

UT3

Entendiendo la definición de una clase. Estructura de control condicional.

Módulo - Programación (1º)

Ciclos - Desarrollo de Aplicaciones Multiplataforma | Desarrollo de Aplicaciones Web

CI María Ana Sanz

Contenidos

- Atributos
 - concepto de variable
- Constructores
- Paso de parámetros
 - formales y actuales
- Sentencia de asignación
- Métodos
 - accesores y mutadores
- Sentencia de escritura
- El operador de concatenación +
- Variables locales
- Constantes
- Sentencia condicional
 - if
 - switch

Objetivos

- Aprender a escribir (codificar) la estructura de una clase
 - definir atributos
 - definir constructores
 - definir métodos
- Escribir los algoritmos que representan los métodos
 - sentencia de escritura
 - sentencia de asignación
 - sentencia condicional

Repaso de conceptos

- Clase
 - Plantilla para describir objetos
 - define atributos (estado) y métodos (comportamiento) para todos los objetos de esa clase
- Objeto
 - instancia particular de una clase
- Múltiples instancias de una clase
 - cada instancia su propio estado
 - todas se comportan igual
- Métodos
 - describen comportamiento de los objetos
 - pueden tener 0, uno o más parámetros
 - pueden devolver 0 o un valor

Atributos, constructores, métodos. Proyecto máquina expendedora.

- Máquina expendedora
 - modela el comportamiento de una máquina que expende tickets
 - el cliente inserta dinero y solicita la impresión del ticket
 - la máquina guarda la cantidad de dinero recogido de entre todos los tickets emitidos.
 - la máquina emite tickets de precio único.
- Ejer 3.1

máquinaE1 : MáquinaExpendedora

private int precio	50	Inspeccionar
private int importe	0	Obtener
private int total	0	

Mostrar campos estáticos Cerrar

Inicialmente al crear
el objeto

máquinaE1 : MáquinaExpendedora

private int precio	50	Inspeccionar
private int importe	50	Obtener
private int total	0	

Mostrar campos estáticos Cerrar

antes de emitir
el ticket

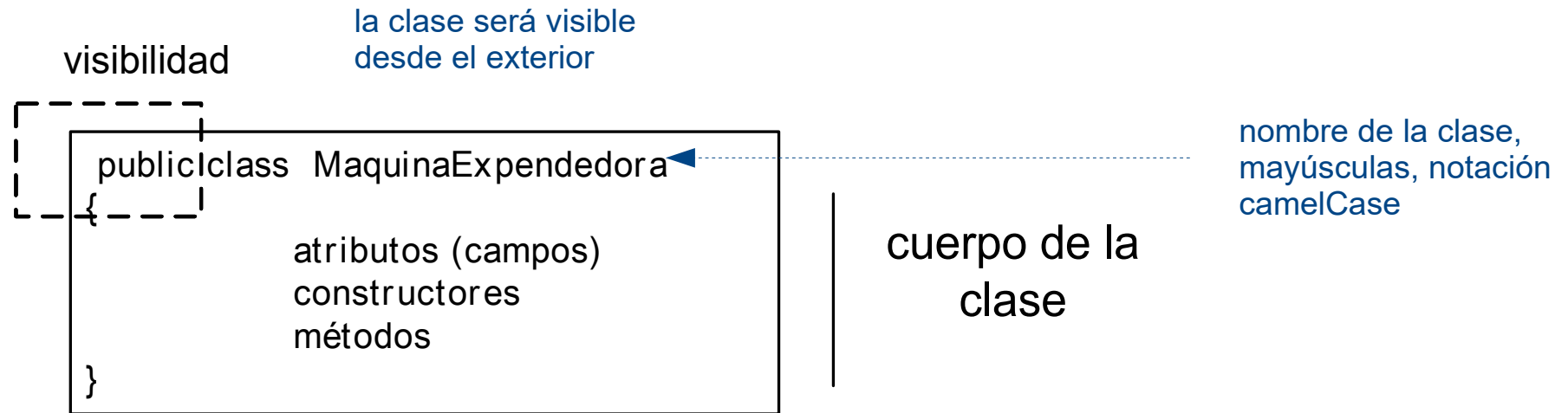
máquinaE1 : MáquinaExpendedora

private int precio	50	Inspeccionar
private int importe	0	Obtener
private int total	50	

Mostrar campos estáticos Cerrar

después de emitir el
ticket

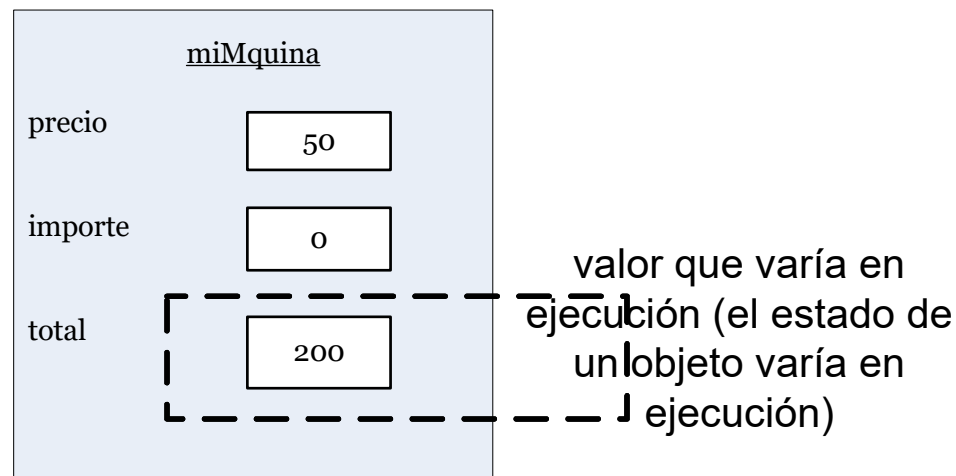
Definición de la clase MáquinaExpendedora



- Código legible (claro)
 - respetando indentaciones, blancos de separación, ..
 - `{ }` solas en una línea
- Primero atributos, luego constructor, después métodos (hay otras convenciones de escritura)

Atributos

- Almacenan el estado de un objeto
- Son las **variables de instancia**
- ¿Qué es una **variable**?
 - lugar de memoria en el que guardamos un valor de un tipo de datos que puede modificarse
 - en este caso la variable está dentro de un objeto (el objeto está en memoria)

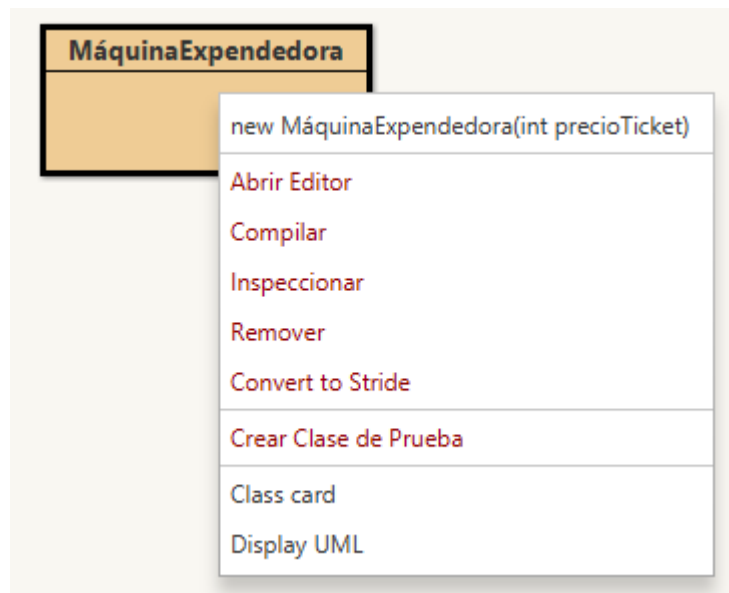


Atributos

- Por cada atributo
 - visibilidad (private – solo visible dentro de la clase)
 - tipo del atributo (int, double,)
 - nombre atributo (empieza en minúsculas - camelCase)
-
- Sentencias Java acaban en ; excepto
 - declaración de clase
 - declaración de métodos
 - líneas que contienen { }
 - **Comentarios** – aumentan la legibilidad
 - // comentario de una línea
 - /* varias líneas */
 - /** comentario javadoc */
 - el compilador los ignora

Constructores

- Método especial
 - crea un objeto (reserva memoria para él) a partir de la clase
 - inicializa correctamente sus atributos
- Tiene el mismo nombre que la clase
- no tiene valor de retorno (sin void)
- puede tener parámetros



Quando se llama al constructor se ejecutan las instrucciones que hay en el cuerpo del constructor. Se suelen incluir instrucciones que inicializan los atributos.

Constructores

```
public class MáquinaExpendedora
{
    // El precio de un ticket en esta máquina
    private int precio;
    // Cantidad de dinero introducida por el usuario hasta ahora
    private int importe;
    // Cantidad total de dinero recogida por la máquina
    private int total;

    /**
     * Crear una máquina que emite tickets de un determinado precio
     * El precio ha de ser mayor que 0 y no hay que verificar esto
     */

```

```
    public MáquinaExpendedora(int precioTicket)
    {
        precio = precioTicket;
        importe = 0;
        total = 0;
    }

    .....
    .....

```

visibilidad del constructor siempre public

Por defecto, Java inicializa los atributos (si no ponemos nada en el constructor, int a 0, String a null, boolean a false,)

Constructores. Ejemplos.

- Define una clase Circulo que modela círculos de un determinado radio. Incluye el constructor que crea los círculos y los inicializa con un valor de radio 5.7.
- Modifica el constructor para que el valor del radio se pase como parámetro al constructor

```
public class Circulo
{
    private double radio;
    public Circulo()
    {
        radio = 5.7;
    }
}
```

todos los círculos tendrán
radio = 5.7

```
public class Circulo
{
    private double radio;
    public Circulo(double queRadio)
    {
        radio = queRadio;
    }
}
```

más flexible

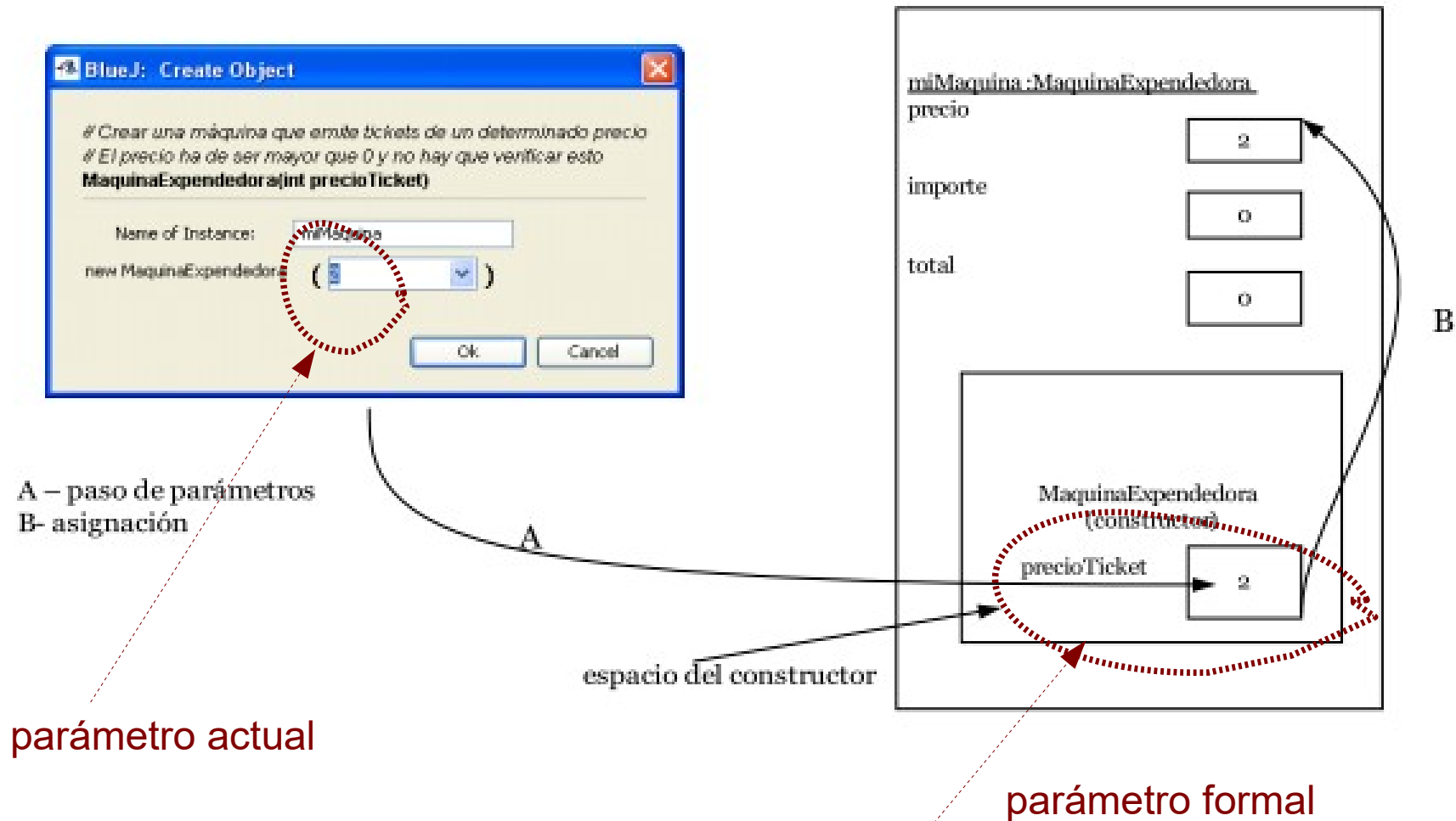
Paso de parámetros

- El constructor (y resto de métodos) pueden tener **parámetros**
 - a través de los parámetros proporcionamos desde fuera de la clase información para asignar a los atributos
- Se especifican en la cabecera del constructor

```
public MaquinaExpendedora(int precioTicket)
{
    .....
}
```

- **Parámetros formales**
 - en la cabecera del constructor (o del método)
 - disponibles solo en el cuerpo del constructor (o método)
 - visibilidad dentro del constructor (o método)
- **Parámetros actuales**
 - valores proporcionados desde fuera a los parámetros formales

Paso de parámetros. Parámetros formales y parámetros actuales.



Parámetros formales y parámetros actuales.

```
public class Punto  
{
```

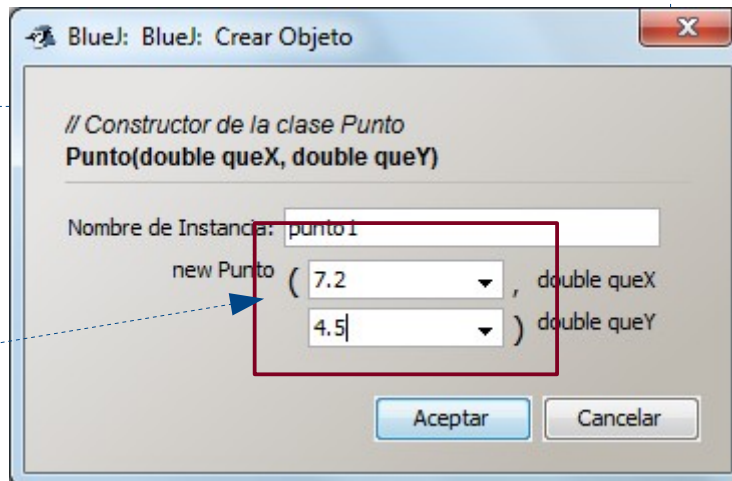
```
    private double x;  
    private double y;
```

```
    /**  
     * Constructor de la clase Punto  
     */  
    public Punto(double queX, double queY)  
    {  
        x = queX;  
        y = queY;  
    }
```

formales

2

actuales



punto1:Punto

4

x 7.2

y 4.5

```
public Punto(double queX, double queY)  
{  
    x = queX;  
    y = queY;  
}
```

3

Parámetros formales y parámetros actuales.

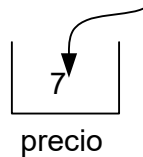
- La correspondencia entre parámetros formales y actuales se hace en nº , orden y tipo
- Parámetros formales -
 - variables locales al cuerpo del constructor (o método)
 - solo se conocen dentro del constructor
 - tiempo de vida de un parámetro formal se limita a lo que dure la llamada al constructor o método.
 - concluida su ejecución los parámetros formales desaparecen y sus valores se pierden.
- **Paso de parámetros en Java – por valor**
 - se copia el valor de los parámetros actuales en los formales (los actuales no de modifican)
- Ejer 3.2

Ejer 3.2

- `public Estudiante(String nombre)`
 - a la clase `Estudiante`
 - `nombre` es un parámetro formal
 - `new Estudiante("Pedro")` posible llamada al constructor
- `public Libro(String titulo, double precio)`
 - 2 parámetros actuales
 - `String` y `double` (puede ser `int`)
 - `new Libro("La colmena", 89.75)`

Sentencia de asignación

- Permite almacenar un valor en una variable (atributo o variable local)
- Con el operador = **variable = expresión;**
 - `precio = precioTicket;`
 - `precio = 7;`
 - `precio = 2 * precioTicket;`
 - `x = queX;`
- Tipo de la expresión coincidente con tipo de variable que la recibe
 - válido también para parámetros



Java permite declarar y asignar, no lo haremos con los atributos ni con parámetros formales, sí con variables locales por comodidad

- Ejer 3.3 a 3.7

Sentencia de asignación

- **Conversiones implícitas**

- Java hace una conversión de un tipo de menor a otro de mayor precisión (en asignaciones, correspondencias de parámetros)
 - `private double precio;`
`precio = 7; // conversión implícita`

- A la inversa no es posible, hay que hacer una **conversión explícita** (*type casting*) (no, de momento)

- `private int precio;`
`precio = (int) 7.99; // conversión explícita`

- En el cuerpo de un constructor (método)

- declaración de variables locales
- sentencias ejecutables
 - asignación
 - if / switch
 - while / for
 - llamadas a métodos
 - creación de objetos

Ejer 3.3 a 3.7 (Sol.)

- Ejer 3.3

```
public class Mascota
{
    private String nombre;
    public Mascota( String nuevoNombre )
    {
        nombre = nuevoNombre;
    }
}
( new Mascota("Pancho") - llamada al constructor )
```

- Ejer 3.4 (probar en el ordenador)

- Ejer 3.5

```
radio = 6;
perimetro = 2 * Math.PI * radio;
area = Math.PI * radio * radio;
```

- Ejer 3.6

```
valorCompra = 653.36;
totalFactura = valorCompra + valorCompra * 0.16;
```

Ejer 3.3 a 3.8 (Sol.)

- Ejer 3.7

```
int edad;  
boolean esAdulto;  
  
esAdulto = (edad >= 18);  
  
.....  
  
char teclaPulsada;  
boolean teclaCorrecta;  
  
teclaCorrecta = teclaPulsada == 'S' ||  
                teclaPulsada == 's' ||  
                teclaPulsada == 'N' ||  
                teclaPulsada == 'n';
```

Ejer 3.3 a 3.8 (Sol.)

- Ejer 3.8
(probar
en el
Codepad)

Declaración	Asignación	Correcto / No correcto
int precio;	precio = 35.0;	No correcto (requiere conversión explícita)
double precio;	precio = 6;	Correcto (se hace conversión implícita)
int texto;	texto = "hola";	No correcto (tipos no compatibles)
float precio;	precio = 34.56f;	Correcto
double tasa;	tasa = precio * 0.07;	Correcto
int pago;	pago = 50;	Correcto
double cambio;	cambio = pago – precio – tasa;	Correcto

Ejemplo para hacer

- Crea un proyecto Bombilla
 - Define una clase Bombilla que modela una bombilla
 - Toda bombilla posee una potencia (diferente para cada bombilla) y tiene una situación, encendida o apagada
 - Define los atributos de la clase Bombilla
 - Define el constructor de tal forma que toda bombilla se crea con una determinada potencia (no siempre la misma) y situación inicial apagada.
 - Crea varios objetos Bombilla en el *Object Bench*
 - Inspecciona el estado de las diferentes bombillas
 - Comentario de principio de clase y en el constructor

Métodos

- Implementan el comportamiento de los objetos
 - lo que el objeto puede hacer
 - lo que se puede pedir al objeto (servicios que proporciona)
- Incluye el código que se ejecuta (las instrucciones) cuando se envía un mensaje al objeto.
- Una clase contiene multitud de métodos
 - cada uno de ellos debe realizar una tarea clara y precisa
 - ejecutar un método es realizar cada una de las sentencias (instrucciones) que contiene, una detrás de otra, en el orden en que aparecen.

```
/**  
 * Mover el triángulo horizontalmente  
 */  
public void moverHorizontal(int distancia)  
{  
    borrar();  
    xPosicion += distancia;  
    dibujar();  
}
```

Estructura de un método

- **Cabecera** – signature del método
 - **Cuerpo** – entre { }
 - declaración de variables locales (si las hay)
 - instrucciones (parte ejecutable)
- Cualquier conjunto de declaraciones y sentencias dentro de un par { } se denomina **bloque**
- El cuerpo de un método es un bloque
 - El cuerpo de una clase también

```
public int getPrecio()  
{  
    // declaraciones de variables locales  
  
    // sentencias  
}
```

```
/**  
 * Recibir una cantidad de dinero de un  
 usuario  
 */  
public void insertarDinero(int cantidad)  
{  
    importe = importe + cantidad;  
}
```


Valor de retorno de un método

- Un método puede tener un **tipo de retorno**.
 - `public int getPrecio()`
- Si no devuelve nada se especifica el tipo `void`.
 - `public void imprimirTicket()`
- Sentencia **return** (**return expresión;**)
 - para devolver un valor el método
 - finaliza la ejecución del método
 - última instrucción del método (habitualmente)
 - Tipo de valor devuelto en la sentencia `return` coincidente con tipo indicado en la signatura

```
public double calcularArea()  
{  
    .....  
    return area; // ha de ser de tipo double  
}
```

Valor de retorno de un método

```
public class Circulo
{
    private double radio;
    .....
    public double calcularArea()
    {
        return Math.PI * radio * radio;
    }
}
```

Métodos accesorios

- Los métodos entran en alguna de las siguientes categorías:
 - métodos accesorios
 - métodos mutadores
 - otros métodos
- Métodos **accesorios** (*getters*)
 - proporcionan información sobre el estado de un objeto
 - ¿cuál es el precio de un ticket? ¿de qué color es el coche?
 - ¿está encendida la bombilla?
 - usualmente contienen una sentencia return que devuelve el valor de uno de los atributos del objeto
 - si el método escribe información sobre el objeto también puede considerarse un accesor

Métodos accesorios

```
public int getPrecio()  
{  
    return precio;  
}
```

```
public String getColor()  
{  
    return color;  
}
```

- Ejer 3.9
- Añade al proyecto Bombilla
 - accesor para la potencia y accesor para conocer la situación de la bombilla
- Crea un proyecto Circulo
 - añade la clase Circulo tal cómo la hemos hecho hasta ahora
 - un atributo radio y constructor con parámetro
 - incluye accesor para el radio
 - añade el método *calcularArea()*
 - añade el método *calcularPerimetro()*

Métodos mutadores

- Métodos **mutadores** (*setters*)
 - Modifican el estado interno del objeto
 - cambian el valor de uno o varios de sus atributos
 - Habitualmente contienen sentencias de asignación y reciben parámetros (no siempre tienen parámetros)

```
public void insertarDinero(int cantidad)
{
    importe = importe + cantidad;
}
```

```
public void setColor(String nuevoColor)
{
    color = nuevoColor;
}
```

acumular (importe += cantidad;)

- `variable = variable + expresión;` `(+= -= *= /= %=)`
 - **variable += expresión;**
- `variable = variable - expresión;`
 - **variable -= expresión;**
- Ejer 3.10 a 3.12

Ejer 3.10 a 3.12 (Sol.)

```
public void setPrecio(double quePrecio)
{
    precio = quePrecio;
}
```

Ejer 3.10

```
public void incrementar(int puntos)
{
    puntuacion += puntos;
}
```

Ejer 3.11

```
public void descontar(int cantidad)
{
    total -= cantidad;
}
```

Ejer 3.12

- Añade un mutador *cambiarSituacion()* a la clase Bombilla
 - enciende / apaga la bombilla (pista – operador lógico)
 - `situacion = !(situacion);`
- Añade un mutador *setRadio()* a la clase Circulo

Ejemplos para hacer

- Crea un proyecto TrianguloRectangulo y añade una clase con el mismo nombre
- La clase modela triángulos que tienen dos catetos
- Define el constructor con dos parámetros que representan los valores iniciales de los catetos
- Incluye accesores y mutadores para cada cateto
- Añade un método *obtenerHipotenusa()* que devuelve el valor de la hipotenusa

Escritura dentro de los métodos

- **System.out.println(parámetro_string)**
 - escribe el parámetro especificado, un String, en la ventana de texto (terminal) y efectúa al final un salto de línea
 - **println()** es un método del objeto *System.out* de Java.
 - **System.out.println("# Máquina expendedora BlueJ");**
 - con **print()** si no queremos salto de línea al final

Escritura dentro de los métodos

```
public void imprimirTicket()
{
    // Simula impresión de un billete

    System.out.println("#####");
    System.out.println("# Máquina expendedora BlueJ");
    System.out.println("# Billete:");
    System.out.println("# " + precio + " cents.");
    System.out.println("#####");
    System.out.println();
    // Actualizar el total recogido por la máquina
    total = total + importe;
    // Poner el importe a 0
    importe = 0;
}
```

+ operador concatenación de cadenas

- `System.out.println("# " + precio + " cents.");`
 - obtiene una nueva cadena resultado de concatenar los tres valores especificados “# 20 cents.”
 - **Probar en el CodePad** (*Bloc de código*)
 - `System.out.println("Total: " + 50 + 33); // Total: 5033`
 - primero se evalúa `Total + 50`
 - `System.out.println("Total: " + (50 + 33)); //Total: 83`
 - `System.out.println(50 + 33 + "ejemplo"); / 83ejemplo`
 - `System.out.println(3 + 9); //12`
 - `System.out.println("" + 3 + 9); //39`
 - `System.out.println(); //línea en blanco`
 - `System.out.print(); //no salta línea después de escribir`
- Ejer 3.13 y 3.14

Ejer 3.13 y 3.14 (Sol.)

Ejer 3.13

```
// no es accesor ni mutador
public void prompt()
{
    System.out.println("Por favor, inserte la cantidad correcta de dinero");
}

// es accesor
public void mostrarPrecio()
{
    System.out.println("Precio total del ticket " + precio + "ctmos.");
}
```

```
// accesor para nº puertas
```

```
public int getPuertas()
{
    return numeroPuertas;
}

public void setPuertas(int cuantasPuertas)
{
    numeroPuertas = cuantasPuertas;
}
```

```
public void saludar()
{
    System.out.println("Hola, me
llamo " + nombre + " y tengo " + años + " años");
}
```

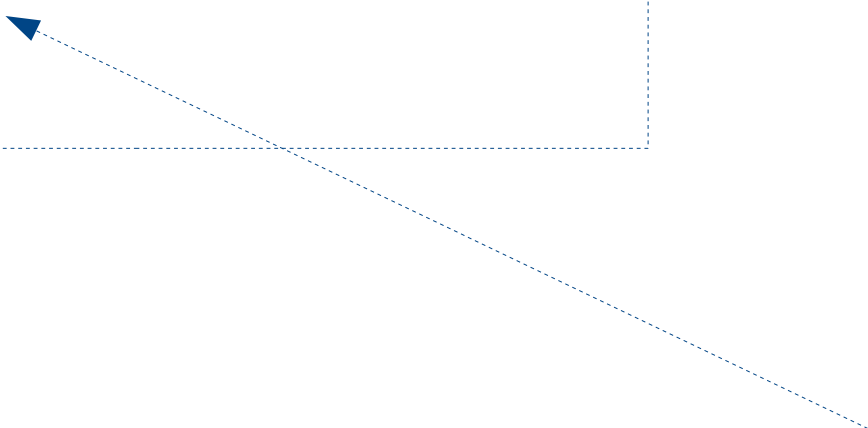
Ejer 3.14

Variables locales

- Variables
 - **de instancia** – atributos
 - **parámetros formales** – locales al método/constructor
 - **locales**
- **Variables locales**
 - se declaran dentro del método / constructor
 - no se especifica visibilidad (no *private*)
 - ámbito limitado al método en que están definidas
 - *tiempo de vida* – lo que dura la ejecución del método
 - métodos y constructores pueden incluir variables locales
 - se declaran indicando tipo y nombre
 - `int total = 0;`
 - `String str = "";`
 - `int suma = 0 // declarar e inicializar`

Variables locales

```
/**
 * Calcula y devuelve el área
 */
public double calcularArea()
{
    double area = Math.PI * radio * radio;
    return area;
}
```



Java por defecto asigna valores a los atributos en el constructor. A las variables locales no, hay que hacer explícita la declaración (el compilador nos avisa si no lo hacemos)

variable local

▪ Ejer 3.15

Ejer 3.15 (Sol.)

```
/**
 *
 * Clase que intercambia el valor de sus atributos
 * @author
 * @version
 */
public class Ejemplo
{
    private int a;
    private int b;

    /**
     * Constructor
     */
    public Ejemplo(int queA, int queB)
    {
        a = queA;
        b = queB;
    }

    /**
     *
     * Intercambiar a y b
     *
     */
    public void intercambiar()
    {
        int aux = a;
        a = b;
        b = aux;
    }
}
```

```
/**
 * Mensaje antes del intercambio
 */
public void promptAntes()
{
    System.out.println("Antes del intercambio");
}

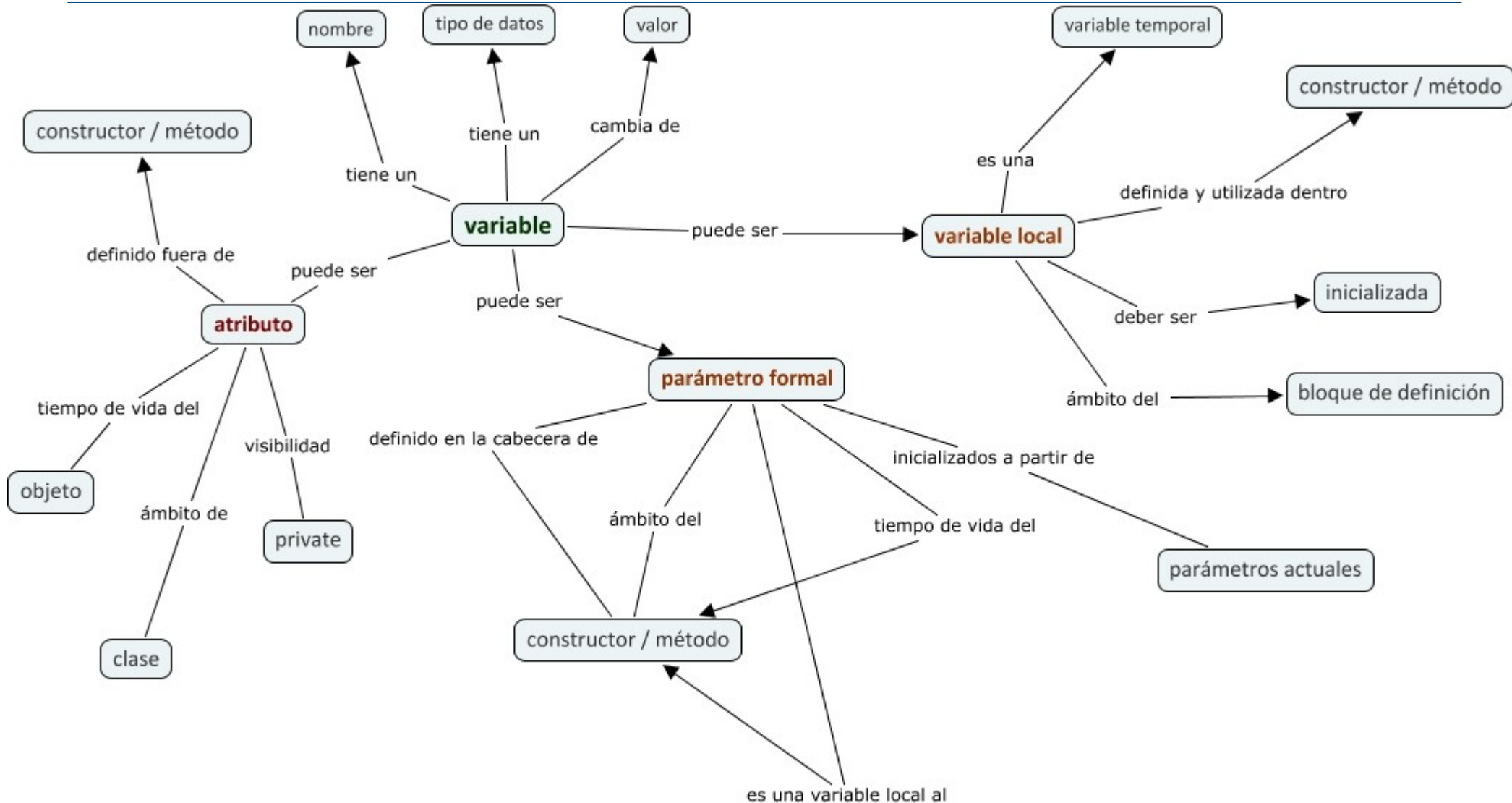
/**
 * Mensaje después del intercambio
 */
public void promptDespues()
{
    System.out.println("Después del intercambio");
}

/**
 * Mostrar valor de los atributos
 */
public void escribir()
{
    System.out.println("Atributo a = " + a + "\nAtributo b = " + b);
}
}
```

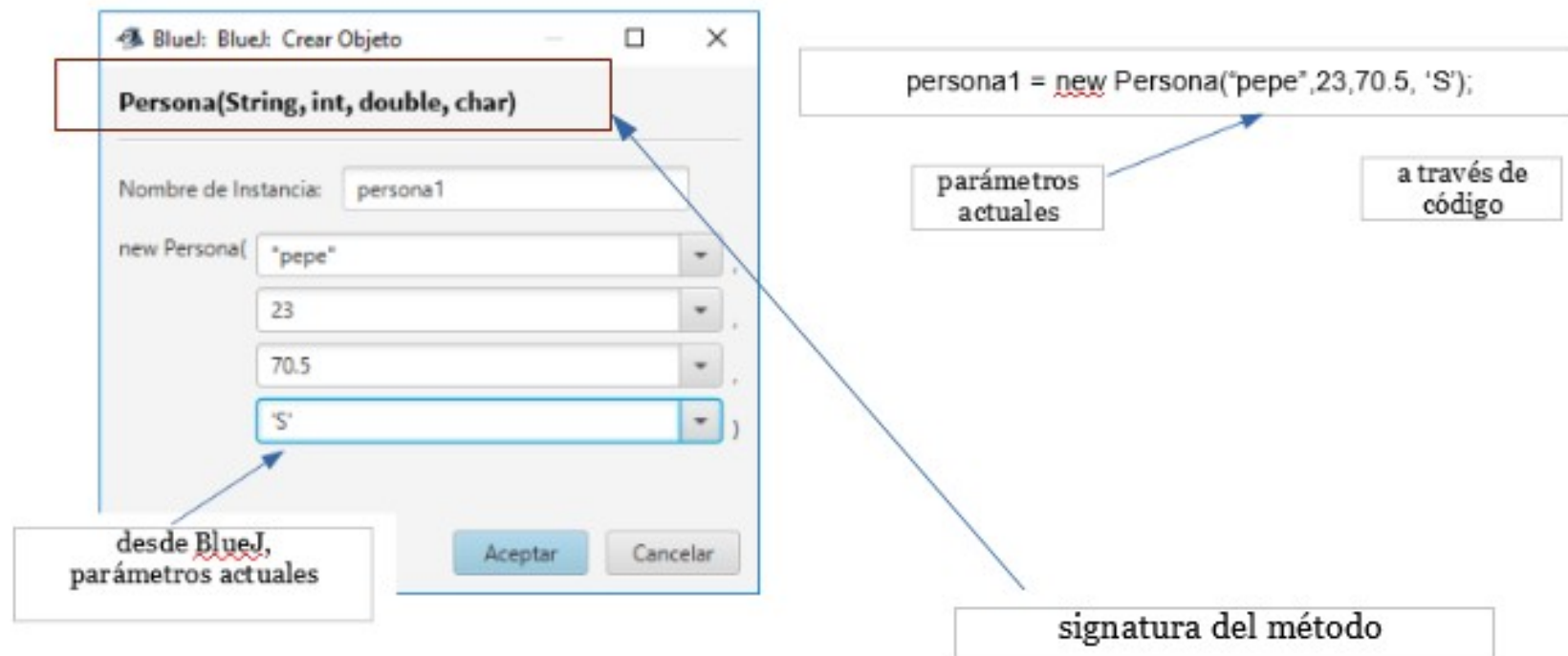
Constantes en Java

- Variable
 - valor que puede cambiar a lo largo de la ejecución de un programa
- **Constante**
 - valor que nunca cambia
- Declarar una constante en Java:
 - ***final tipo nombre_constante = valor;***
 - nombres en mayúsculas
 - definidas antes que los atributos (convención)
 - dan legibilidad (evitan *números mágicos*)
- **final double PI = 3.1416;** (si es local)
- **private final double IVA = 0.16;** (si es un atributo)

Resumen de variables



Resumen de variables



Cuestionario UT3

- Cuestiones UT3
 - Asociar definiciones y términos
- Ejercicios
 - AD01 Libro
 - AD02Hucha
 - AD03 Contador
 - AD04FacturaLuz
 - AD05 ConversorFahrenheit
 - AD06 Descomposición en monedas

AD 06 Descomposición en monedas

```
public class Dinero
{
    //constantes
    private final int DIEZ = 10;
    private final int CINCO = 5;
    private final int DOS = 2;
    private final int UNO = 1;
    //asumimos que la cantidad es <=100€
    private int euros;
    . . . . .
}
```

96

9 de 10

1 de 5

0 de 2

1 de 1

AD 06 Descomposición en monedas

```
/**
 * Muestra en pantalla la descomposición
 * mínima de monedas que representa la
 * cantidad guardada
 */
public void printDescomposicionMonedas()
{
    int auxDinero = euros; // guardamos en una variable
    // el valor del atributo para no perderlo

    int diez = auxDinero / DIEZ;
    auxDinero = auxDinero % DIEZ;
    int cinco = auxDinero / CINCO;
    auxDinero = auxDinero % CINCO;
    int dos = auxDinero / DOS;
    auxDinero = auxDinero % DOS;
    System.out.println(euros + " € son\n" +
        "Billetes de " + DIEZ + " = " + diez +
        "\nBilletes de " + CINCO + " = " + cinco +
        "\nMonedas de " + DOS + " = " + dos +
        "\nMonedas de " + UNO + " = " + auxDinero);
}
```

Estructuras de control condicional

- Problemas con nuestra MaquinaExpendedora
 - no verifica que las cantidades de dinero introducidas sean positivas
 - ¿qué pasa si hacemos *insertarDinero(-20)* ?
 - no verifica si se ha introducido suficiente dinero para emitir un ticket
 - no devuelve ningún cambio

máquinaE1 : MáquinaExpendedora

private int precio	30	Inspeccionar	
private int importe	20		Obtener
private int total	0		

< >

Mostrar campos estáticos

Cerrar

máquinaE1 : MáquinaExpendedora

private int precio	30	Inspeccionar	
private int importe	0		Obtener
private int total	20		

Mostrar campos estáticos

Cerrar

- Vamos a mejorarla
 - Abrimos el proyecto MaquinaExpendedoraMejorada

Estructuras de control condicionales

```
public class MáquinaExpendedora
{
    .....
    /**
     * Recibir una cantidad de dinero de un usuario
     * Verificar que la cantidad es positiva
     */

    public void insertarDinero(int cantidad)
    {
        if (cantidad > 0)
        {
            importe = importe + cantidad;
        }
        else
        {
            System.out.println("Introduzca una cantidad positiva: " + cantidad);
        }
    }
}
```

alterar flujo de ejecución
secuencial

Estructuras de control condicionales

```
public void imprimirTicket() {  
    if (importe >= precio) {  
        // Simula impresión de un billete  
        System.out.println("#####");  
        System.out.println("# Máquina expendedora BlueJ");  
        System.out.println("# Billeto:");  
        System.out.println("# " + precio + " cents.");  
        System.out.println("#####");  
        System.out.println();  
        // Actualizar el total recogido por la máquina con el precio  
        total = total + precio;  
        // decrementar el importe con el precio  
        importe = importe - precio;  
    }  
    else {  
        System.out.println("# Debe insertar al menos: " +  
            (precio - importe) + " céntimos más ");  
    }  
}
```

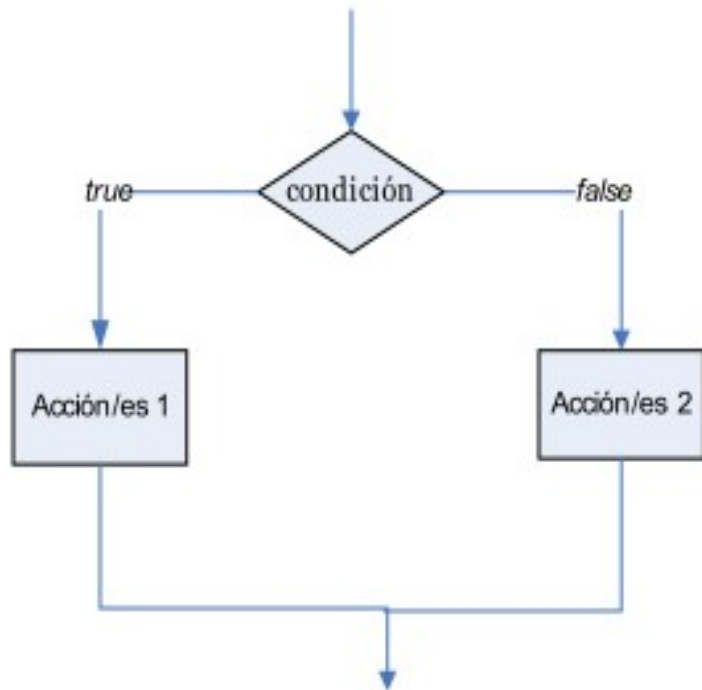
Estructuras de control condicionales

```
/**  
 * Devolver el dinero del importe y poner el importe a 0  
 */  
  
public int devolverCambio()  
{  
    int cambio = importe;  
    importe = 0;  
    return cambio;  
}  
}
```


Estructuras de control

- Alteramos el flujo de ejecución secuencial de las instrucciones de un método
 - con las **estructuras de control**
 - **condicionales** – se hacen unas u otras instrucciones dependiendo de una condición (if / switch)
 - **repetitivas** – ejecutan un conjunto de instrucciones repetidamente un nº determinado o indeterminado de veces (while / for)

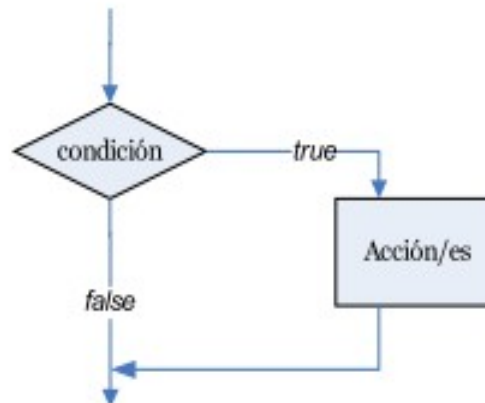
Sentencia if



Alternativa doble

```
if (condición)
{
    acción/es 1
}
else
{
    acción/es 2
}
```

Condición - expresión booleana que se evalúa. Si es *true* se realizan acción/es1. Si es *false* se realizan acción/es2.



Alternativa simple

```
if (condición)
{
    acción/es
}
```

Sentencia if

```
/**
 * Recibir una cantidad de dinero de un usuario
 * Verificar que la cantidad es positiva
 */
public void insertarDinero(int cantidad)
{
    if (cantidad > 0)
    {
        importe = importe + cantidad;
    }
    else
    {
        System.out.println("Introduzca una cantidad positiva: " + cantidad);
    }
}
```

Sentencia if

```
public void imprimirTicket()
{
    if (importe >= precio)
    {
        // Simula impresión de un billete
        System.out.println("#####");
        System.out.println("# Máquina expendedora BlueJ");
        System.out.println("# Billete:");
        System.out.println("# " + precio + " cents.");
        System.out.println("#####");
        System.out.println();
        // Actualizar el total recogido por la máquina con el precio
        total = total + precio;
        // decrementar el importe con el precio
        importe = importe - precio;
    }
    else
    {
        System.out.println("# Debe insertar al menos: " +
            (precio - importe) + " céntimos más ");
    }
}
```

Consideraciones sobre la sentencia if

- Si el conjunto de acciones a ejecutar es una única sentencia puede omitirse las { } del bloque.
 - nosotros siempre las pondremos
- Acción/es pueden ser cualquier instrucción válida incluso otra sentencia if. Podemos construir en este último caso *if anidados*.
- Cuando hay if anidados, la parte else siempre se corresponde con el último if abierto dentro del mismo bloque.

Consideraciones sobre la sentencia if

```
public void printDescripcion()  
{  
    if (edad < 13) {  
        System.out.println(nombre + " es un niño");  
    }  
    else {  
        if (edad < 18) {  
            System.out.println(nombre + " es un adolescente");  
        }  
        else {  
            System.out.println(nombre + " es un adulto");  
        }  
    }  
}
```

Consideraciones sobre la sentencia if

```
public void printDescripcionOtraVersion()
{
    if (edad < 13) {
        System.out.println(nombre + " es un niño");
    }
    else if (edad < 18) {
        System.out.println(nombre + " es un adolescente ");
    }
    else {
        System.out.println(nombre + " es un adulto");
    }
}
```

Ejemplos sentencia if

```
public boolean esPositivoPar(int numero)
{
    if ( (numero > 0) && (numero % 2 == 0)) {
        return true;
    }
    return false;
}
```

```
public boolean esPositivoPar(int numero)
{
    int resul;
    if (numero > 0 && numero % 2 == 0) {
        resul = true;
    }
    else {
        resul = false;
    }
    return resul;
}
```


Ejemplos sentencia if

```
public boolean metodoMisterio(int valor)
{
    if (valor >= 0) {
        return true;
    }
    return false;
}
```

```
public boolean metodoMisterio(int valor)
{
    return (valor >= 0);
}
```

```
public boolean metodoMisterio(int valor)
{
    boolean resul = false;
    if (valor >= 0) {
        resul = true;
    }
    return resul;
}
```

Ejemplos sentencia if

- Completar clase EjemplosCondicional
- Ejer 3.16 a 3.19
- Acabar cuestionario

Sentencia condicional switch

```
switch (expresión)
{
    case valor1:  sentencias1;
                  break;
    case valor2:  sentencias2;
                  break;
    case valor3:  sentencias3;
                  break;
    .....
    default:      sentenciasn+1;
                  break;
}
```

```
switch (expresión)
{
    case valor1:
    case valor2:
    case valor3:  sentencias123;
                  break;
    case valor4:
    case valor5:  sentencias45;
                  break;
    .....
    default:      sentenciasn+1;
                  break;
}
```

expresion se evalúa a int, byte, short, char (a partir de la versión Java 7 también String)

case XX -XX es un valor constante, no una variable

Sentencia condicional switch

```
switch (expresión)
{
    case valor1: sentencias1;
                break;
    case valor2: sentencias2;
                break;
    case valor3: sentencias3;
                break;
    .....
    default:    sentenciasn+1;
                break;
}
```

```
switch (expresión)
{
    case valor1:
    case valor2:
    case valor3: sentencias123;
                break;
    case valor4:
    case valor5: sentencias45;
                break;
    .....
    default:    sentenciasn+1;
                break;
}
```

expresion se evalúa a int, byte, short, char (a partir de la versión Java 7 también String)

case XX -XX es un valor constante, no una variable

Consideraciones sobre la sentencia *switch*

- La sentencia *switch* consta de una expresión que se evalúa a un valor entero (*int*, *byte*, *short* o *char*) .
 - A partir de Java 7 también *String*
- Incluye una serie de etiquetas *case* con un valor constante
 - cada uno de los valores posibles que se pueden obtener al evaluar la expresión
- La expresión se evalúa y se ejecutan las sentencias asociadas a la etiqueta *case* que corresponde con el valor obtenido.
- La sentencia *break* finaliza la ejecución de la instrucción *switch*
- Si ningún valor de las etiquetas *case* se corresponde con el resultado de la expresión, se ejecutan las sentencias correspondientes a la parte *default*. Esta parte es opcional.

Ejemplos sentencia switch

```
switch (dia) {  
    case 1: nombreDia = "Lunes";  
            break;  
    case 2: nombreDia = "Martes";  
            break;  
    case 3: nombreDia = "Miércoles";  
            break;  
    case 4: nombreDia = "Jueves";  
            break;  
    case 5: nombreDia = "Viernes";  
            break;  
    case 6: nombreDia = "Sábado";  
            break;  
    case 7: nombreDia = "Domingo";  
            break;  
    default: nombreDia = "Incorrecto"  
            break;  
}
```

```
switch (dia) {  
    case 1:  
    case 2:  
    case 3:  
    case 4:  
    case 5: nombreDia = "Dia laborable";  
            break;  
    case 6:  
    case 7:  
            nombreDia = "Día no laborable";  
            break;  
    default: nombreDia = "Incorrecto"  
            break;  
}
```

Ejemplos sentencia switch Java 7

```
public void demoSwitchJava7(String nombreDia)
{
    switch (nombreDia) {
        case "lunes":
        case "martes":
        case "miercoles":
        case "jueves":
        case "viernes":
            System.out.println("Laborable");
            break;
        case "sabado":
        case "domingo":
            System.out.println("No laborable");
            break;
    }
}
```

Ejemplos sentencia switch

- Escribe el siguiente método, a) con if b) con switch

- **public String notaToString(int nota)**
 - devuelve un String
 - 1- NP / 2,3,4 – INS / 5 –
SUF / 6 – B / 7,8 – NOT / 9,10 SB

```
public String notaToStringConIf(int nota)
{
    String strNota = "";
    if (nota == 1) {
        strNota = "NP";
    }
    else if (nota < 5) {
        strNota = "INS";
    }
    else if (nota == 5) {
        strNota = "SUF";
    }
    else if (nota == 6) {
        strNota = "B";
    }
    else if (nota <= 8) {
        strNota = "NOT";
    }
    else {
        strNota = "SB";
    }
    return strNota;
}
```

```
public String notaToStringConSwitch(int nota)
{
    String strNota = "";
    switch (nota) {
        case 1: strNota = "NP";
                break;
        case 2:
        case 3:
        case 4: strNota = "INS";
                break;
        case 5: strNota = "SUF";
                break;
        case 6: strNota = "B";
                break;
        case 7:
        case 8: strNota = "SUF";
                break;
        default: strNota = "SB";
    }
    return strNota;
}
```


Ejer 3.20 Con if (Sol.)

- Ejer 3.20

```
public int calcularDiasMesConIf(int mes, int año)
{
    int diasMes;
    if (mes < 1 || mes > 12) // mes incorrecto
    {
        diasMes = -1;
    }
    else if (mes == 1 | mes == 3 || mes == 5 ||
             mes == 7 || mes == 8 || mes == 10 || mes == 12)
    {
        diasMes = 31;
    }
    else if (mes == 4 | mes == 6 || mes == 9 || mes == 11)
    {
        diasMes = 30;
    }
    else if (año % 4 == 0) // mes febrero, ver si año es bisiesto
    {
        diasMes = 29;
    }
    else
    {
        diasMes = 28;
    }
    return diasMes;
}
```

Ejer 3.20 con switch (Sol.)

■ Ejer 3.20

```
public int calcularDiasMesConSwitch(int mes, int año)
{
    int diasMes;
    if (mes < 1 || mes > 12) { // mes incorrecto
        diasMes = -1;
    }
    else {
        switch (mes)
        {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12: diasMes = 31;
            break;
            case 4:
            case 6:
            case 9:
            case 11: diasMes = 30;
            break;
            default:
                if (año % 4 == 0) { // es bisiesto
                    diasMes = 29;
                }
                else {
                    diasMes = 28;
                }
                break;
        }
    }
    return diasMes;
}
```

Ejercicios adicionales

- EJAD07 Ordenador
- EJAD08 Calentador
- EJAD09 Personaje
- EJAD10 Cafetera
- EJAD11 Calculadora
- EJAD12 Hora
- EJAD13 Fecha
- EJAD14 Fecha (Cont...)
- EJAD15 TresNumeros
- EJAD16 CartaBaraja

Ejercicios – Comentarios - toString()

- método **toString()**
 - método de la clase Object
 - sin argumentos y valor de retorno String
 - **public String toString()**
 - lo redefiniremos dando nuestra propia implementación
 - representación textual del objeto
 - devuelve una cadena que contiene información de interés sobre el objeto
 - **return “Nombre: “ + nombre + “\nEdad: “ + edad;** o también
 - **String strResul = “Nombre: “ + nombre + “\nEdad: “ + edad;**
return strResul;
 - conviene que toda clase incluya uno

Ejercicios - Comentarios - `Math.pow()` / `Math.random()`

- clase **Math**
 - contiene métodos que efectúan operaciones matemáticas habituales
- **`public static double pow(double x, double y);`**
 - calcula x^y
 - `double potencia = Math.pow(4, 3);` // probar en el CodePad
- **`public static double random()`**
 - **`Math.random()`** - genera un aleatorio $0 \leq n^o < 1.0$
 - **`Math.random() * 10`** - genera un aleatorio $0 \leq n^o < 10.0$
 - **`(int) (Math.random() * 10)`** - genera un aleatorio $0 \leq n^o < 10$
 - **`(int) (Math.random() * 10) + 1`** - genera un aleatorio $1 \leq n^o \leq 10$

Ejercicios – Comentarios – `Math.random()`

- **public static double random()**
 - **`(int) (Math.random() * 50) + 50`**
 - genera un aleatorio $50 \leq n \leq 99$ ($50 \leq n < 100$) [50, 100)
 - min: 50 max:100

- **`(int) (Math.random() * (max – min + 1)) + min`**
 - aleatorio entre [min, max]
- **`(int) (Math.random() * (max – min)) + min`**
 - aleatorio entre [min, max)

Ejercicios – Comentarios – `Math.random()`

- `public static double random()`
 - `int dado = (int) (Math.random() * 6) + 1;`
`//simular lanzamiento dado`
 - `(int) (Math.random() * 2)`
 - Simular lanzamiento moneda

Ejercicios – Comentarios – `Math.random()`

- **`import static java.lang.Math.pow;`**
 - así puedo poner en el código `pow(2, 3)` y no `Math.pow(2, 3)`
-

Otros métodos de la clase Math

Math.round(x)

Redondea el valor real x al entero más cercano (arriba o abajo). Lo devuelve como long.

Ej. Math.round(2.3) devuelve 2 / Math.round(-1.87) devuelve -2
Math.round(2.7) devuelve 3 / Math.round(2.5) devuelve 3
Math.round(-1.5) devuelve -1

Math.ceil(x)

Redondea el valor real x al entero más cercano siempre mayor (redondea hacia arriba). Lo devuelve como double.

Ej. Math.ceil(2.3) devuelve 3 / Math.ceil(-1.87) devuelve -1
Math.ceil(2.7) devuelve 3 / Math.ceil(2.5) devuelve 3
Math.ceil(-2.5) devuelve -2

Math.floor(x)

Redondea el valor real x al entero más cercano siempre menor (redondea hacia abajo). Lo devuelve como double.

Ej. Math.floor(2.3) devuelve 2 / Math.floor(-1.87) devuelve -2
Math.floor(2.7) devuelve 2 / Math.floor(2.5) devuelve 2
Math.floor(-2.5) devuelve -3

Otros métodos de la clase Math

