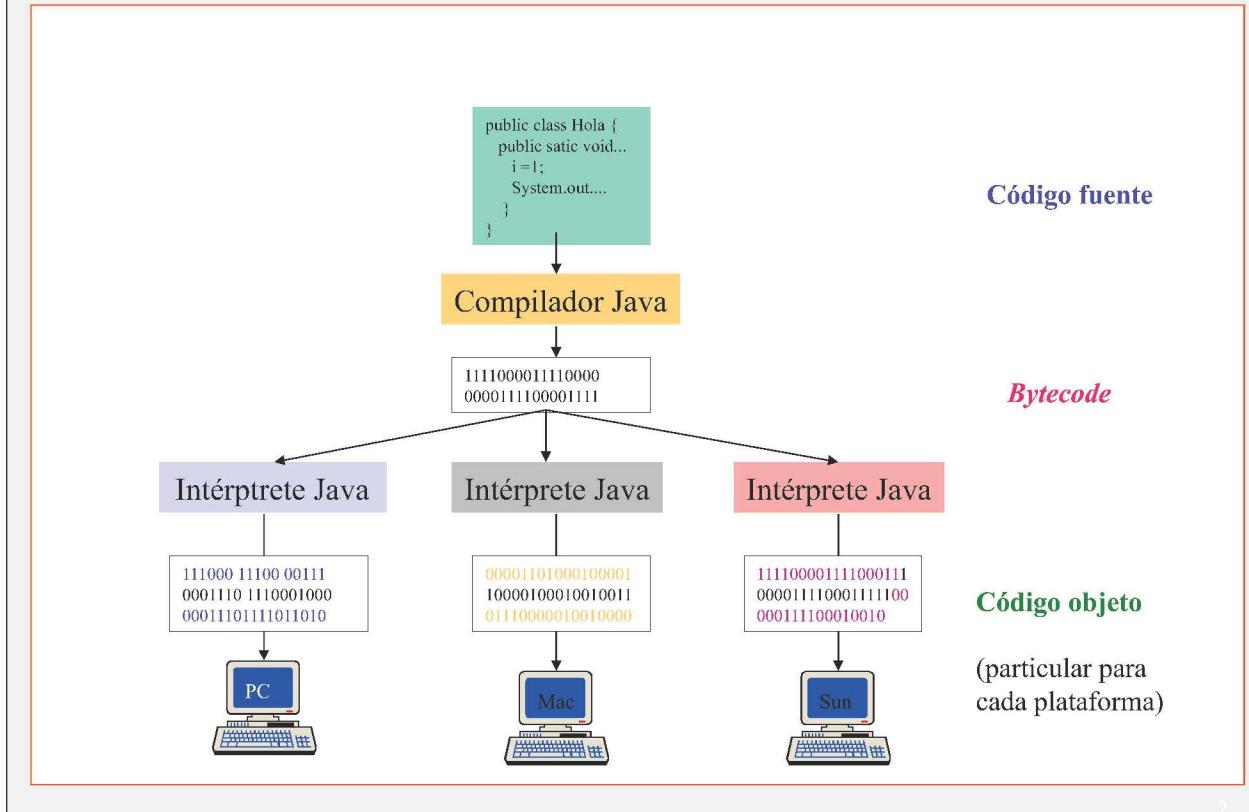


# Desarrollo y ejecución de aplicaciones en Java



## Una aplicación en Java

- ❖ Se estructura como una **clase** (no privada) que tiene un método **main**. La aplicación puede hacer uso de otras clases.
- ❖ Un archivo puede contener más de una clase. Sólo una de ellas puede ser **no privada**.
- ❖ El **nombre un archivo** debe ser el mismo que el de la **clase no privada** que contiene.
- ❖ Java distingue entre **mayúsculas** y **minúsculas**.

```
/**  
Estructura de una aplicación  
*/  
public class IdentificadorClase {  
    public static void main (String[ ] idArreglo ) {  
        ...  
    } // Fin método main  
} // Fin clase
```

## Compilación y ejecución de una aplicación

- ❖ Para **ejecutar** una aplicación se debe indicar a la **JVM** el **nombre de la clase** que contiene el método **main**:

```
C:\temp>java MiTarea
```

- ❖ Para **compilar** una clase se debe indicar al **compilador** el **nombre del archivo** que contiene el código a compilar:

```
C:\temp>javac MiTarea.java
```

- ❖ Recordar nuevamente que Java es sensible a mayúsculas y minúsculas

## Tipos de datos

- ❖ Los tipos de datos utilizados por programas en Java se clasifican en:
  - **Primitivos**: sirven para definir variables que guardan valores numéricos, lógicos y caracteres unitarios.
  - **Clases e interfaces**: sirven para definir variables que almacenan tipos de datos estructurados, con las funciones asociadas.
- ❖ Java requiere la **declaración** de variables antes de que sean utilizadas. El tipo de una variable declarada no puede ser modificado. Se dice que es un lenguaje fuertemente “tipificado”.

## Tipos de datos primitivos

- ❖ Entero
    - byte entero 8 bit -128..127
    - short entero 16 bit -32.768..32.767
    - int entero 32 bit -2.147.483.648.. 2.147.483.647
    - long entero 64 bit -9.223.372.036.854.755.808..  
9.223.372.036.854.755.807
  - ❖ Punto flotante
    - float p flotante 32 bit 7 dígitos significativos ( $10^{-46}, 10^{38}$ )
    - double p flotante 64 bit 15 dígitos significativos ( $10^{-324}, 10^{308}$ )
  - ❖ Caracter
    - char carácter Unicode
  - ❖ Lógico
    - boolean lógico false, true

# Operadores

- ❖ Asignación
    - =
  - ❖ Numéricos
    - Binarios: + (suma), - (resta), \* (producto), / (división)
    - Unarios: ++ (auto incremento), -- (auto decremento)
  - ❖ Relacionales:
    - == (igual), != (distinto), < (menor), <= (menor o igual), > (mayor), >= (mayor o igual)
  - ❖ Lógicos:
    - && (and), || (or), ! (not)

Nota: Las expresiones matemáticas se evalúan con precedencia de \* y / sobre + y -.

## La clase **String**

- ❖ La clase **String** permite manipular cadenas de caracteres, por ejemplo:

"Hola mundo", "Mario Mora", "12/10/99", "A", "".

- ❖ Para determinar la igualdad entre dos strings se debe usar el método **equals**.

```
if( nombre.equals( nombreAmigo ) )
```

```
while( !opcion.equals( "salir") )
```

```
if( "Euro".equals( monedaIngresada ) )
```

## Métodos útiles de la clase **String**

- ❖ **public char charAt(int index)**: retorna el char correspondiente a la posición index (primera posición es 0).
- ❖ **public int indexOf(char ch)**: retorna la primera posición que ocupa el carácter ch.
- ❖ **public int lastIndexOf(char ch)**: retorna la última posición que ocupa el carácter ch.
- ❖ **public boolean equalsIgnoreCase(String str)**: retorna true si el String es igual al argumento, independientemente de su escritura en mayúsculas o minúsculas.
- ❖ **public String toLowerCase()**: retorna un String equivalente, pero con todas sus letras en minúsculas.
- ❖ **public String toUpperCase()**: retorna un String equivalente, pero con todas sus letras en mayúsculas.
- ❖ **public int length()**: retorna el el largo del String.

## Estructuras de control: decisión

- Simple:

```
if ( condición )  
    instrucción
```

```
if ( condición )  
    instrucción 1  
else  
    instrucción 2
```

- Compleja:

```
if ( condición ) {  
    instrucción 1-1  
    instrucción 1-2  
    ...  
} else {  
    instrucción 2-1  
    instrucción 2-2  
    ...  
}
```

10

## Estructuras de control: iteración

- Mientras - repetir:

```
while ( condición )  
    instrucción
```

```
while ( condición ) {  
    instrucción 1  
    instrucción 2  
    ...  
}
```

- Hacer - mientras:

```
do  
    instrucción  
while ( condición )
```

```
do {  
    instrucción 1  
    instrucción 2  
    ...  
} while ( condición )
```

11

## Estructuras de control: iteración

- Ciclo for:

**for ( [tipo] var=valor inicial ; condición de iteración ; incremento)**  
*instrucción*

**for ( [tipo] var=valor inicial ; condición de iteración ; incremento)**  
*instrucción*

...

}

Ejemplo:

```
for( int i = 1; i < 10 , i++ )  
    System.out.println( "Hola" );
```

12

## Declaración de variables

- ❖ Una variable se declara según el formato:

**tipo identificador [=valor] [, identificador [= valor] ]**

Ejemplo: **int a, b, c**

**double peso = 7.05**

**char letra = 'T'**

**byte edad = 20, veces = 3**

- ❖ Convención: el identificador de una variable de tipo primitivo se escribe en minúsculas.

**Ejemplo:** edad, edadAlumno, estadoCivil, esElUltimo

13

## Clases y métodos útiles

### ❖ Output (por pantalla):

**System.out.print()**

**System.out.println()**

Ejemplo:

`System.out.println("Hello");`

### ❖ Conversión de String a número:

**Byte.parseByte()**

**Short.parseShort()**

**Integer.parseInt()**

**Float.parseFloat()**

**Double.parseDouble()**

Ejemplo:

`int x;  
x = Integer.parseInt( "1050" );`

14

## Ejemplo de una aplicación

```
/**  
 * Programa ejemplo para cálculo del Factorial de un número específico.  
 * @author Franco Guidi  
 */  
  
public class Factorial1 {  
  
    public static void main( String arg[] ) {  
  
        int número, i=1, fact=1;  
        número = 3;  
        while( i <= número ) {  
            fact = fact * i;  
            i++;  
        }  
        System.out.println( "El factorial de " + número + " es " + fact );  
    } // Fin método main  
  
} // Fin clase
```

Clase

Método main

15

## Estructura de una aplicación

- ❖ El método **main** puede recibir **parámetros** desde la línea de comandos del intérprete. Estos se almacenan en las posiciones del **arreglo** de **String** indicado en la declaración del método.

```
/**  
 * Programa ejemplo para cálculo del Factorial de un número específico,  
 * acepta el número proporcionado por parámetro.  
 * @author Franco Guidi  
 */  
public class Factorial2 {  
  
    public static void main( String arg[] ) {  
  
        int número, i=1, fact=1;  
        número = Integer.parseInt( arg[0] );  
        while( i <= número ) {  
            fact = fact * i;  
            i++;  
        }  
        System.out.println( "El factorial de " + número + " es " + fact );  
  
    } // Fin método main
```

ANALIZAR  
MEJORAS AL  
CÓDIGO

16

## Clases y métodos útiles (cont.)

- ❖ Lectura de datos (consola)

**BufferedReader**, **InputStreamReader**, **System.in**,  
**IOException**

ANALIZAR  
MEJORAS AL  
CÓDIGO

```
import java.io.*;  
public class Factorial3 {  
    public static void main( String arg[ ] ) throws IOException {  
        int número, i=1, fact=1;  
        BufferedReader lector = new BufferedReader( new InputStreamReader( System.in ) );  
        String ingresado;  
        System.out.println( "Ingrese número para cálculo de factorial: " );  
        ingresado = lector.readLine();  
        número = Integer.parseInt( ingresado );  
        while( i <= número ) {  
            fact = fact * i;  
            i++;  
        }  
        System.out.println( "El factorial de " + número + " es " + fact );  
    } // Fin método main  
} // Fin clase
```

17

## Valores literales

- ❖ Son valores que aparecen explícitamente en el código fuente. Por ejemplo:

'T' : tipo **char**  
"T", "Pedro", "15" : tipo **String**  
**150** : tipo **int**  
**150.0** : tipo **double**  
**150.0f** : tipo **float**  
**0150**: tipo int **octal**  
**0x150**: tipo int **hexadecimal**

19

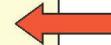
## Valores y variables: promoción automática de tipos

- ❖ Una variable de un tipo de dato puede recibir un valor de otro tipo si:

- los dos tipos son compatibles, y
- el tipo de destino es “de mayor capacidad” que el tipo de origen

Ejemplo:

```
int datoEntero = 100;  
double datoDouble;  
datoDouble = datoEntero;
```



Promoción

19

## Valores y variables: casting

- ❖ Permite convertir un valor de un tipo de mayor capacidad a otro de menor capacidad (funciona si y sólo si el tipo de menor capacidad puede contener al valor convertido). El *casting* tiene prioridad sobre los operadores **+, -, \*, /**.

Formato: **var1 = (tipo de var1) var2**

Ejemplo  
`double j=1.0;  
int i;  
i = (int) j;`

*Correcto*

Ejemplo  
`int j=310;  
byte i;  
i = (byte) j;`

*Incorrecto: no es posible almacenar el valor 310 en una variable byte*

20

## Valores y operaciones

- ❖ La operación de dos números de un mismo tipo, genera un resultado también del mismo tipo.
- ❖ Se exceptúan de la regla anterior las operaciones sobre datos tipo **byte** o **short**: la operación de dos datos de tipo **byte** o **short**, genera como resultado un **int**.
- ❖ El resultado de cualquier expresión aritmética es del tipo correspondiente al del operando de mayor jerarquía, en el orden:
  - **double**
  - **float**
  - **long**
  - **int**

21

## Valores y operaciones (cont.)

- ❖ Suponer variable **x** definida como **double**:

```
x = 10 / 4;           // x toma valor 2.0 double
x = 10.0 / 4;         // x toma valor 2.5 double
x = 10.0f / 4;        // x toma valor 2.5 double
x = (double) 10 / 4;   // x toma valor 2.5 double
x = (double) (10/4);  // x toma valor 2.0 double
```

- ❖ Suponer variable **x** definida como **float**:

```
x = 10 / 4;           // x toma valor 2.0 float
x = 10.0 / 4;          // Error: x no puede almacenar
                      // valor 2.5 double
x = 10.0f / 4;         // x toma valor 2.5 float
x = (float) 10 / 4;    // x toma valor 2.5 float
x = (double) (10/4);  // Error: x no puede almacenar
                      // valor 2.0 double
```

22

## Valores y operaciones: precisión

- ❖ Las operaciones aritméticas con tipos enteros (**byte**, **short**, **int** o **long**), se realizan con exactitud.
- ❖ Las operaciones aritméticas con tipos de punto flotante (**float** o **double**), se realizan con exactitud limitada.
- ❖ Por tanto para comparar “igualdad” de dos variables del tipo punto flotante se debe determinar si su diferencia en valor absoluto es menor de un cierto error aceptable.
  - Ejemplo: si **var1** y **var2** son del tipo **double**:

```
if( Math.abs( var1 - var2 ) < 0.0000001 )
    System.out.println( "Son iguales" );
else
    System.out.println( "No son iguales" );
```

23

## Arrays en Java

- ❖ Cada array se reconoce por un **identificador** y cada dato se almacena en una posición **indexada**.
- ❖ Un array de **largo N**, tiene posiciones indexadas mediante **enteros** desde **0** hasta **N-1**.

nombres					
	“Ana”	“Juan”	“Pedro”	“Pía”	“Mario”
Posición:	0	1	2	3	4

- ❖ El **máximo largo** posible para un array corresponde al mayor valor permitido para un **int**.
- ❖ La definición de variable, instanciación del array y su asignación a la variable puede realizarse en una sola instrucción:  
**tipo[ ] variable = new tipo[ entero ]**
- ❖ No es posible modificar el largo de un array instanciado.

24

## Arrays en Java (cont.)

- ❖ Ejemplo de instanciación:

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

- ❖ Instanciación a partir de expresión literal:

```
int[] nota ;  
nota = { 23, 14, 55, 18 } ;
```

- ❖ El largo de un array se puede conocer por medio de la variable **length**:

```
while( i < nota.length ) {  
    System.out.println( nota[ i ] );  
    i++;  
}
```

25

## Arrays de arrays

- ❖ Java permite implementar matrices bidimensionales de la siguiente forma:

**tipo[ ][ ] variable = new tipo[ entero1 ][ entero2 ]**

- ❖ Ejemplo

```
double[ ][ ] utilidad;
utilidad = new double[10][15];
```

- ❖ En realidad Java maneja arrays de arrays:

- los arrays de un array no requieren ser de la misma longitud

```
int[ ][ ] nota;
nota = { {-1,7,15}, {3, 2}, {4,-3,12}, {3} };
```

26

## Clases en Java

- ❖ Una clase en Java se compone (normalmente) de:

- Variable(s)
- Constructore(s)
- Método(s)

- ❖ Convención: el nombre de una clase se escribe con la primera letra de cada palabra en mayúsculas (nomenclatura camello):

- EsteEsUnEjemplo

```
public class CajaAhorro {
    private int saldo;
    private int transacciones;
    public CajaAhorro() {
        saldo = 0;
        transacciones = 0;
    }
    public void depositar( int monto ) {
        saldo = saldo + monto;
        transacciones++;
    }
    public void girar( int monto ) {
        saldo = saldo - monto;
        transacciones++;
    }
    public int obtenerSaldo() {
        return saldo;
    }
    public int obtenerTransacciones() {
        return transacciones;
    }
}
```

27

## Clases y variables

### ❖ Variables de instancia:

- Pertenecen a cada objeto o instancia de clase.

### ❖ Variables de clase:

- Compartidas por todas las instancias de la clase. Se declaran como "static".

```
public class Factura {  
    public int numero;  
    public static int contFacturas;  
    ...  
}
```

Variable de instancia  
Variable de clase

28

## Visibilidad de variables de clase o de instancia

### ❖ Modificadores de visibilidad de una variable de clase o de instancia:

- **public**: visibilidad general.
- **private**: visibilidad limitada a la clase en que ha sido declarada. No se heredan.
- **protected**: visibilidad limitada a la clase en que ha sido declarada, y a sus subclases.
- **default** (omisión): visibilidad dentro del package.

29

## Constructores de clases

- ❖ En Java todas las clases tienen un constructor. Si no se lo especifica, Java asigna a la clase el constructor por omisión (sin parámetros, sin código).
- ❖ Los constructores deben tener el mismo nombre que la clase a la que pertenecen. Se distinguen entre ellos por el número y tipo de sus parámetros (sobrecarga de constructores).

```
public class Reloj {  
    private int horas, minutos, segundos;  
  
    public Reloj(int hh, int mm, int ss) {  
        horas = hh;  
        minutos = mm;  
        segundos = ss;  
    }  
    ...  
}
```

30

## Clases y métodos

### ❖ Métodos de instancia:

- Proveen operaciones que se invocan sobre objetos (instancias de clases)...  
... por tanto tienen acceso a las variables de instancia del objeto.

```
public class Cuenta {  
    ...  
    public int getSaldo(){  
        ...  
    }  
}
```

```
Cuenta miCuenta;  
...  
x = miCuenta.getSaldo();
```

31

## Clases y métodos (cont.)

### ❖ Métodos estáticos:

- No se ejecutan sobre un objeto...  
... por tanto no tienen acceso a las variables de instancia de objetos.
- Se declaran con el modificador “**static**”.
- Sirven para proveer funciones de tipo general. Ejemplo: método **parseInt** de la clase **Integer**.
- Se invocan sobre la clase (o sobre una instancia):
- Notar que el **main** de una aplicación es un método estático.

```
public class Cuenta {  
    ...  
    public static int convierte (String Z){  
        ...  
    }  
  
    x = Cuenta.convierte( "1001" );
```

32

## Visibilidad de los constructores y métodos de una clase

### ❖ Modificadores de visibilidad de un método:

- **public** : visibilidad general.
- **private**: visibilidad limitada a la clase en que ha sido declarado (i.e. sólo otros métodos y constructores de la misma clase).
- **protected**: visibilidad limitada a la clase en que ha sido declarado, y a sus subclases.
- **default** (omisión): visibilidad dentro del package.

33

## Parámetros en constructores y métodos

- ❖ Formato para declarar parámetros:  
`( tipo1 var1, tipo2 var2, ..., tipoN varN)`
- ❖ Si un constructor o método tiene parámetros denominados igual que variables de instancia, los primeros ocultan a estas últimas. La palabra reservada `this` permite hacer una autorreferencia al objeto, y acceder a sus propiedades.

```
public class Punto {  
    private int x;  
    private int y;  
    ...  
    public void setCoordenadas(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    ...  
}
```

34

## Valores retornados por métodos

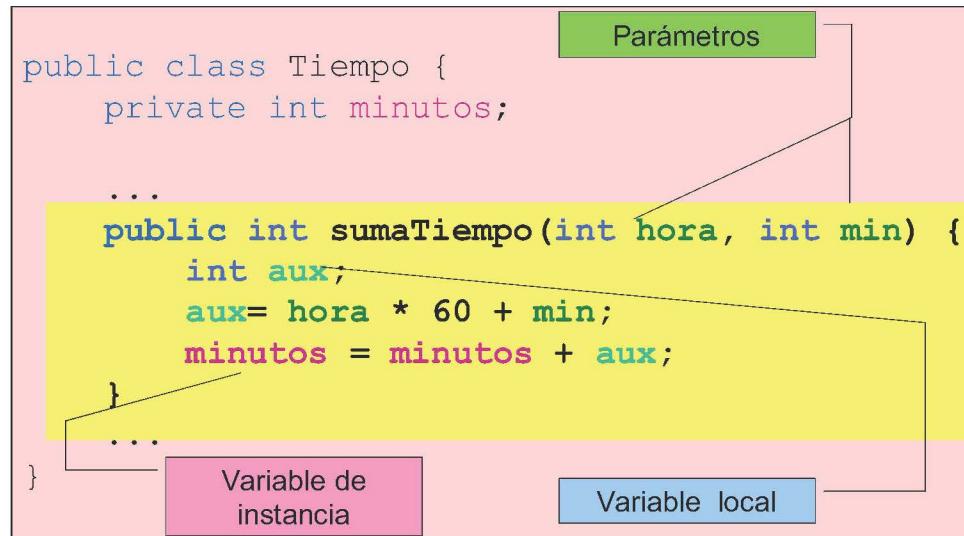
- ❖ Los métodos **pueden o no** retornar un valor.
- ❖ Modificadores de tipo de valor retornado:
  - Una referencia a una clase: `String`, `CuentaCorriente`, `Pez`, etc.
  - `void`: el método no retorna valor.
  - Un tipo primitivo: `short`, `byte`, `int`, `long`, `char`, `boolean`.
  - Una referencia a un arreglo: `int[]`, `String[][]`, `Pez[]`, etc.
- ❖ Un método puede retornar como máximo un único valor.
- ❖ Para retornar valores se utiliza la instrucción `return`.  
Formato:  

return literal;	o	return variable;
-----------------	---	------------------
- ❖ El flujo de un método termina cuando se alcanza una instrucción `return`.
- ❖ Todos los flujos de un método **no void** deben terminar en un `return`. Un método void puede terminar su ejecución en una instrucción `return` sin valor.

35

## Variables locales en constructores y métodos

- ❖ **VARIABLES LOCALES:** variables declaradas dentro del cuerpo del constructor o método. Se crean y utilizan en cada ejecución del constructor o método. No existen fuera de él.



36

## Nombres de métodos

- ❖ Convención: los nombres de métodos se escriben con la primera letra en minúsculas, y la primera letra de las siguientes palabras en mayúsculas.

```
public class CajaAhorro {  
    private int saldo;  
    private int transacciones;  
    ...  
    public void depositarDinero( int monto ) {  
        if( monto < 0)  
            return;  
        saldo = saldo + monto;  
        transacciones++;  
    }  
    ...  
}
```

37

## Sobrecarga de métodos

- ❖ Los métodos se diferencian por **nombre del método**, y **cantidad, tipo y orden de sus parámetros**. Todo esto constituye la “**firma del método**” (*method signature*).
- ❖ El tipo de valor retornado **no forma parte** de la “firma del método” (no es utilizado para distinguir entre métodos).

```
public double sumaTiempo() { ...  
public double sumaTiempo(int a) { ...  
public double sumaTiempo(double a) { ...  
public double sumaTiempo(int a, double b) { ...  
public double sumaTiempo(double a, int b) { ...
```

38

## Instanciación de objetos

- ❖ Para **instanciar** un objeto se debe usar el operador **new**:

```
CuentaCorriente miCuenta;  
miCuenta = new CuentaCorriente(1000);
```

- ❖ Una variable definida como referencia a objetos de una clase puede contener la dirección **null**.
- ❖ Recordar que la sola instanciación de un arreglo que debe referenciar objetos de una cierta clase no instancia estos objetos. Los objetos referenciados deben instanciarse separadamente.

39

## Eliminación de objetos

- ❖ Cuando un objeto deja de ser referenciado, se vuelve **inaccesible**.
- ❖ El “recolector automático de basura” de Java (*automatic garbage collector*) lo destruye, liberando la memoria utilizada.

40

## Herencia

- ❖ Java sólo soporta la herencia simple.
- ❖ Toda clase en Java es subclase de otra clase. Las clases en que el programador no se especifica una superclase son automáticamente subclases de la clase **Object**.
- ❖ Para declarar una subclase:

```
public class NombreSubclase extends NombreSuperclase
```

41

## Herencia (cont.)

- ❖ Las variables y métodos privados no son heredados en las subclases.
- ❖ Los constructores no son heredados en las subclases, pero se puede hacer referencia a ellos usando **super**:

```
super( lista de parámetros );
```

- ❖ La instrucción **super** debe ser la primera dentro del constructor de la subclase. Si el programador no la indica, Java automáticamente agrega una referencia al constructor sin parámetros de la superclase:

```
super();
```

42

## Operador **instanceof**

- ❖ El operador **instanceof** permite reconocer la clase a la que pertenece un objeto referenciado desde una variable determinada.
- ❖ Formato:

**NombreVar instanceof NombreClase**

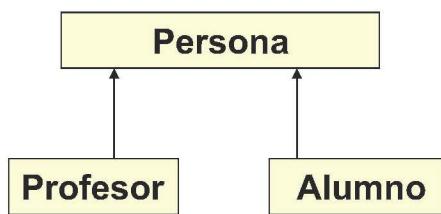
- ❖ Ejemplo:

```
if( pers instanceof Persona )  
    System.out.println( "La variable pers referencia a una Persona" );  
else  
    System.out.println( "La variable pers no referencia a una Persona" );
```

43

## Operador instanceof

❖ Suponer:



Persona p1 =null;

Profesor p2 = new Profesor();

Alumno p3 = new Alumno();

if( p1 instanceof Persona ) --> **false**  
if( p2 instanceof Profesor ) --> **true**  
if( p3 instanceof Alumno ) --> **true**  
if( p2 instanceof Alumno ) --> **false**  
if( p3 instanceof Persona ) --> **true**  
if( p2 instanceof Persona ) --> **true**

44

## Prerrequisito para continuar

❖ Revisar el siguiente vocabulario:

- Tipo de dato primitivo
- Clase, interface, clase abstracta
- Herencia, super-clases, sub-clases, asociación, composición
- Instancia de clase, objeto
- Constructor
- Método, método de instancia, método de clase
- Variable de instancia, variable estática, variable local
- Implementar una clase, implementar un método
- Sobrecarga de métodos
- Sobre escritura de métodos heredados

45

## Prerrequisito para continuar

- **REPASO:**

- Observar los métodos redefinidos: `test()`

```
public class Uno {  
    public int test() {return 1;}  
    public int result1() { return this.test();}  
}  
  
public class Dos extends Uno{  
    public int test() {return 2;}  
}  
  
public class Tres extends Dos {  
    public int result2() { return this.result1(); }  
    public int result3() { return super.test(); }  
}  
  
public class Cuatro extends Tres {  
    public int test() {return 4;}  
}
```



46

## Prerrequisito para continuar

- **REPASO:**

```
public class PruebaConstructores{  
    public static void main (String args[]){  
        Uno ob1 = new Uno();  
        Dos ob2 = new Dos();  
        Tres ob3 = new Tres();  
        Cuatro ob4 = new Cuatro();  
  
        System.out.println("ob1.test = "+ ob1.test());  
        System.out.println("ob1.result1 = " + ob1.result1());  
        System.out.println("ob2.test = " + ob2.test());  
        System.out.println("ob2.result1 = " + ob2.result1());  
        System.out.println("ob3.test = " + ob3.test());  
        System.out.println("ob4.result1 = " + ob4.result1());  
        System.out.println("ob3.result2 = " + ob3.result2());  
        System.out.println("ob4.result2 = " + ob4.result2());  
        System.out.println("ob3.result3 = " + ob3.result3());  
        System.out.println("ob4.result3 = " + ob4.result3());  
    }  
}
```



47

## Prerrequisito para continuar

- **CUESTIONES:**

	V	F
Una clase abstracta es una clase que no se puede instanciar		
Los métodos abstractos de las clases abstractas no tienen implementación		
En la declaración de una interfaz pueden aparecer implementación de métodos y atributos		
Para compilar correctamente una clase que implementa una interfaz, ésta debe contener todos los métodos declarados en la interfaz.		
Para compilar correctamente una clase que hereda de una clase abstracta, ésta debe contener todos los métodos abstractos que herede.		
Las interfaces encapsula información de los datos.		
Las interfaces deben definir los atributos como constantes.		
Una clase puede ser abstracta y no contener ningún método abstracto.		
Un tipo B es compatible con un tipo A sólo si la clase B es descendiente de la clase A.		