

Programación I

Tema 3: Introducción a Java

¿Qué es exáctamente Java?

- Lenguaje de programación de Alto Nivel
- Orientado a Objetos
- Adecuado a la red
 - Aplicaciones cliente
 - Aplicaciones servidor

Java

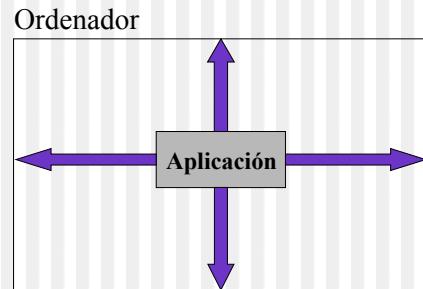
- Características
- Tipos de aplicaciones
- Arquitectura de desarrollo
- Anatomía de una aplicación
- El primer programa Java

Tipos de aplicaciones Java

- Aplicaciones monopuesto
- Aplicaciones red
 - Cliente
 - Applets
 - Servidor
 - Servlets
 - JSP
 - EJB

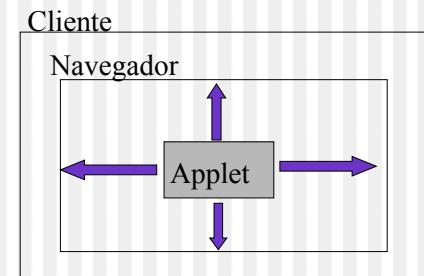
Aplicaciones monopuesto

- Son aplicaciones Java que se ejecutan localmente sin limitaciones



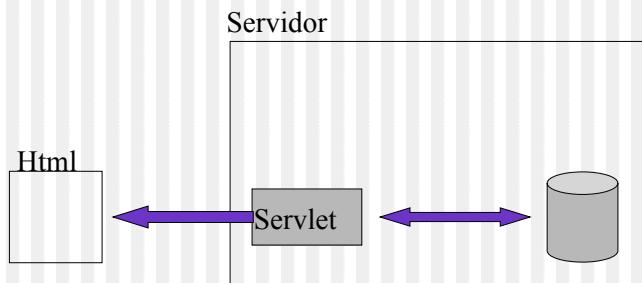
Applet

- Son aplicaciones Java que se ejecutan bajo el contexto de un navegador. Se ejecutan en un contexto seguro.



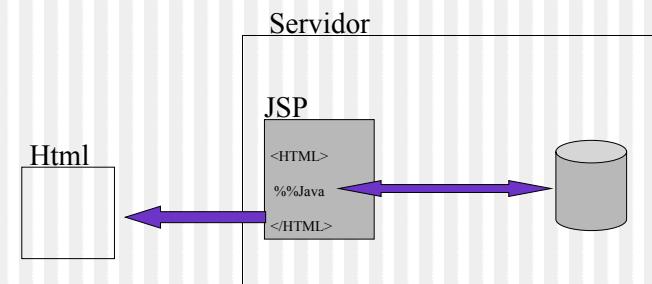
Servlets

- Aplicaciones Java que devuelven como resultado HTML. Normalmente se utilizan para la generación de páginas dinámicas o control.



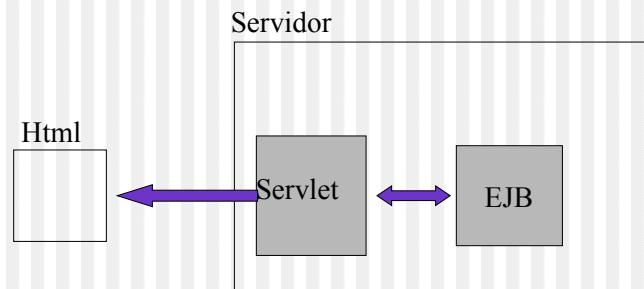
JSP

- Páginas HTML con código Java embebido. Normalmente el código Java suele ser el encargado de acceder a alguna fuente de datos y llenar parte de la página.

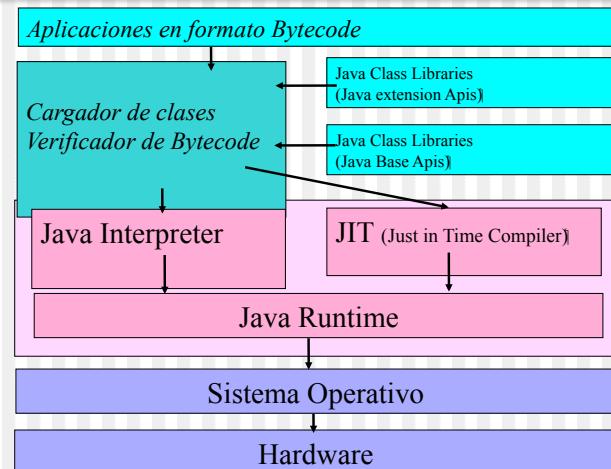


Enterprise JavaBeans (EJB)

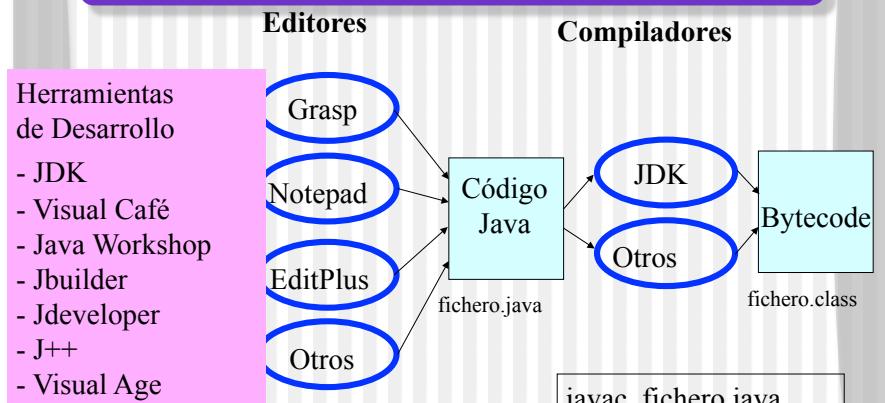
- Piezas de software reutilizables implementados en Java.



Arquitectura de ejecución



Arquitectura de desarrollo



Arquitectura de ejecución

- Librería de clases de java

- Java Base API
 - Java.lang: Interfaz y núcleo de clases
 - Java.util: Clases de utilidades
 - Java.io: Clases de input/output
 - Java.net: Clases de red (Basada en TCP/IP)
 - Java.applet: Clase Applet
- Otras librerías
 - JGL (*Comercial*)
 - JDBC
 - Swing

Arquitectura de ejecución

- Cargador de clases
 - Verifica los bytecodes de aplicación
 - Carga las clases requeridas por el bytecode
- JVM (Java Virtual Machine)
 - Ejecuta el pseudocódigo (*bytecodes*)
 - Dependiente del navegador Web y el S.O.
 - Permite dos alternativas
 - Interpretar el código (Java Interpreter)
 - Compilar de forma dinámica

Por dónde empezar

- Editor : Jbuilder
- Entorno de desarrollo: JDK (JBuilder)
 - Variables de entorno
 - CLASSPATH: La lista de directorios que contienen los ficheros.class que queremos ejecutar. Debería contener al menos \$JAVA_HOME/lib/classes.zip
 - PATH: Lista de búsqueda de los ejecutables debería contener al menos \$JAVA_HOME/bin

¿Qué puede fallar?

Compilar	→ “Syntax Error”
Cargar	→ “Class not found Exception”
Verificar	→ “Security Exception”
Ejecutar	→ “Null Pointer Exception”

Variables de entorno

```
set PATH=c:\jdk1.2\bin;C:\WINDOWS\COMMAND\  
set CLASSPATH=c:\jdk1.2\lib\classes.zip;;
```

Conservando el valor de las variables antiguas:

```
set PATH=c:\jdk1.2\bin;C:\WINDOWS\COMMAND%;%PATH%  
set CLASSPATH=c:\jdk1.2\lib\classes.zip;%CLASSPATH%;;
```

Una aplicación básica en Java

```
HolaMundo.java
// Aplicación de ejemplo
// "Hola Mundo"
//
class HolaMundo{
    public static void main(String args[]){
        System.out.println("Hola Mundo");
    }
}
```

1. Compilamos: `javac HolaMundo.java`

2. Ejecutamos: `java HolaMundo`



Anatomía de una aplicación

```
// Aplicación de ejemplo
// "Hola Mundo"
//
class HolaMundo{
    public static void main(String args[]){
        System.out.println("Hola Mundo");
    }
}
```

Comentarios (3 tipos)
// Para comentar una línea
/*Para comentar
una línea o más*/
/**Para documentación usando javadoc*/

Anatomía de una aplicación

```
// Aplicación de ejemplo
// "Hola Mundo"
//
class HolaMundo{
    public static void main(String args[]){
        System.out.println("Hola Mundo");
    }
}
```

Declaración de la clase

El nombre de la clase (que debe ser igual que el del fichero fuente HolaMundo.java) se utiliza para crear el fichero *class*

Anatomía de una aplicación

```
// Aplicación de ejemplo
// "Hola Mundo"
//
class HolaMundo{
    public static void main(String args[]){
        System.out.println("Hola Mundo");
    }
}
```

Método main

- Es el método que busca el intérprete para ejecutar en primer lugar.
- Se le pasa como argumento un array de objetos String llamado args.

Definición de variables y constantes

```
// Aplicación de ejemplo
// "Hola Mundo"
class HolaMundo{
    static int numero;
    public static void main(String args[]){
        String nombre;
        System.out.println("Hola Mundo");
    }
}
```

Declaración de variables o constantes

- Dentro de los métodos o subprogramas
- Fuera (globales)

Variables y Constantes

- Globales
 - Definidas fuera de los métodos (subprogramas)
 - Accesibles para todos los métodos
 - static
- Locales
 - Dentro de un bloque (generalmente un método)
 - Accesible sólo en el bloque en el que se ha definido

Variables y Constantes

- Definición de una variable
 - *Tipo nombreVariable;*
 - [static] int numero;
 - [static] String nombre;
- Definición de una constante
 - *Tipo final nombreConstante = valorinicial;*
 - [static] final float PI = 3.1416f;

Tipos primitivos

- Numéricas

Tipo	Contenido	Capacidad
byte	Entero	1 Byte
Short	Entero	2 Bytes
Int	Entero	4 Bytes
Long	Entero	8 Bytes
Float	Real	4 Bytes
double	Real	8 Bytes

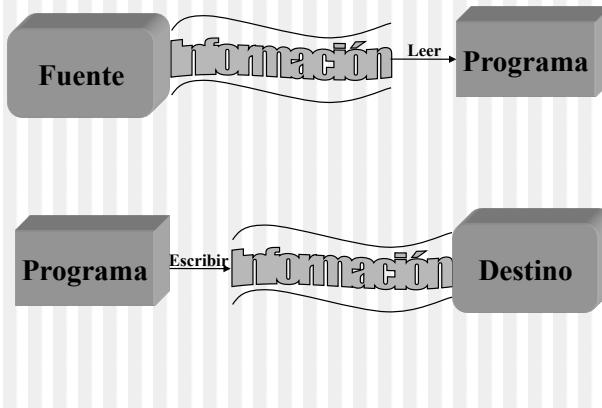
- Carácter
 - Char
- Cadenas de caracteres
 - String

Operaciones

Tipo	Operadores	Ejemplo
Op. Sufijo	[] .(params) exp++ exp--	indice++
Op. Unarios	++exp --exp +exp -exp ! ~	-valor
Creación o Cast	new (type)exp	(int)num
Multiplic.	* / %	num1 * num2
Aditivos	+ -	num1 + num2
Desplazamiento	<< >> >>>	op1 >> 2
Relacionales	< <= > >= instanceof	num1 < 5
Igualdad	== !=	num1 == 5
AND a nivel de Bit	&	ip & 128
XOR a nivel de Bit	^	flags ^ 8
OR a nivel de Bit		flags 16
AND	&&	a>7 && encontrado
OR		a>7 b<6
Condicional	expr?res1:res2	a>7?5:3
Asignación	= += -= *= /= %= &= = ^= >>= <<= >>>=	i+=5

→ Precedencia → ↓

Instrucciones de Lectura/ Escritura



Instrucciones de E/S

- Funcionalidad del paquete **java.io**
 - **import java.io.*;** al principio del fichero
 - Funciones que permiten tanto leer como escribir
 - Conversión del flujo de datos a caracteres
 - Conversión de caracteres a datos

Instrucciones de E/S

- Entrada
 - **System.in**
 - Lectura carácter a carácter (**read**)
 - **java.io.IOException** ← Error de lectura
 - Lectura con buffer
- ```
BufferedReader entrada = new BufferedReader(new
InputStreamReader(System.in));
```

Sólo 1 vez. Al inicio



## Instrucciones de E/S

- Lectura
  - Leer un String

```
try {
 cadena = entrada.readLine();
}
catch (IOException exc)
{
 // Tratamiento de los errores
 exc.printStackTrace(); // Visualiza la traza del error
}
```

## Instrucción de E/S

- Lectura
  - Leer un Real

```
cadena = entrada.readLine();
numReal = Float.valueOf(cadena).floatValue();
```

Puede generar el error

**java.lang.NumberFormatException**

```
cadena = entrada.readLine();
numReal = Float.parseFloat(cadena);
```

## Instrucción de E/S

- Lectura
  - Leer un Entero

```
cadena = entrada.readLine();
numEntero = Integer.valueOf(cadena).intValue();
```

Puede generar el error  
**java.lang.NumberFormatException**

```
cadena = entrada.readLine();
numEntero = Integer.parseInt(cadena);
```

## Instrucciones de E/S

- Conversión de String a datos numéricos

- Short
  - shortValue
  - parseShort
- Integer
  - intValue
  - parseInt
- Long
  - longValue
  - parseLong
- Float
  - floatValue
  - parseFloat
- Double
  - doubleValue
  - parseDouble

## Instrucciones de E/S

- Escritura
  - **System.out.println(datos);**
    - System.out.println("El resultado es " + res);
  - **System.out.print(datos);**
    - System.out.print("El resultado es " + res);
- Carácteres especiales
  - \n ← salto de línea
  - \t ← tabulador

## Ejemplo Lectura/Escritura

```
import java.io.*;
public class HelloWorld {
 public static void main(String[] args) {
 int numEntero;
 String linea;
 BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Introduce un número entero:");
 try {
 linea = entrada.readLine();
 numEntero = Integer.parseInt(linea);
 // Equivale a numEntero = Integer.valueOf(linea).intValue();
 }
 catch (IOException exc) {
 exc.printStackTrace();
 }
 catch (NumberFormatException nExc) {
 nExc.printStackTrace();
 }
 System.out.print("Hello World");
 }
}
```

## Ejemplo Lectura/Escritura

```
import java.io.*;
public class HelloWorld {
 public static void main(String[] args) {
 int numEntero;
 String linea;
 BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Introduce un número entero:");
 try {
 linea = entrada.readLine();
 try {
 numEntero = Integer.valueOf(linea).intValue();
 }
 catch (NumberFormatException nExc) {
 nExc.printStackTrace();
 }
 }
 catch (IOException exc) {
 exc.printStackTrace();
 }
 System.out.print("Hello World");
 }
}
```

## Lectura/Escritura - Ficheros

- Lectura/Escritura en ficheros binarios
  - FileInputStream/FileOutputStream
- Lectura/Escritura en ficheros de texto
  - FileReader/FileWriter
- Es conveniente el uso de *Buffers* para minimizar los accesos a la memoria secundaria

## Clase File

- Para manipular un fichero, primero hay que crear una instancia de **File** (del paquete **java.io**).

```
File fichEnt = new File("ejemplo.dat");
```

Abre el fichero "ejemplo.dat" en el directorio actual.

```
File fichEnt = new File("C:\\ProgEj", "prim.txt");
```

Abre el fichero "prim.txt" en el directorio C:\\ProgEj. Atención al uso del carácter de escape \.

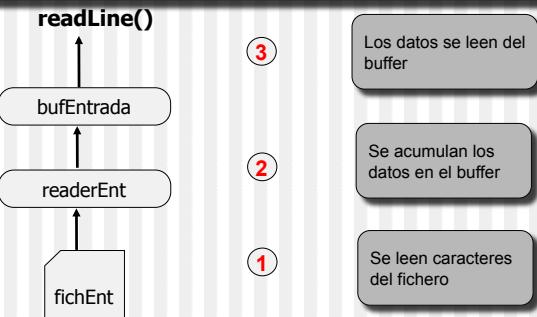
```
File fichEnt = new File("C:/\\ProgEj/test.dat");
```

Abre el fichero "test.dat" en el directorio C:\\ProgEj usando el separador de ficheros genérico / y facilitando el camino absoluto

```
File fichEnt = new File("C:/ProgEj/test.dat");
```

## Lectura en ficheros de texto

```
File fichEnt = new File("ejemplo1.txt");
FileReader readerEnt = new FileReader(fichEnt);
BufferedReader bufEntrada = new BufferedReader(readerEnt);
```



## Algunos métodos

```
if (fichEnt.exists()) { ... }
```

Para comprobar si **fichEnt** está vinculado a un **fichero real** correctamente

```
if (fichEnt.isFile()) { ... }
```

Para determinar si **fichEnt** está vinculado a un **fichero o a un directorio**.

```
File folder = new File("C:/ProyJava/Cap_11");
String nombreFicheros[] = folder.list();
for (int i=0; i < nombreFicheros.length; i++)
{
 System.out.println(nombreFicheros[i]);
}
```

Enumera los nombres de **todos los ficheros** del directorio C:\\ProyJava \\Cap\_11.

## Lectura en ficheros de texto

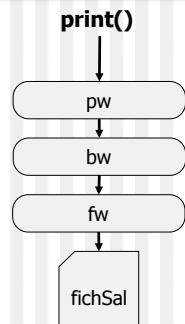
```
// Preparar el fichero
File fichEnt = new File("ejemplo1.txt");
FileReader readerEnt = new FileReader(fichEnt);
BufferedReader bufEntrada = new BufferedReader(readerEnt);
String str;

str = bufEntrada.readLine();
int i= Integer.parseInt(str);

bufEntrada.close();
```

## Escritura de ficheros de texto

```
File fichSal = new File("Ejemplo1.txt");
FileWriter fw = new FileWriter(fichSal);
BufferedWriter bw = new BufferedWriter(fw);
PrintWriter pw = new PrintWriter(bw);
```



- ① Los datos se escriben en **pw**
- ② Los datos se acumulan en el buffer
- ③ Los datos se procesan para la escritura.
- ④ Los caracteres se escriben en el fichero

## Escritura de ficheros de texto

```
// Preparar el fichero
File fichSal = new File("Ejemplo1.txt");
FileWriter fw = new FileWriter(fichSal);
BufferedWriter bw = new BufferedWriter(fw);
PrintWriter pw = new PrintWriter(bw);

pw.println(987654321);

// Cerrar el fichero
pw.close();
```

## Escritura de ficheros de texto

```
// Preparar el fichero para añadir datos al final

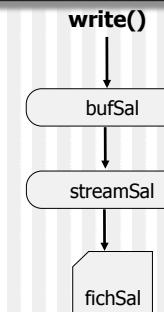
File fichSal = new File("Ejemplo1.txt");
FileWriter fw = new FileWriter(fichSal, true);
BufferedWriter bw = new BufferedWriter(fw);
PrintWriter salida = new PrintWriter(bw);

salida.println(987654321);

// Cerrar el fichero
salida.close();
```

## Escritura en ficheros binarios

```
File fichSal = new File("ejemplo2.data");
FileOutputStream streamSal = new FileOutputStream(fichSal);
BufferedOutputStream bufSal = new BufferedOutputStream(streamSal);
```



- ① Los datos se acumulan en el buffer
- ② Se procesan los datos para la escritura
- ③ Los bytes se escriben en el fichero

## Escritura en Ficheros binarios

```
// Preparar el fichero y el gestor de salida
File fichSal = new File("ejemplo2.data");
FileOutputStream streamSal = new FileOutputStream(fichSal);
BufferedOutputStream bufSal = new BufferedOutputStream(streamSal);

// Datos a guardar
byte[] arrayBytes = {10, 20, 30, 40, 50, 60, 70, 80};

bufSal.write(arrayBytes);

// Cerrar el gestor de salida y el fichero
bufSal.close();
```

## Lectura en ficheros binarios

```
// Preparar el fichero para la lectura
File fichEnt = new File("ejemplo2.data");
FileInputStream streamEnt = new FileInputStream(fichEnt);
BufferedInputStream bufEnt = new BufferedInputStream(streamEnt);

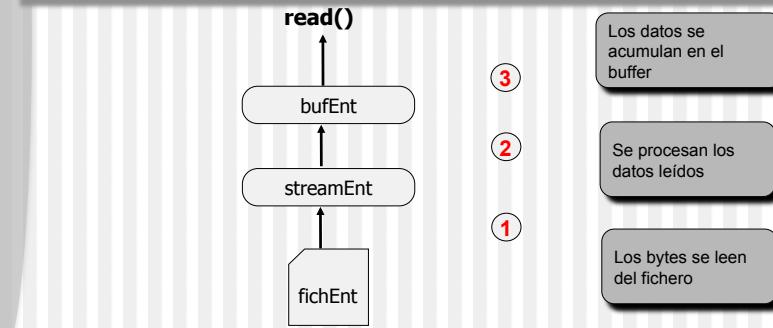
// Preparar el array para la lectura
int tamFich = (int)fichEnt.length();
byte[] matrBytes = new byte[tamFich];

bufEnt.read(matrBytes);

for (int i = 0; i < tamFich; i++)
{
 System.out.println(matrBytes[i]);
}
// Cerrar el fichero
bufEnt.close();
```

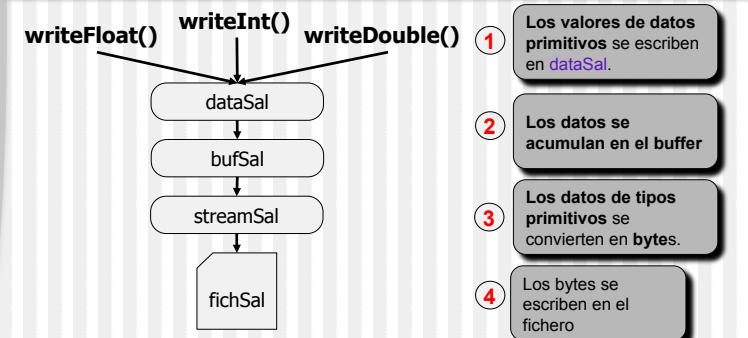
## Lectura en ficheros binarios

```
File fichEnt = new File("ejemplo2.data");
FileInputStream streamEnt = new FileInputStream(fichEnt);
BufferedInputStream bufEnt = new BufferedInputStream(streamEnt);
```



## Escritura de datos primitivos

```
File fichSal = new File("ejemplo3.data");
FileOutputStream streamSal = new FileOutputStream(fichSal);
BufferedOutputStream bufSal = new BufferedOutputStream(streamSal);
DataOutputStream dataSal = new DataOutputStream(bufSal);
```



## Escritura de datos primitivos

```
File fichSal = new File("ejemplo3.data");
FileOutputStream streamSal = new FileOutputStream(fichSal);
BufferedOutputStream bufSal = new BufferedOutputStream(streamSal);
DataOutputStream dataSal = new DataOutputStream(bufSal);

dataSal.writeDouble(13.2);

dataSal.close();
```

## Lectura de datos primitivos

```
File fichEnt = new File("ejemplo3.data");
FileInputStream streamEnt = new FileInputStream(fichEnt);
BufferedInputStream bufEnt = new BufferedInputStream(streamEnt);
DataInputStream dataEnt = new DataInputStream(bufEnt);

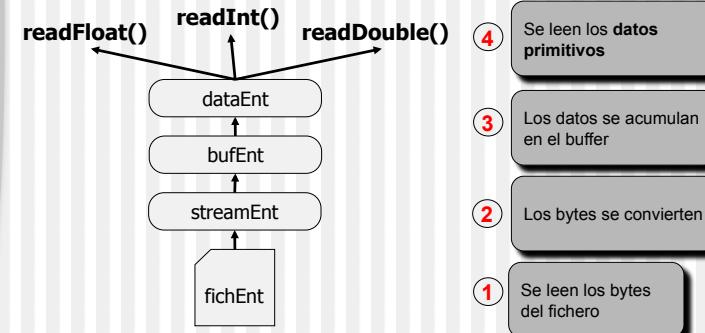
Double d2 = dataEnt.readDouble();

System.out.println(d2);

dataEnt.close();
```

## Lectura de datos primitivos

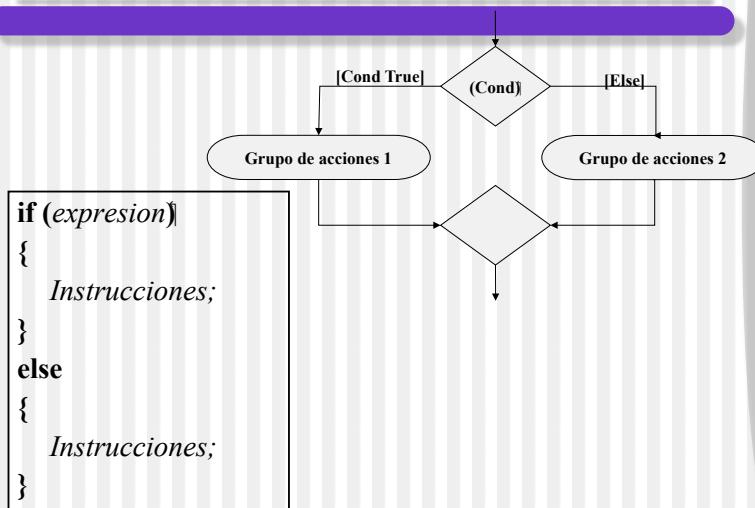
```
File fichEnt = new File("ejemplo3.data");
FileInputStream streamEnt = new FileInputStream(fichEnt);
BufferedInputStream bufEnt = new BufferedInputStream(streamEnt);
DataInputStream dataEnt = new DataInputStream(bufEnt);
```



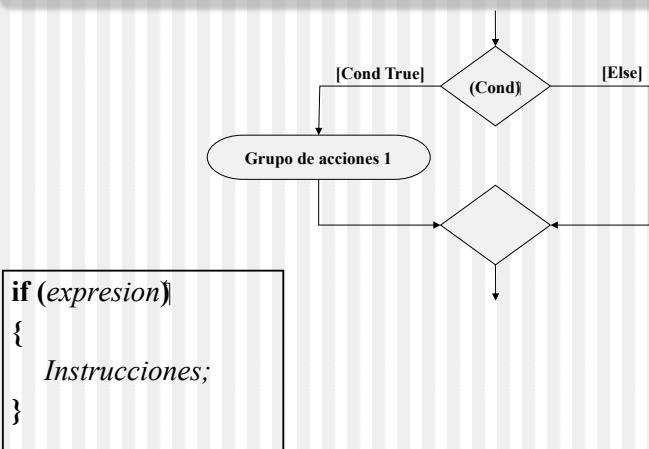
## Excepciones

- FileNotFoundException
  - Al abrir el fichero
    - FileInputStream, FileOutputStream,...
- IOException
  - Cerrar fichero
  - Escribir
  - Leer
  - ...
- EOFException

## Instrucciones condicionales



## Instrucciones condicionales



## Instrucciones condicionales

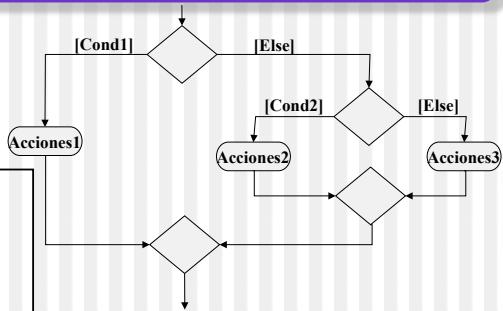
```
if (num % 2 == 0)
{
 System.out.println("El número es par");
}
else
{
 System.out.println("El número es impar");
}
```

## Instrucciones condicionales

```
if (num < 0)
{
 num = -num;
}
System.out.println("El ABS del número es " + num);
```

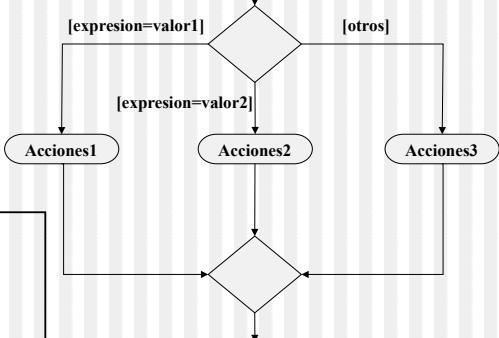
## Instrucciones condicionales

```
if (expresión1)
{
 Instrucciones;
}
else if (expresión2)
{
 Instrucciones;
}
else
{
 Instrucciones
}
```



## Instrucciones condicionales

```
switch (expresión1) {
 case valor1:
 Instrucciones;
 [break];
 case valor1:
 Instrucciones;
 [break];
 default
 Instrucciones
}
```



## Instrucciones condicionales

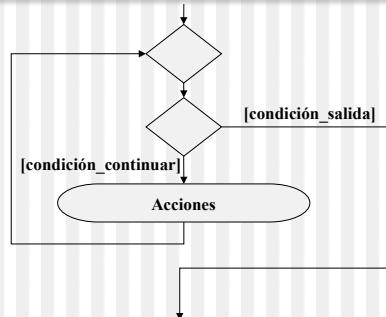
```
if (nota < 5)
{
 System.out.println("Suspensos");
}
else if (nota >= 5 && nota < 7)
{
 System.out.println("Aprobado");
}
else
{
 System.out.println("Te has pasado");
}
```

## Instrucciones condicionales

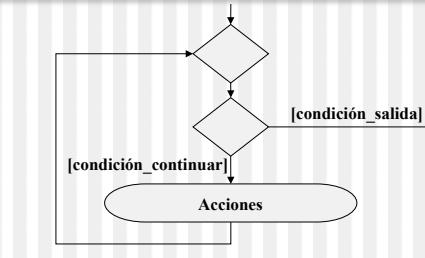
```
switch ((int)nota) {
 case 5:
 case 6:
 System.out.println("Aprobado");
 break; // Para que no se ejecuten las instrucciones que vienen detrás
 case 7:
 case 8:
 System.out.println("Notable");
 break; // Para que no se ejecuten las instrucciones que vienen detrás
 case 9:
 System.out.println("Sobresaliente");
 break; // Para que no se ejecuten las instrucciones que vienen detrás
 default
 System.out.println("Suspensos");
}
```

## Instrucciones Iterativas

```
while (expresión)
{
 Instrucciones;
}
```



## Instrucciones Iterativas



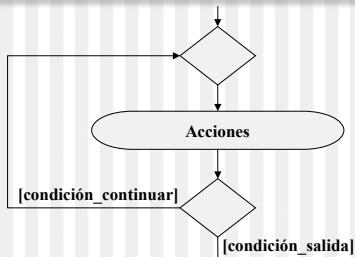
## Instrucciones Iterativas

```
linea = entrada.readLine();
num = Integer.parseInt(linea);
while (num!=0)
{
 System.out.println(num);
 linea = entrada.readLine();
 num = Integer.parseInt(linea);
}
```

## Instrucciones Iterativas

```
for (int num = 0; num <= 10; num++)
{
 System.out.println(num);
}
```

## Instrucciones Iterativas



```
do
{
 Instrucciones;
} while (expresion);
```

## Instrucciones Iterativas

```
do
{
 System.out.println("Introduce un nº positivo");
 linea = entrada.readLine();
 num = Integer.parseInt(linea);
} while (num < 0);
System.out.println("El número es " + num);
```

## Instrucciones Iterativas

- **break**
  - Sale del bloque ({...}) en el que se encuentra.
  - Puede simplificar el código
- **continue**
  - Salta a la siguiente iteración del bucle
  - Puede simplificar el código