derecha
&&	AND condicional	de izquierda a derecha
||	OR condicional	de izquierda a derecha
?:	condicional	de derecha a izquierda
=	asignación	de derecha a izquierda
+=	asignación, suma	
-=	asignación, resta	
*=	asignación, multiplicación	
/=	asignación, división	
%=	asignación, residuo	
&=	asignación, AND a nivel de bits	
^=	asignación, OR excluyente a nivel de bits	
|=	asignación, OR incluyente a nivel de bits	
<<=	asignación, desplazamiento a la izquierda a nivel de bits	
>>=	asignación, desplazamiento a la derecha a nivel de bits con signo	
>>>=	asignación, desplazamiento a la derecha a nivel de bits sin signo	

Operadores Lógicos

OPERADOR	NOMBRE	EJEMPLO	DEVUELVE VERDADERO CUANDO...
&&	y	(7 > 2) && (2 < 4)	las dos condiciones son verdaderas
||	o	(7 > 2) || (2 < 4)	al menos una de las condiciones es verdadera
!	no	!(7 > 2)	la condición es falsa

Orden de los operadores

Operador(es)	Operación(es)	Orden de evaluación (precedencia)
* / %	Multiplicación División Residuo	Se evalúan primero. Si hay varios operadores de este tipo, se evalúan de *izquierda a derecha*.
+ –	Suma Resta	Se evalúan después. Si hay varios operadores de este tipo, se evalúan de *izquierda a derecha*.
=	Asignación	Se evalúa al último.

Control de Flujo en Java

- Condicionales: if y switch
- Ciclo for
- Ciclo While
- Ciclo Do While

- Palabras clave:
  - Break
  - Continue

Condicionales: if y switch

```
if (condición) {

    instrucciones a ejecutar si la condición es verdadera

} else {

    instrucciones a ejecutar si la condición es falsa

}
```

```
public class ComparacionCadena {
  public static void main(String[] args) {
    String miFruta = "naranja";

    if ("naranja".equals(miFruta)) {
      System.out.println("iguales");
    } else {
      System.out.println("distintas");
    }
```

Condicionales: if y switch

```
switch(variable) {
  case valor1:
    sentencias
    break;

  case valor2:
    sentencias
    break;
    .
    .
    .

  default:
    sentencias
    }
```

```
String nombreDelMes;

switch (mes) {
  case 1:
  nombreDelMes = "enero";
  break;
case 2:
  nombreDelMes = "febrero";
  break;
        |
        |
default:
  nombreDelMes = "no existe ese mes";
}
```

Ciclo: for

```
for (expresion1 ; expresion2 ; expresion3) {

  sentencias

}
```

```
public class EjemploFor {
  public static void main(String[] args) {
    for (int i = 1; i < 11; i++) {
      System.out.println(i);
    }
  }
}
```

Ciclo: While

```
while (expresion) {

  sentencias

}
```

```
int i = 1;

while (i < 11) {
  System.out.println(i);
  i++;
}
```

Ciclo: Do While

```
do {

  sentencias

} while (expresion)
```

```
int i = 1;

do {
  System.out.println(i);
  i++;
} while (i < 11);
```

Arreglos de Datos

Arreglos de una dimensión

Matrices

Arreglos de mas dimensiones

Arreglos de una dimensión

```
int[] n; // se define n como un array de enteros
n = new int[4]; // se reserva espacio para 4 enteros

n[0] = 26;
n[1] = -30;
n[2] = 0;
n[3] = 100;


int[] x = {8, 33, 200, 150, 11};
```

Matrices

```
int[][] n = new int[3][2]; // array de 3 filas por 2 columnas

n[0][0]=20;
n[1][0]=67;
n[1][1]=33;
n[2][1]=7;
```

```
String[][] matriz ={{"Juan","Gómez"},{"David", "Gutierrez"}};
```

Arreglos de n-Dimensiones

```java
// Arreglo de n dimensiones

int nDim[][][][];
nDim = new int[2][2][2][2];
nDim[0][1][0][1]=1;
System.out.println(nDim[0][1][0][1]);

int[][][] nDim2={{{1,2},{4,5}},{{10,20},{40,50}}};

System.out.println(nDim2[0][1][1]);
System.out.println(nDim2[1][1][1]);
```

For al estilo foreach

```java
double[] nota = new double[4];

double suma = 0;

for (double n : nota) { // for al estilo foreach
  System.out.print(n + "  ");
  suma += n;
}
```

Este estilo no me deja acceder a los índices

Ciclos y condicionales

```
1  if ( CONDITION )
2  {
3    STATEMENTS
4  }
5  AFTER_IF_PART
```

Ciclos y condicionales

```
1  if ( CONDITION )
2  {
3    STATEMENTS1
4  }
5  else
6  {
7    STATEMENTS2
8  }
9  AFTER_IF_ELSE_PART
```

Condicionales con casos especiales

```
1  if ( CONDITION1 ) { STATEMENTS1 }
2  else if ( CONDITION2 ) { STATEMENTS2 }
3  else if ( CONDITION3 ) { STATEMENTS3 }
4  else { STATEMENTS4 }
```

```
1  if ( CONDITION1 ) { STATEMENTS1 }
2  else {
3    if ( CONDITION2 ) { STATEMENTS2 }
4    else { if ( CONDITION3 ) { STATEMENTS3 }
5      else { STATEMENTS4 }
6    }
7  }
```