

# PROGRAMACIÓN EN JAVA

---

## TEMA 1: Introducción a la programación en JAVA



## Bibliografía

---

- “JAVA 2 Lenguaje y aplicaciones”  
Fco. Javier Ceballos. Ed. Ra-Ma, 2006  
Capítulos: 1 - 7

# Contenido

---

- **Características de JAVA**
- **¿Que es JAVA?**
- **Realización de un programa en JAVA**
- **Programación en JAVA**
- **Arrays**
- **Operadores**
- **Separadores**
- **Sentencias de control de flujo**
- **Paquetes**
- **Paquetes de JAVA**
- **Clases de JAVA**
- **Tipos de clases**
- **Variables y métodos de instancia**
- **Alcance de objetos y reciclado de memoria**
- **Herencia**
- **Control de acceso**
- **Variables y métodos estáticos**
- **this y super**
- **Clases abstractas**
- **Interfaces**
- **Referencias en JAVA**
- **Una mínima aplicación en JAVA**
- **Compilación y ejecución**
- **Problemas de compilación**

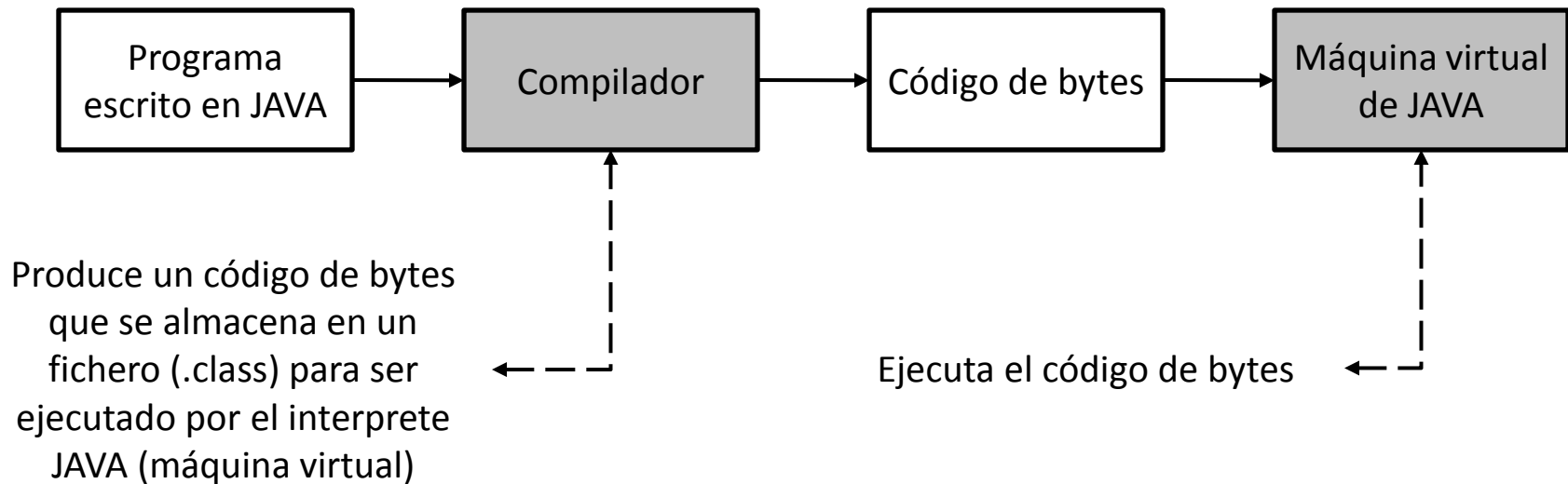
# Características de JAVA

---

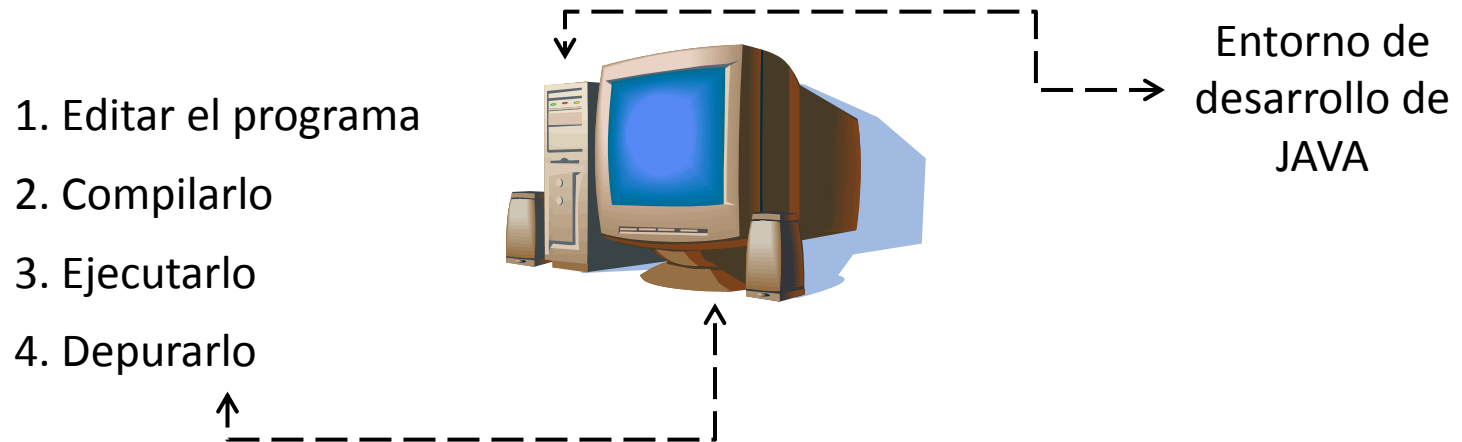
- **Simple** → Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas
- **OO** → Incorpora conceptos como encapsulación, herencia y polimorfismo y por supuesto, trabaja con objetos y clases
- **Distribuido** → Proporciona librerías y herramientas para que los programas se puedan correr en varias máquinas
- **Robusto** → Realiza verificaciones en busca de problemas tanto en tiempo de compilación como de ejecución
- **Portable** → Porque es de arquitectura independiente
- **Interpretado** → Se ejecuta directamente el código objeto
- **Multithreaded** → Permite muchas actividades simultaneas en un programa
- **Seguro** → Evita el acceso ilegal a la memoria. El código pasa por muchos test antes de ser ejecutado

# ¿Que es JAVA?

- Es un lenguaje de programación de alto nivel introducido por *Sun Microsystems*, con el que se pueden escribir tanto programas convencionales como para Internet.
- Incluye 2 elementos: un *compilador* y un *intérprete*



# Realización de un programa en JAVA



# Programación en JAVA

---

- Tipos de programas
  - **Stand Alone:** Aplicación independiente
  - **Applets:** Programas que se ejecutan en el navegador del cliente
  - **Servlets:** Programa sin interfaz que se ejecuta en el servidor
- ¿Por qué programar en JAVA?
  - Independencia de la plataforma (portabilidad)
  - Es orientado a objetos
  - Es un lenguaje fácil de aprender. (fundamentado en C++)
  - Facilidades: No existen punteros, las cadenas de caracteres son objetos y la administración de memoria es automática

# Programación en JAVA (cont.)

---

- **Comentarios:** en una línea (//) o varias líneas (/\* \*/)
- **Identificadores:**
  - Nombre de una variable, función, clase u objeto
  - Comienzan con una letra, subrayado (\_) o dólar (\$)
  - No hay longitud máxima
  - Hay distinción entre mayúsculas y minúsculas
- **Tipos de datos:** enteros, reales en como flotante, booleanos, caracteres y cadenas
- **Enteros:**
  - byte (8 bits)
  - short (16 bits)
  - int (32 bits – 4 bytes)
  - long (64 bits – 8 bytes)



# Programación en JAVA (cont.)

---

- **Reales en coma flotante:**

- float (32 bits)
- double (64 bits)

- **Booleanos:**

- true
- false

- **Caracteres:**

- Ejemplo: a\t

- **Cadenas:**

- Ejemplo: “Programación en JAVA”

# Arrays

---

- Se pueden declarar arrays de cualquier tipo:
  - `char s[];`
  - `int arreglo[];`
  - `int tabla[][] = new int [4][5];`
- Se puede usar el método *length* para conocer la longitud de cualquier array
  - `int a[][] = new int [10][3];`
  - `a.length; /*10*/`
  - `a[0].length; /*3*/`
- Creación de arrays:
  - `int lista[] = new int [50];`
  - `String nombres[] = {"Juan","Pepe","Maria"};`
  - Error: `int lista [50];`

# Operadores

---

- Son parecidos a los de C. Por orden de precedencia:

```
.    []    ()  
++  --  
!    ~    instanceof  
*    /    %  
<<  >>  >>>  
<    >    <=  >=  ==  !=  
&    ^    |  
&&  ||  
?:  
=    op=(*= /= %= += -= etc.)
```

- Para las cadenas se pueden usar los operadores relacionales para comparaciones además de + y += para concatenación:
  - String nombre = “nombre” + “apellido”;

# Separadores

---

- Definen la forma y función del código:
  - `()` → Se usa en listas de parámetros en la definición y llamada a métodos. También para definir precedencia
  - `{}` → Para contener valores de matrices inicializadas y para definir bloques de código
  - `[]` → Para declarar tipos matriz. También para referenciar valores de matriz
  - `;` → Separa sentencias
  - `,` → Separa identificadores consecutivos en una declaración de variables. Para encadenar sentencias en un for. Para separar parámetros pasados a una función
  - `.` → Para separar nombres de paquete de sub-paquetes y clases

# Sentencias de control de flujo

---

## a) Estructura de selección if ... else

```
if (condición) {  
    instrucciones; }  
else if (condición) {  
    instrucciones; }  
else {  
    instrucciones; }
```

## b) Estructura de selección switch

```
switch (expresión-int) {  
    case constante-exp1:  
        instrucciones;  
    break;  
    ...  
    case constante-expN:  
        instrucciones;  
    break;  
    default:  
        instrucciones;  
    break;  
}
```

## c) Estructura de repetición for

```
for (i=1;i<=lim_superior;i++) {  
    instrucciones;  
}
```

## d) Estructura de repetición while

```
while (condición) {  
    instrucciones; }
```

## e) Estructura de repetición do-while

```
do {  
    instrucciones;  
} while (condición);
```

# Paquetes

- **Paquete:** Permite agrupar clases e interfaces. Los nombres de los paquetes son palabras separadas por puntos. Similar a las librerías en otros lenguajes (C ,pascal)
- En un programa en JAVA se importan clases o paquetes. Una clase puede hacer referencia a clases en otros paquetes
- **import:** Se utiliza para cargar paquetes de clases (especificando el nombre del paquete y el nombre de la clase)
- Con '\*' se pueden cargar varias clases
  - `import java.Date; /*esto indica al compilador que importe las clases necesarias del paquete java.Date proporcionado por java*/`
  - `import java.awt.*;`
- Paquetes de JAVA:

▪ <code>java.applet</code>	Contiene clases para usar con applets
▪ <code>java.awt</code>	Contiene clases para generar interfaz grafica
▪ <code>java.swing</code>	Contiene clases nuevas para generar interfaz grafica
▪ <code>java.io</code>	Paquete de entrada/salida
▪ <code>java.lang</code>	Contiene clases del lenguaje: Object, Thread ...
▪ <code>java.net</code>	Paquete para aplicaciones de red
▪ <code>java.util</code>	Contiene una miscelánea de clases

# Paquetes de JAVA

---

- **java.applet:** Paquete que contiene la clase Applet y varias interfaces que permiten la creación de applets
- **java.awt:** Paquete que contiene todas las clases e interfaces necesarias para crear y manipular interfaces gráficas con el usuario (awt, Abstract Window Toolkit Package)
- **java.awt.image:** Paquete que contiene clases e interfaces que permiten almacenar y manipular imágenes en un programa
- **javax.swing:** Mejora las clases proporcionadas por el paquete java.awt. Aquí se definen un conjunto de componentes escritos en JAVA para diseñar interfaces gráficas de usuario
- **java.io:** Paquete que contiene clases que permiten la entrada y salida de datos
- **java.lang:** El compilador importa este paquete en todos los programas. Contiene clases e interfaces básicas
- **java.net:** Paquete que contiene las clases para comunicarse a través de la red
- **java.util:** Paquete que contiene clases e interfaces como: manipulación de fechas, números aleatorios y otros

# Clases de JAVA

---

- Una **clase** es la representación abstracta de un elemento del mundo real con todos sus atributos o características.
- Las clases son lo más simple de JAVA. Todo en JAVA forma parte de una clase. El conocimiento de las clases es fundamental para poder entender los programas JAVA.
- Todas las acciones de los programas JAVA se colocan dentro del bloque de una clase o un objeto.
- Así pues, el esqueleto de cualquier aplicación JAVA se basa en la definición de una clase, cuyo formato se muestra a continuación:

```
class nombre_de_la_clase
{
    cuerpo_de_la_clase
}
```



# Clases de JAVA (cont.)

---

- Todo en JAVA forma parte de una clase.
- JAVA no soporta funciones o variables globales.
- Fuera de la clase sólo deben aparecer los **imports**.
- Tipos de clases:
  - **abstract** Clase que tiene al menos un método abstracto
  - **final** Clase de la cual no se puede heredar
  - **public** Clases accesibles desde otras clases
  - **synchronizable** Clases con métodos sincronizados

# Tipos de clases

---

- **abstract:** Una clase abstract tiene al menos un método abstracto. Una clase abstracta no se instancia, sino que se utiliza como clase base para la herencia.
- **final:** Una clase final se declara como la clase que termina una cadena de herencia. No se puede heredar de una clase final. Por ejemplo, la clase **Math** es una clase final.
- **public:** Las clases public son accesibles desde otras clases, bien sea directamente o por herencia. Son accesibles dentro del mismo paquete en el que se han declarado. Para acceder desde otros paquetes, primero tienen que se importadas.
- **synchronizable:** Este modificador especifica que todos los métodos definidos en la clase son sincronizados, es decir, que no se puede acceder al mismo tiempo desde distintos threads; el sistema se encarga de colocar las banderas (flags) necesarias para evitarlo. Este mecanismo hace que desde threads diferentes se puedan modificar las mismas variables sin que haya problemas de que se sobre-escriban.

# Variables y métodos de instancia

- Una clase en JAVA puede contener variables y métodos:

```
public class MiClase{
    int i;
    public MiClase(){
        i=10;
    }
    public void Suma_a_i(int j)
    {
        i=i+j;
    }
}
```

- Una clase puede tener más de un constructor

- Ámbito de una variable:

```
class Ambito{
    int i=1;
    {
        int i=2; //error
    }
}
```

- Si un bloque de instrucciones contiene dentro otro bloque de instrucciones, en el bloque interior no se puede declarar una variable con el mismo nombre que otra del bloque exterior.

- Método: Función que puede ser llamada dentro de la clase o por otras clases

## Variables y métodos de instancia (cont.)

---

- Para la creación de una instancia de la clase debemos usar siempre ***new***

```
MiClase mc;  
mc = new MiClase();  
mc.i++;  
mc.Suma_a_i(10);
```

- Finalizadores:

- JAVA no utiliza destructores
- Usa un *thread* llamado *garbage collector*
- Podemos hacer uso del método *finalize* para cerrar un canal (flujos, recursos, ficheros, pero no así con la memoria)

```
protected void finalize() {  
    close();  
}
```

# Alcance de objetos y reciclado de memoria

---

- Garbage collector (recolector automático de basura)

```
String s;           //no se ha asignado memoria todavía  
s = new String("abc"); //memoria asignada  
s = "def";          //se ha asignado nueva memoria  
                    //(nuevo objeto)
```

- Al final de la tercera sentencia el primer objeto se ha salido del alcance y por lo tanto es marcado y luego eliminado
- Esto lo lleva a cabo el thread

# Herencia

---

- Para crear una clase derivada de otra se usa la palabra reservada ***extends***.
- Se pueden sustituir los métodos proporcionados por la clase base.

```
import MiClase;
public class MiNuevaClase extends MiClase {
    public void Suma_a_i(int j) {
        i = i + ( j / 2 );
    }
}
MiNuevaClase mnc;
mnc = new MiNuevaClase();
mnc.Suma_a_i(10);
```

- En JAVA **no existe** la herencia múltiple.
- Se pueden compartir métodos por medio de interfaces.

# Control de acceso

---

- Especificadores de acceso: public, protected, private y friendly
- **public:**
  - Puede acceder cualquier clase a las variables y métodos
- **protected:**
  - Pueden acceder solamente las subclases de la clase
- **private:**
  - Las variables y métodos sólo se pueden acceder dentro de la clase. Las subclases no pueden acceder
- **friendly:**
  - (Por defecto) Las variables y métodos son accesibles por todos los objetos dentro del mismo paquete

# Variables y métodos estáticos

---

- **static:** Para crear una variable con el mismo valor para todos los objetos (existe una única copia)

```
class Documento extends Pagina {  
    static int version = 10;  
}
```

- Un método *static* sólo puede acceder a variables estáticas

```
class Documento extends Pagina {  
    static int version = 10;  
    int numero_de_capitulos;  
    static void annade_un_capitulo() {  
        numero_de_capitulos++; //error  
    }  
    static void modifica_version(int i) {  
        version++;  
    }  
}
```



# this y super

---

- **this:** Hace referencia a los miembros de la propia clase
- **super:** Hace referencia a los métodos de la clase padre

```
public class MiClase {  
    int i;  
    public MiClase() {  
        i=10;  
    }  
    public MiClase(int valor) {  
        this.i = valor;  
    }  
    public void Suma_a_i(int j) {  
        i = i + j;  
    }  
}
```

```
import MiClase;  
public class MiNuevaClase extends MiClase {  
    public void Suma_a_i(int j) {  
        i = i + (j / 2);  
        super.Suma_a_i(j);  
    }  
}  
  
MiNuevaClase mnc;  
mnc = new MiNuevaClase();  
Mnc.Suma_a_i(10);
```

# Clases abstractas

---

- **Clase abstracta:** Clases que declara una serie de métodos sin definirlos; la implementación se hace en las subclases

```
public abstract class Graphics {  
    public abstract void drawLine(int x1, int y1, int x2, int y2);  
    public abstract void drawOval(int x, int y, int width, int height);  
    ...  
}  
  
public class MiClase extends Graphics {  
    public void drawline(int x1, int y1, int x2, int y2){  
        <código para pintar líneas>  
    }  
}
```

## Clases abstractas (cont.)

---

- Cuando una clase contiene un método abstracto tiene que declararse abstracta
- No todos los métodos de una clase abstracta tienen que ser abstractos
- Las clases abstractas no pueden tener métodos privados
- No pueden tener métodos estáticos
- Una clase abstracta tiene que derivarse
- No se pueden crear instancias de una clase abstracta
- Tiene similitud con las clases abstractas de C++

# Interfaces

---

- **Interface:** Contiene una colección de métodos que se implementan en otro lugar
- La diferencia con las clases abstractas es que no es necesario usar la herencia

```
public interface VideoClip {  
    void play();  
    void bucle();  
    void stop();  
}  
  
class MiClase implements VideoClip {  
    void play() { <código> }  
    void bucle() { <código> }  
    void stop { <código> }  
}
```

- Desde MiClase se puede hacer uso de play(), bucle() y stop()

# Referencias en JAVA

---

- No son punteros ni referencias como en C++
- Son identificadores de instancias de las clases JAVA

```
public class Habitacion {  
    private int numHabitacion;  
    private int numCamas;  
    public Habitacion(int h, int c) {  
        numHabitacion = h;  
        numCamas = c;  
    }  
}
```

```
public class Hotel1 {  
    public static void main (String [] args) {  
        Habitacion llaveHab1;  
        Habitacion llaveHab2;  
  
        llaveHab1 = new Habitacion(10,20);  
        llaveHab2 = new Habitacion(15,30);  
  
        Habitacion llaveHab3, llaveHab4;  
        llaveHab3 = llaveHab1;  
        llaveHab4 = llaveHab2;  
        ...  
    }  
}
```

## Referencias en JAVA (cont.)

---

- En C y C++ podemos usar punteros (nos da mayor libertad al trabajar con la memoria)
- Pero esto nos puede llevar a colgar el sistema
- En JAVA esto no puede suceder ya que disponemos del recolector de basura (él se encarga de liberar la memoria)
- Con el uso de referencias en JAVA tenemos un mayor nivel de protección
- Podemos declarar un array de referencias, esto es, un array de instancias de una clase

```
Habitacion llavesMaestras[];  
llavesMaestras = new Habitacion[habitaciones];  
for(int i=0; i<habitaciones; i++)  
    llavesMaestras[i] = new Habitacion(h,c);
```

# Una mínima aplicación en JAVA

## ▪ Aplicación HolaMundo:

```
//Aplicación HolaMundo de ejemplo      1
class HolaMundoApp {                    2
    public static void main (String [] args) { 3
        System.out.println("Hola Mundo!");    4
    }                                          5
}                                             6
```

1: Línea de comentario

2: Declaración de la clase HolaMundoApp. Se utiliza para crear el fichero HolaMundoApp.class en el directorio actual

3: Método main, que se ejecuta en primer lugar

*public* indica que puede ser llamado por cualquier otro método

*static* indica que sólo existe una copia del método (no se instancia)

*void* indica que main() no devuelve nada

args[] declaración de un array de Strings (Almacena los argumentos por línea de órdenes pasados al programa)

4: Se usa el método println() de la clase out que está en el paquete System para imprimir el mensaje "Hola Mundo!"

5: Llave de cierre de main()

6: Llave de cierre de la clase

# Compilación y ejecución

---

- Los ficheros fuentes deben terminar con la extensión .java
- Para crearlos y editarlos podemos usar cualquier editor (notepad – windows, vim - linux)
- Compilación: `javac nombre_programa.java`
- Luego se genera el archivo con extensión .class
- Ejecución: `java nombre_programa`
- Con esto visualizamos por pantalla:  
    Hola Mundo!



# Problemas de compilación

---

- **javac: command not found**

Previamente, para que el sistema operativo encuentre la utilidad *javac*, utilizando la línea de órdenes hay que añadir a la variable de entorno *path* la ruta de la carpeta donde se encuentra esta utilidad

*set path=%path%;c:\java\jdk1.5.0\bin*

- **HolaMundoApp.java:3: Method printl (java.lang.String) not found in class java.io.PrintS System.out.printl("Hola Mundo!");**

Error tipográfico, el método es println no printl

- **In class HolaMundoApp: main must be public and static**





Se olvidó colocar la palabra static en el método main()

- **Can't find class HolaMundoApp**

El nombre de la clase es distinto al del fichero que contiene el código fuente ...

Ojo!! Este es un error muy frecuente

# Otra mínima aplicación en JAVA

```
public class UnaClase {  
    private int numero1;  Variable miembro o atributo de la clase  
    public void calcular() {  
        int a = 1;  Variable local del método  
        {  
            System.out.println(a + ", " + numero1);  
            int b = 2;  Variable de bloque  
            System.out.println(a + ", " + b);  
            {  
                int c = 3;  Variable de bloque  
                System.out.println(a + ", " + b + ", " + c);  
            } // Fin del ámbito de c. Final del bloque donde se ha declarado  
            System.out.println(a + ", " + b + ", " + c); // esta línea provoca un  
                                                    // error de compilación.  
                                                    // c esta fuera de su ámbito  
                                                    // y por tanto, no declarada  
        } //Fin del ámbito de b. Final del bloque donde se ha declarado  
    } // Fin del ámbito de a. Final del método calcular  
} //Fin del ámbito de número1. Final de la clase
```