***Universidad Autónoma de Tlaxcala***


# Introducción al lenguaje JAVA

M.C. José Juan Hernández Mora



# Primera Sesión

1. Introducción a la programación orientada a objeto
2. Introducción al lenguaje Java
3. Primer programa en Java
4. Primer applet en Java
5. Literales en Java
6. Tipos de datos básicos de Java
7. Variables en Java
8. Operadores en Java
   - Operadores aritméticos
   - Operadores relacionales
   - Operadores lógicos
   - Operadores de bits
   - Asignación
   - Precedencia de operadores
9. Condicionales
   - sentencia ***if***
   - Sentencia ***switch***
10. Ciclos en Java
    - Sentencia ***while***
    - Sentencia ***do...while***
    - Sentencia ***for***

# Segunda Sesión

1. Arreglos
2. Matrices
3. Clases en Java
4. Clases de Usuario en Java
5. Objetos definidos por el usuario
6. La sentencia ***static***
7. Clases de la librería de Java
8. La clase ***Math***
9. La clase ***String***



# Tercera Sesión

1. Applets en Java
   - Estructura de un applet
2. Gráficos en Java
   - El paquete ***awt***
   - Líneas
   - Rectángulos
   - Óvalos
   - Arcos
   - Polígonos
3. Colores en Java
4. Animaciones sencillas
5. Eventos del Ratón
6. Eventos del Teclado

# Cuarta sesión

1. Imágenes en Java
2. Animaciones sencillas con imágenes en Java
3. Interfaz de usuario en Java
   - Etiquetas
   - Botones
   - Casillas
   - Grupos de casillas
   - Listas
   - Menú
   - Áreas de Texto
   - Gestores de Organización
4. Imágenes en Java
5. Entornos de desarrollo



# Clases

Una clase es una agrupación de ***datos*** (variables o campos) y de ***funciones*** (métodos) que operan sobre esos datos.

## La definición de una clase se realiza en la siguiente forma:

[public] **class** Classname {
// definición de variables y métodos
...
}



## Objetos

- Un ***objeto*** (en inglés, ***instance***) es un ejemplar concreto de una clase.
- Las ***clases*** son como tipos de variables, mientras que los ***objetos*** son como variables concretas de un tipo determinado.

## *Encapsulación*.

Las clases pueden ser declaradas como públicas (***public***) y como ***package*** (accesibles sólo para otras clases del ***package***).

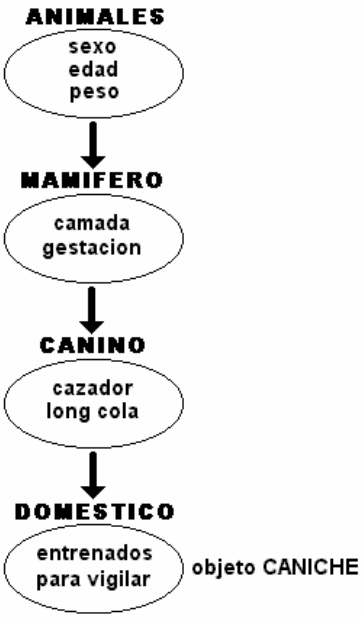
Las variables miembro y los métodos pueden ser ***public***, ***private***, ***protected*** y ***package***. De esta forma se puede controlar el acceso y evitar un uso inadecuado.



## *Herencia*.

Una clase puede derivar de otra (***extends***), y en ese caso hereda todas sus variables y métodos.

Una clase derivada puede ***añadir*** nuevas variables y métodos y/o ***redefinir*** las variables y métodos heredados.




# *Polimorfismo*.

Los objetos de distintas clases pertenecientes a una misma jerarquía o que implementan una misma interface pueden tratarse de una forma general e individualizada, al mismo tiempo.

Esto, facilita la programación y el mantenimiento del código.

# Inventores de Java

Diseñado por James Gosling en 1990 (Sun Microsystems)

Java para internet  Bill Joy  1995



# INTRODUCCIÓN A JAVA

***Java*** surgió en 1991 cuando un grupo de ingenieros de *Sun Microsystems* trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos.

# INTRODUCCIÓN A JAVA

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada.



# INTRODUCCIÓN A JAVA

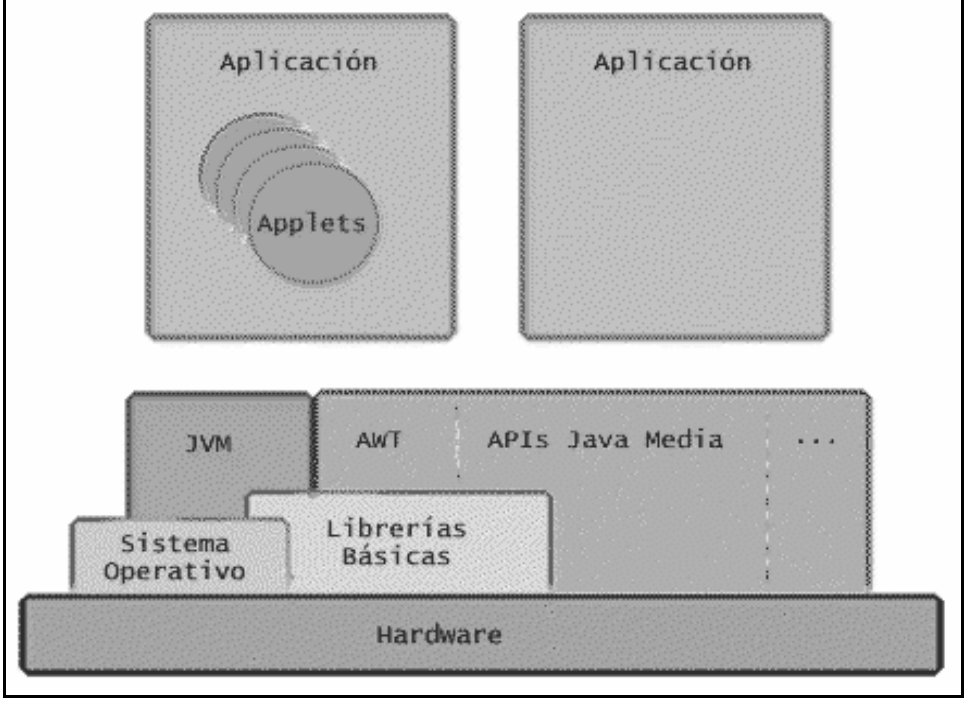
Desarrollaron un código "neutro" que no dependía del tipo de electrodoméstico, el cual se ejecutaba sobre una "*máquina hipotética o virtual*" denominada ***Java Virtual Machine*** (***JVM***).

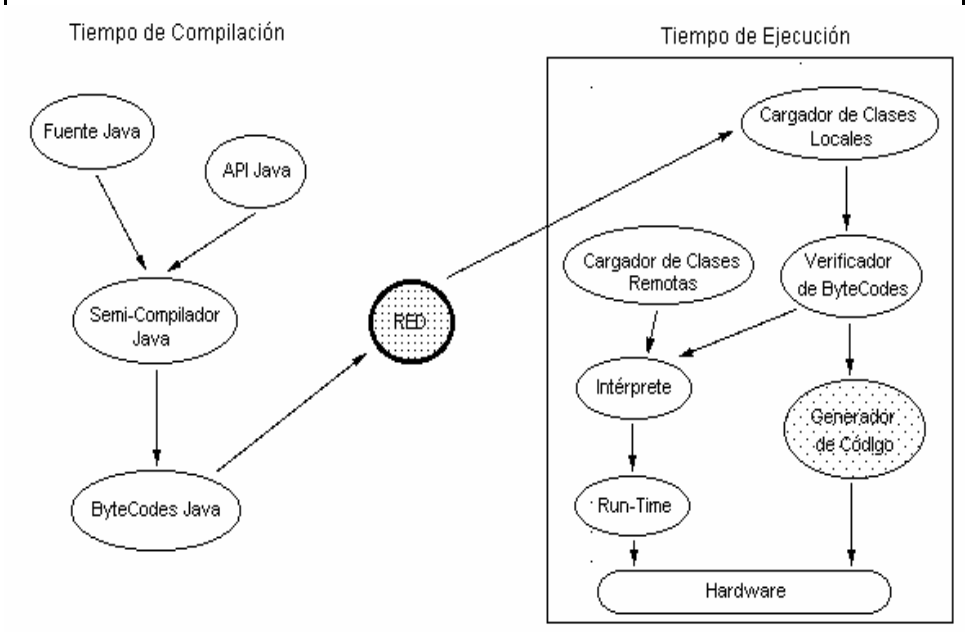
# INTRODUCCIÓN A JAVA

Esto permitía lo que luego se ha convertido en el principal lema del lenguaje:

"*Write Once, Run Everywhere*".








# Primer programa

```
public class HolaMundoApp
{
 public static void main( String args[] )
   {
       System.out.println( "Hola Mundo!" );
    }
 }
```

# Compilación

- **javac.exe**
- **javac holamundo.java**



# Ejecución

- **java.exe**
- **java holamundo**

# Primer Applet

```
import java.awt.Graphics;
import java.applet.Applet;
public class HolaMundo extends Applet {
     public void paint( Graphics g ) {
    // Pinta el mensaje en la posición indicada
        g.drawString( "Hola Mundo!",25,25 );
     }
}
```



# **Llamada a Applets**

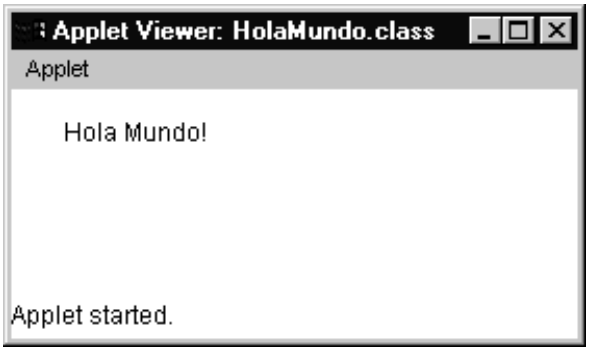
## Archivo *.html

```
<APPLET CODE="Codigo.class" WIDTH=100 HEIGHT=50>

</APPLET>
```

# Ejecución applet

- **appletviewer.exe**






# Introducción al Lenjuage JAVA

# Comentarios:

```
// comentarios para una sola línea

 /*
 comentarios de una o más líneas
*/
```



# Identificadores

Los identificadores nombran variables, funciones, clases y objetos; cualquier cosa que el programador necesite identificar o usar.

- En Java, un identificador comienza con una letra, un subrayado (_) o un símbolo de dólar ($). Los siguientes caracteres pueden ser letras o dígitos. Se distinguen las mayúsculas de las minúsculas y no hay una longitud máxima establecida para el identificador

## Serían identificadores válidos:

- Identificador
- nombre_usuario
- Nombre_Usuario
- _variable_del_sistema
- $transaccion



## *Palabras Clave*

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

# Tipos de datos

| Tipo de variable | Descripción |
|---|---|
| Boolean | 1 byte. Valores true y false |
| Char | 2 bytes. Unicode. Comprende el código ASCII |
| Byte | 1 byte. Valor entero entre -128 y 127 |
| Short | 2 bytes. Valor entero entre -32768 y 32767 |
| Int | 4 bytes. Valor entero entre -2.147.483.648 y 2.147.483.647 |
| Long | 8 bytes. Valor entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807 |
| Float | 4 bytes (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38 |
| Double | 8 bytes (unas 15 cifras decimales equivalentes). De -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308 |



# Tipos de datos

El tipo ***boolean*** no es un valor numérico: sólo admite los valores ***true*** o ***false***.

El tipo ***char*** contiene caracteres en código UNICODE (que incluye el código ASCII), y ocupan 16 bits por carácter. Comprende los caracteres de prácticamente todos los idiomas.

Los tipos ***byte***, ***short***, ***int*** y ***long*** son números enteros que pueden ser positivos o negativos, con distintos valores máximos y mínimos. A diferencia de C/C++, en ***Java*** no hay enteros ***unsigned***.

# Tipos de datos

Los tipos ***float*** y ***double*** son valores de punto flotante (números reales) con 6-7 y 15 cifras.

Se utiliza la palabra ***void*** para indicar la ausencia de un tipo de variable determinado.



# Tipos de datos

Una variable se define especificando el ***tipo*** y el ***nombre*** de dicha variable.

Estas variables pueden ser tanto de tipos ***primitivos*** como ***referencias*** a objetos de alguna clase perteneciente al ***API*** de ***Java*** o generada por el usuario.

# OPERADORES DE JAVA

M.C. José Juan Hernández Mora

## Operadores aritméticos

- ***suma*** (+)
- ***resta*** (-)
- ***multiplicación*** (*)
- ***división*** (/)
- ***resto de la división*** (%).

M.C. José Juan Hernández Mora

# Operadores de asignación

El operador de asignación por excelencia es el ***operador igual*** (=).

La forma general de las sentencias de asignación con este operador es:

variable = expresión;



# Operadores de asignación

| Operador | Utilización | Expresión equivalente |
|---|---|---|
| += | op1 += op2 | op1 = op1 + op2 |
| -= | op1 -= op2 | op1 = op1 - op2 |
| *= | op1 *= op2 | op1 = op1 * op2 |
| /= | op1 /= op2 | op1 = op1 / op2 |
| %= | op1 %= op2 | op1 = op1 % op2 |

## Operadores unarios

Los operadores ***más*** (+) y ***menos*** (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica.



## Operadores incrementales

***Java*** dispone del operador ***incremento*** (++) y ***decremento*** (--). El operador (++) incrementa en una unidad la variable a la que se aplica, mientras que (--) la reduce en una unidad.

# Operadores relacionales

| Operador | Utilización | El resultado es true |
|---|---|---|
| > | op1 > op2 | si op1 es mayor que op2 |
| >= | op1 >= op2 | si op1 es mayor o igual que op2 |
| < | op1 < op2 | si op1 es menor que op2 |
| <= | op1 <= op2 | si op1 es menor o igual que op2 |
| == | op1 == op2 | si op1 y op2 son iguales |
| != | op1 != op2 | si op1 y op2 son diferentes |



# Operadores lógicos

| Operador | Nombre | Utilización | Resultado |
|---|---|---|---|
| && | AND | op1 && op2 | true si op1 y op2 son true. Si op1 es false ya no se evalúa op2 |
| \|\| | OR | op1 \|\| op2 | true si op1 u op2 son true. Si op1 es true ya no se evalúa op2 |
| ! | negación | ! op | true si op es false y false si op es true |
| & | AND | op1 & op2 | true si op1 y op2 son true. Siempre se evalúa op2 |
| \| | OR | op1 \| op2 | true si op1 u op2 son true. Siempre se evalúa op2 |

# Operador de concatenación de cadenas de caracteres (+)

El operador más (+) se utiliza también para concatenar cadenas de caracteres. Por ejemplo, para escribir una cantidad con un rótulo y unas unidades puede utilizarse la sentencia:

```
System.out.println ("El total asciende a " + result + " unidades");
```



# Operadores que actúan a nivel de bits

| Operador | Utilización | Resultado |
|---|---|---|
| >> | op1 >> op2 | Desplaza los bits de op1 a la derecha una distancia op2 |
| << | op1 << op2 | Desplaza los bits de op1 a la izquierda una distancia op2 |
| >>> | op1 >>> op2 | Desplaza los bits de op1 a la derecha una distancia op2 (positiva) |
| & | op1 & op2 | Operador AND a nivel de bits |
| \| | op1 \| op2 | Operador OR a nivel de bits |
| ^ | op1 ^ op2 | Operador XOR a nivel de bits (1 si sólo uno de los operandos es 1) |
| ~ | ~op2 | Operador complemento (invierte el valor de cada bit) |

# Precedencia de operadores

```
[] . (params) expr++ expr--
++expr --expr +expr -expr ~ !
new (type)expr
* / %
+ -
<< >> >>>
< > <= >= instanceof
== !=
&
^
|
&&
||
? :
= += -= *= /= %= &= ^= |= <<= >>= >>>=
```



# Ciclos

# o

# birfucaciones

## *Bifurcación if*

```
if (boolean_Expresión) {
   sentencia;
}
```



## *Bifurcación if else*

```
if (boolean_Expresión) {
   statements1;
} else {
   statements2;
}
```

## *Bifurcación if elseif else*

```
if (boolean_Expresión1) {
    statements1;
} else if (boolean_Expresión2) {
    statements2;
} else if (boolean_Expresión3) {
    statements3;
} else {
    statements4;
}
```



## ejemplo

- Programa mayor y menor "maymen.java"

# break

- La sentencia ***break*** es válida tanto para las bifurcaciones como para los bucles.
- Hace que se salga inmediatamente del bucle o bloque que se está ejecutando, sin sin realizar la ejecución del resto de las sentencias.



# *Sentencia switch*

```
switch (expression) {
   case value1: statements1; break;
   case value2: statements2; break;
   case value3: statements3; break;
   case value4: statements4; break;
   case value5: statements5; break;
   case value6: statements6; break;
[default: statements7;]
}
```

# Ejemplo

- Programa “cinco.java”



# **Bucles o**
# Ciclos

- Un ***bucle*** se utiliza para realizar un proceso repetidas veces. Se denomina también ***lazo*** o ***loop***.

## *Bucle while*

```
while (booleanExpression) {
   statements;
}
```



## Ejemplo

Programa del factorial

“facw.java”

# *Bucle for*

```
for (initialization; booleanExpression; increment)
{
  statements;
}
```



- La sentencia o sentencias ***initialization*** se ejecuta al comienzo del ***for***
- ***increment*** después de ***statements***.
- ***booleanExpression*** se evalúa al comienzo de cada iteración
- El bucle termina cuando la expresión de comparación toma el valor ***false***.

# for

- Cualquiera de las tres partes puede estar vacía.
- La ***initialization*** y el ***increment*** pueden tener varias expresiones separadas por comas.



# Ejemplo

- Programa factorial “facf.java”

## for en while

```
initialization;
while (booleanExpression) {
   statements;
increment;
}
```



## *Bucle do while*

```
do {
   statements
} while (booleanExpression);
```

# Ejemplo

- Programa factorial “facd.java”



# *continue*

La sentencia ***continue*** se utiliza en los bucles (no en bifurcaciones).

Finaliza la iteración “i” que en ese momento se está ejecutando (no ejecuta el resto de sentencias que hubiera hasta el final del bucle).

# Programas

- Programa el mayor, menor y medio de tres números
- Programa de los romanos del 1 al 3999
- Programa del cambio de 100 pesos



# Tarea

- Programa de los cheques
- Programa de Fibonacci
- Programa de los primos

# Fin por Hoy

M.C. José Juan Hernández Mora