UT3 Entendiendo la definición de una clase. Estructura de control condicional.

En la unidad de trabajo anterior aprendimos que:

- *una clase define los atributos y métodos de las instancias de la clase*
- *las instancias de la clase son los objetos*
- *una clase puede tener varias instancias cada una de las cuales tendrá su propio conjunto de valores para los atributos. Este conjunto de valores representa el estado de un objeto. Además los atributos tienen un determinado tipo*
- *los métodos describen el comportamiento de los objetos, lo que éstos pueden hacer*
- *un método puede tener cero, uno o varios parámetros y puede devolver o no un valor*

En este tema aprenderemos a escribir la estructura de una clase. Veremos cuáles son los elementos básicos al definir una clase: atributos (campos), constructores y métodos.

Comprobaremos también que los métodos incluyen sentencias que ejecutadas de forma secuencial realizan la tarea para la que fueron diseñados. Los métodos describen un algoritmo. Inicialmente estudiaremos la sentencia de asignación y la sentencia de escritura. Posteriormente utilizaremos la sentencia de control condicional.

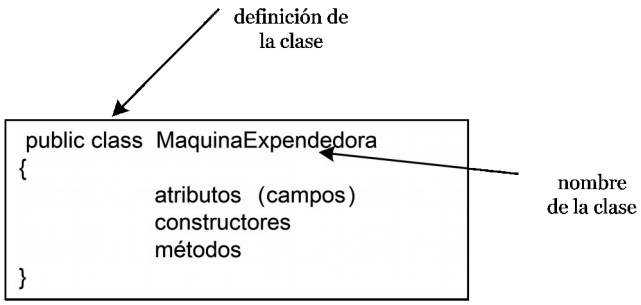
3.1.- Atributos, constructores y métodos

Para explicar los nuevos conceptos utilizaremos el proyecto MaquinaExpendedora. Este proyecto contiene una clase, MaquinaExpendedora, que modela el comportamiento de una máquina que expende tickets de forma automática. Los clientes de esta máquina insertan dinero y solicitan la impresión de un 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.

- Abre el proyecto MaquinaExpendedora
- Crea un objeto *miMaquina* de la clase MaquinaExpendedora
- Llama al método getPrecio()
- Llama al método insertarDinero() que simula la inserción de dinero en la máquina
- Invoca al método getImporte() para verificar que la máquina registra el dinero insertado
- Inserta varias cantidades de dinero y cuando haya suficiente llama al método imprimirTicket()
- Examina el balance (*importe*) de la máquina después de emitir el billete. ¿Cómo lo has hecho?

Una **clase** se define:



public – atributo de visibilidad de la clase.
Indica que la clase será visible desde el exterior

Todo lo que va entre las { } constituye el *cuerpo de la clase* y pertenecerá a ella:

- **atributos** – campos (variables) que almacenan los datos de cada objeto
- **constructores** – métodos "especiales" que permiten crear el objeto de la clase e inicializan apropiadamente sus atributos
- **métodos** – implementan el comportamiento del objeto

3.1.1.- Atributos (campos)

Los atributos almacenan el *estado* de un objeto. Se definen al comienzo de la clase (aunque no es obligatorio hacerlo así, seguiremos este criterio).

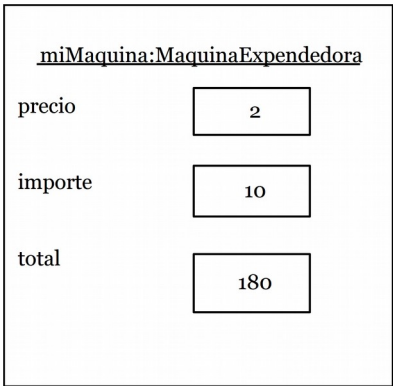
Los atributos o campos se denominan también ***variables de instancia***.

Para cada atributo se especifica:

- su **visibilidad** – en nuestro caso es *private* lo que significa que el atributo sólo será visible dentro de la clase
- tipo del atributo (*int*, *float*, *long*, *char*,.....)
- nombre del atributo

```
public class MaquinaExpendedora
{
   // 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;
   ……..
}
```

Los atributos son variables, *variables de instancia*. Una **variable** es una zona de memoria donde se puede almacenar un valor (de un determinado tipo). Este valor además podrá cambiar.



Las variables de instancia son zonas de memoria dentro del objeto donde se almacenan los valores correspondientes a su estado.

Todas las sentencias en Java terminan en **;**.

Se pueden incluir comentarios en el código fuente de la clase para aclarar su lectura, para hacer el código más legible:

- un comentario de una sola línea se escribe precedido de //

```
// El precio de un ticket en esta máquina
private int precio;
```

- si un comentario ocupa varias líneas se precede de /* y termina en */
- Java soporta además los comentarios *javadoc* que empiezan con /** y terminan en */ (son los que aparecerán en formato HTML cuando se genera la documentación de un proyecto)

Los comentarios son ignorados por el compilador. No son sentencias ejecutables.

3.1.2.- Constructores

El constructor es un método particular que tiene como misión crear un nuevo objeto a partir de la clase e inicializar correctamente sus atributos (su estado) para poder utilizar el objeto.

El constructor tiene el mismo nombre que la clase.

Un constructor *no tiene* valor de retorno.

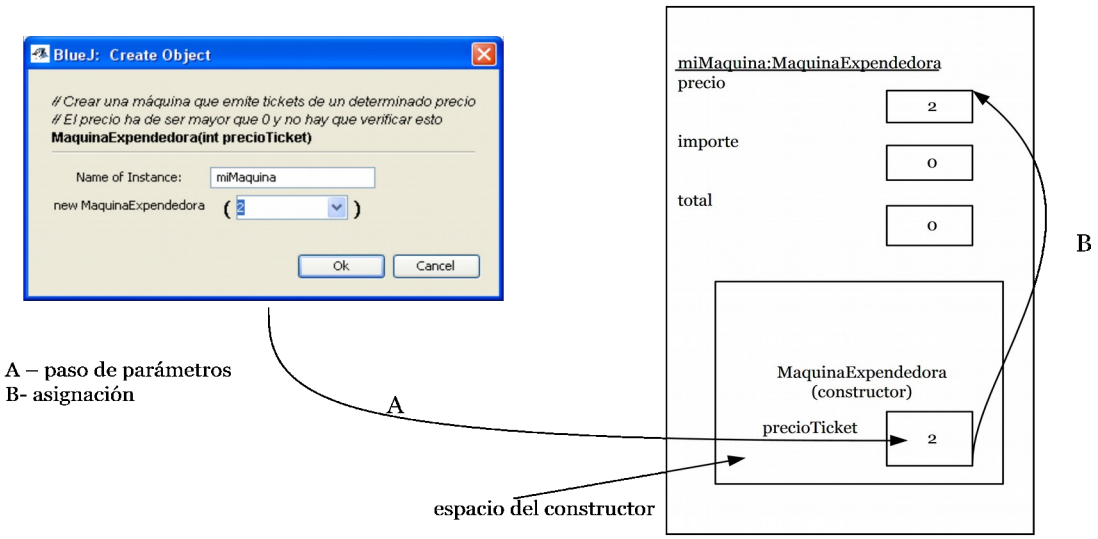
```
public class  MaquinaExpendedora
{
   ……..
   /**
    * Crear una máquina que emite tickets de un determinado precio
    * El precio ha de ser mayor que 0 y no hay que verificar esto
    */
   public MaquinaExpendedora(int precioTicket)
   {
      precio = precioTicket;
      importe = 0;
      total = 0;
   }
  ……
}
```

3.2.- Paso de parámetros

El constructor, como el resto de métodos, puede tener **parámetros**, en nuestro ejemplo, *int precioTicket*. Esto permite proporcionar desde fuera de la clase el valor que tendrá el atributo precio de la máquina y este precio estará disponible (en el atributo) durante todo el tiempo de vida del objeto.

Los parámetros se especifican en la cabecera del constructor.

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



Los parámetros declarados en la cabecera del constructor (o de un método) se denominan **parámetros formales**. Están disponibles para el objeto sólo dentro del cuerpo del constructor (o del método) en el que se declaran. Su ámbito de visibilidad se reduce al constructor (o al método). El ámbito de un atributo, en cambio, es toda la clase, puede ser accedido desde cualquier lugar de la clase.

Los valores que se proporcionan desde fuera a los parámetros formales se denominan **parámetros actuales**.

Un concepto relacionado con una variable es su *tiempo de vida*. El tiempo de vida de un parámetro formal se limita a lo que dure la llamada al constructor o método. Una vez que ha concluido su ejecución los parámetros formales desaparecen y sus valores se pierden.
El tiempo de vida de un atributo es el tiempo de vida del objeto al que pertenece.

Ejer.3.2.

- ¿A qué clase pertenece el siguiente constructor? ¿Cuántos parámetros tiene? ¿Cómo se denomina el parámetro *nombre*, formal o actual?
  public Estudiante(String nombre)
- ¿Cuántos parámetros actuales hay que proporcionar al siguiente constructor?
  public Libro(String titulo, double precio)
- ¿De qué tipo serán los atributos de la clase Libro?

3.3.- Sentencia de asignación

Una sentencia de asignación permite almacenar un valor en una variable. La asignación se realiza con el operador = .

```
variable = expresión;
```

Ej. precio = precioTicket; //guardamos en el atributo precio el valor del parámetro formal //precioTicket

El tipo de la expresión debe coincidir con el tipo de la variable a la que se asigna. Esta regla es válida para los parámetros. La correspondencia entre parámetros formales y actuales ha de ser en orden, tipo y nº.

Ejer.3.3. Supongamos una clase Mascota con un atributo *nombre* de tipo String. Escribe una sentencia de asignación en el siguiente constructor que inicialice el atributo con el valor proporcionado al constructor.

```
public Mascota(String nuevoNombre)
{

}
```

Ejer.3.4. ¿Crees que hay algo incorrecto en el siguiente trozo de código? ¿Qué hay diferente en relación al constructor de la clase MaquinaExpendedora? Haz los cambios en el constructor del proyecto y compílalo. ¿Se produce algún error? Crea un objeto de la clase MaquinaExpendedora e inspecciona el valor del atributo *precio*. ¿Qué ha ocurrido? Deshaz los cambios y deja el constructor tal como estaba.

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

Ejer.3.5. Supongamos definidos los siguientes atributos:

```
private int radio;
private double  perimetro;
private double  area;
```

- asigna al radio el valor 6
- suponiendo que *radio* representa el radio de un círculo, asigna a *perimetro* el perímetro de dicho circulo
- asigna a *area* el área del circulo

Ejer.3.6. Dadas las siguientes declaraciones:

```
private double valorCompra;
private double totalFactura;
```

- asigna a la variable de instancia *valorCompra* el valor 653.36 que representa el valor total de una compra realizada
- asigna a la variable de instancia *totalFactura* el valor de la compra con el incremento del 16% de IVA.

Ejer.3.7.

- Declara una variable *edad* de tipo entero y una variable lógica *esAdulto*.
- Asigna a la variable *esAdulto* una expresión booleana que devuelva *true* si la edad corresponde a una persona mayor de edad
- Declara una variable *teclaPulsada* de tipo carácter y una variable lógica *teclaCorrecta*
- Asigna a la variable *teclaCorrecta* el valor de una expresión lógica que se evalúe a true si la tecla pulsada es correcta, es decir, si es una 'S' o una 's' o una 'N' o una 'n'.

Ejer.3.8. Indica si las siguientes asignaciones son correctas o no:

Declaración de variables	**Asignación de valores a las variables**
int precio;	precio = 35.0;
double precio;	precio = 6;
int texto;	texto = "hola";
float precio;	precio = 34.56f;
double tasa;	tasa = precio * 0.07;
int pago;	pago = 50;
double cambio;	cambio = pago – precio – tasa;

- Un valor de un tipo de menor precisión puede guardarse en una variable de tipo con mayor precisión. Java hace una ***conversión de menor a mayor precisión***. Es una ***conversión implícita***.

int + long → long
int + float → float
float + double → double
int + double → double

A la inversa no es posible. Para permitirlo hay que efectuar una conversión de tipo (***type casting***) o ***conversión explícita***.

Ej.
```
float precio = (float) 34.56;
int valor = (int) 7.67;
char letra = (char) 65; //asigna la letra 'A', se admite char letra = 65;
int suma = 30;
double media = suma / 4; // el resultado es 7.0 y no 7.5
```

(Evitaremos, de momento, las conversiones de tipo)

3.4.- Métodos

Los **métodos** implementan el *comportamiento* de los objetos. Representan, por tanto:

- lo que el objeto puede hacer
- lo que se puede solicitar al objeto (los servicios que proporciona)

El método incluye el código que se ejecuta cuando se envía un mensaje al objeto.

Una clase puede contener cualquier cantidad de métodos, cada uno de ellos debe realizar una tarea específica. La ejecución de un método consiste en la realización de cada una de las sentencias (instrucciones) que contiene, una detrás de otra, en el orden en que aparecen.

Todo método consta de:

- una ***cabecera***: la signatura del método

```
public int getPrecio()
public void imprimirTicket()
```

- un **cuerpo**: contiene la parte de declaraciones de variables locales y sentencias ejecutables (instrucciones) que definen qué ocurre dentro de un objeto cuando el método se ejecuta. El cuerpo se encierra entre **{ }.**

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

    // sentencias
}
```

- 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.

Un método puede tener un tipo de retorno.
Si no devuelve nada se especifica el tipo *void*.

```
public int getPrecio()
public void imprimirTicket()
```

Cuando un método devuelve un valor lo hace a través de la sentencia **return**. Esta sentencia marca, además, el final del método, acaba su ejecución. El tipo de valor devuelto en la sentencia return ha de coincidir con el tipo indicado en la signatura del método.

```
public double calcularArea()
{
    ……..

    return area;  // ha de ser de tipo double
}
```

Un método puede tener o no parámetros:

- si no tiene - `public int getPrecio()`
- si tiene parámetros - `public void setPrecio(int nuevoPrecio)`

Normalmente, los métodos entran en alguna de las siguientes categorías: *métodos accesores, métodos mutadores y otros métodos*.

3.4.1.- Métodos accesores

Estos métodos:

- proporcionan información sobre el estado de un objeto
- permiten responder a preguntas tales como: ¿cuál es el precio de un ticket? ¿de qué color es el coche?
- 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

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

}
```

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

Ejer.3.9.

- Compara el método *getImporte*() con el método *getPrecio*().
- Define un accesor *getTotal*() que devuelve el valor del atributo *total*.
- Elimina la sentencia return del cuerpo del método *getPrecio*(). ¿Qué mensaje de error se genera al compilar la clase?
- Compara la signatura del método *getPrecio*() e *imprimirTicket*(). ¿Cuál es la diferencia principal entre ellas?
- ¿Por qué los métodos *insertarDinero*() e *imprimirTicket*() no tienen sentencias return?

3.4.2.- Métodos mutadores

Estos métodos:

- modifican el estado de un objeto, el valor de uno o varios de sus atributos
- representan peticiones al objeto para que modifique sus atributos
- habitualmente contienen sentencias de asignación y reciben parámetros

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

}
```

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

}
```

- La sentencia importe = importe + cantidad; significa que añadimos a importe lo que tenga más el valor de la variable (parámetro) cantidad. Se dice que estamos *acumulando* el valor de cantidad en importe. La sentencia puede expresarse en Java de forma abreviada con el operador += importe += cantidad;

  En general, *variable = variable + expresión;* es lo mismo que *variable += expresión;*
  variable = variable - expresión; es lo mismo que *variable -= expresión;*

Ejer.3.10.

- ¿Cómo podemos saber por la signatura del método,

  ```
  public void setPrecio(int precioCoste)
  ```

  que no es un constructor?

- Completa el método *setPrecio*() para que actúe como un mutador y cambie el valor del atributo precio.

Ejer.3.11. Completa el cuerpo del siguiente método e indica si es un mutador o un accesor:

```
/**
* Incrementar el atributo puntuación en la cantidad de
*  puntos dados
*/

public void incrementar(int puntos)
{
    ..............................
}
```

Ejer.3.12. Completa el siguiente método:

```
/**
 * Reduce el atributo total en la cantidad dada
 */
public void descontar(int cantidad)
{
    ..............................
}
```

3.5.- Escritura dentro de los métodos

El resultado del método:

```
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;
}
```

produce,

```
############################
# Máquina expendedora BlueJ
# Billete:
#  20  cents.
############################
```

El método System.out.println() escribe el parámetro especificado (será 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");
```

3.5.1.- El operador + como concatenación de cadenas

System.out.println("# " + precio + " cents."); Aquí el operador + actúa como *operador de concatenación* obteniendo una nueva cadena resultado de concatenar los tres valores especificados. El parámetro final que se pasa al método println() es "# 20 cents." si el precio es 20.

System.out.println(); deja una línea en blanco

Ejer.3.13.

- Añade un método a la clase MaquinaExpendedora llamado prompt() . Este método no tendrá parámetros, no devuelve ningún valor y su misión será visualizar en pantalla la siguiente cadena de caracteres: "Por favor, inserte la cantidad correcta de dinero". ¿Qué tipo de método es, accesor, mutador, ...?
- Añade ahora el método mostrarPrecio(). El método no tiene parámetros, no devuelve nada y escribe en pantalla: "El precio total del ticket es XXXX cents." donde XXXX será el valor del atributo *precio*.
- Compila el proyecto. Crea dos objetos de la clase MaquinaExpendedora con diferentes precios para los tickets y llama al método mostrarPrecio() sobre cada objeto. ¿La salida es la misma? ¿Por qué?

Ejer.3.14. Responde a las siguientes cuestiones:

- ¿Cuál es la estructura de una clase?
- ¿Cuál es la función de un constructor?
- ¿Qué particularidades tiene el constructor?
- ¿Cuántos valores devuelve un constructor?
- ¿Qué es un método accesor? ¿Y un mutador?
- Sea el atributo, *private int numeroPuertas;* de una clase Coche. Define un accesor y un mutador para ese atributo.
- ¿Cuál es la función de la sentencia *return*?
- ¿Es correcto, *return cantidad + 3;* ?
- Pon las sentencias, *cantidad = cantidad + valor;* y *suma = suma – 10;* en su forma abreviada.
- Supón una clase con los atributos,

```
private String nombre;
private int edad;
```

escribe un método *saludar()* que muestre en pantalla el mensaje "*Hola, me llamo XXXX y tengo XXXX años*"

3.6.- Variables locales

Una **variable local** es una variable que se declara dentro de un método. Tienen un ámbito que se limita al del método en el que están definidas.

Su tiempo de vida es el tiempo que dura la ejecución de un método. Se crea cuando se invoca al método y se destruye cuando el método concluye.

Todos los métodos, incluidos los constructores, pueden incluir variables locales. Se declaran indicando su tipo y el nombre de la variable (no se especifica su visibilidad, *private*, ... como en los atributos).

Se utilizan a menudo como almacenamiento temporal, para ayudar a un método a realizar su tarea.

Ej.
```
public class  Cuadrado
{
    private double lado;
    ..................................................
    /**
     * Cálculo del área de un cuadrado
     */
    public double calcularArea()
    {
       double area;  //es una variable local
       area = lado * lado;
       return area;
    }
}
```

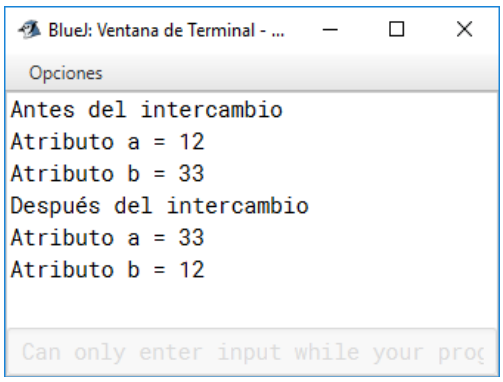
Ejer.3.15. Crea un nuevo proyecto en BlueJ, el proyecto EjemploIntercambio y define una clase Ejemplo con dos atributos de tipo entero, *a* y *b*. Incluye el constructor de la clase que tendrá dos parámetros, los que van a permitir inicializar los atributos. Añade un método intercambiar() cuya responsabilidad será intercambiar el valor de los dos atributos (deberás utilizar una variable local para efectuar el intercambio). Añade un método escribir() que permita visualizar en pantalla el estado de un objeto de la clase Ejemplo de la forma.

Atributo a = XXX
Atributo b = XXX

Añade dos métodos más, promptAntes*()* que visualiza la cadena "*Antes del intercambio*" y promptDespues*()* que visualiza la cadena "*Después del intercambio* ".

Incluye comentarios aclaratorios en los métodos y en el constructor.

Compila el proyecto. Crea un objeto de la clase y llama en el orden adecuado a los métodos de forma que obtengan en la ventana del terminal lo que se muestra en la figura siguiente:



3.7.- Constantes en Java

El valor de una variable puede cambiar a lo largo de la ejecución de un programa. Una **constante**, sin embargo, representa a un valor que nunca cambia.

Para declarar una constante en Java:

final tipo nombre_constante = valor;

```
final double PI = 3.1416;
private final double IVA = 0.16;
```

La declaración es similar a la de un atributo salvo que:

- se incluye la palabra reservada ***final*** antes del tipo de la constante. Esto indica a Java que el valor que se declara no puede cambiar.
- hay que inicializar la constante en el momento de la declaración
- por convención se escriben en mayúsculas
- la visibilidad (*private*, ...) sólo se especifica si la constante es un atributo más de la clase. Si la constante se declara dentro de un método (de forma similar a una variable local) no hay que especificar visibilidad.

3.8.- Resumen acerca de las variables

Una *variable* es una zona de memoria que tiene un nombre y a la que se le asigna un valor. Este valor puede modificarse a lo largo de la ejecución de un programa.

Toda variable tiene un tipo de datos.

Las variables pueden ser: *atributos*, *parámetros formales y variables locales*. Antes de ser utilizada, toda variable ha de ser declarada indicando su nombre y su tipo. Si la variable es un atributo habrá que especificar además su visibilidad.

a) ***Atributos*** –

- se definen fuera de los constructores y métodos
- se utilizan para almacenar datos acerca del estado de un objeto
- su tiempo de vida es el tiempo de vida de un objeto
- tienen un ámbito de clase, son accesibles desde cualquier punto de la clase, por lo tanto, podemos referirnos a ellos desde los constructores y desde los métodos
- se definen como *private* (esta es la visibilidad que especificaremos para los atributos) por lo que no son accesibles desde el exterior de la clase
- Java asigna un valor por defecto a los atributos si no se indica nada cuando se declaran

Ej.

```
public class  Persona
{
    private String nombre;
    private int  edad;
     private double  peso;
     private char  estadoCivil;
            ……..
}
```

b) ***Parámetros formales*** –

- son variables locales que se definen en la cabecera de un constructor o un método
- su ámbito es el del constructor o método que los define
- su tiempo de vida es lo que dura la ejecución del constructor o método
- reciben sus valores desde fuera siendo inicializados por los valores de los parámetros actuales que forman parte de la llamada al constructor o método

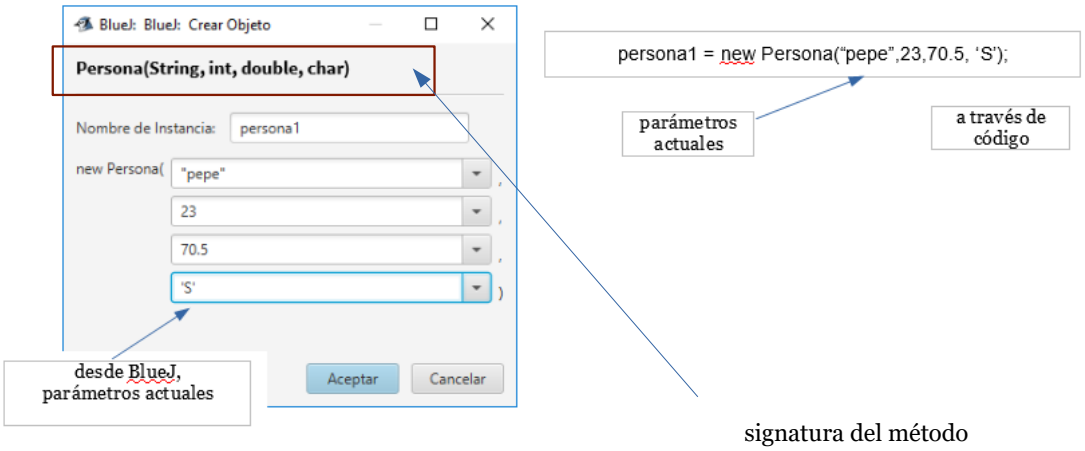
Ej.

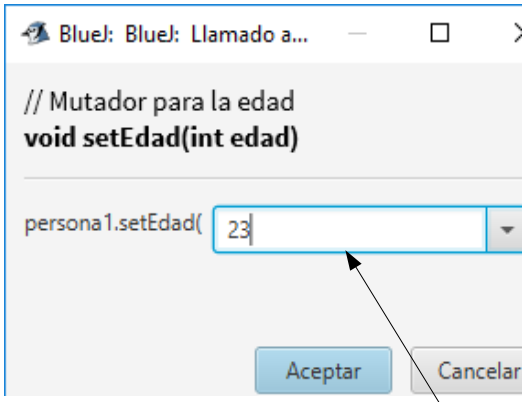
```
public class  Persona
{
    ..................................................
        /**
        * Constructor de la clase Persona
        */
       public Persona(String queNombre, int queEdad,
                                 double quePeso, char queEstadoCivil)
        {
        ..................................................
        }

        public void setPeso(double nuevoPeso)
        {
        ..................................................
        }

}
```

parámetros formales





persona1.setEdad(23);

parámetro actual

parámetro actual

c) ***Variables locales*** –

- son variables temporales
- se definen dentro del constructor o método
- se utilizan sólo dentro del cuerpo del constructor o método que las define

- deben ser inicializadas antes de utilizarlas en una expresión (Java no asigna un valor por defecto a las variables locales)
- su ámbito se limita al bloque en el que están definidas

Ej.

```
public int  calcular(int valor)
{
      int tmp = 0;
      tmp = tmp + valor;
      return tmp;
}
```

3.9.- Alterando el flujo de control de un programa: la sentencia condicional

Volviendo a nuestro proyecto MaquinaExpendedora, si analizamos el comportamiento de la máquina de tickets observamos que es inadecuado en algunos casos:

- no verifica que las cantidades de dinero introducidas sean positivas
- no devuelve ningún cambio
- no verifica si se ha introducido suficiente dinero para emitir un ticket

Mejoraremos el comportamiento de esta máquina. El nuevo código de la clase (incluido en el proyecto MaquinaExpendedoraMejorada) será el siguiente:

```
/**
 * Este proyecto modela una sencilla máquina expendedora de billetes.
 * El precio del ticket se especifica vía el constructor.
 * Los objetos verifican que un usuario solo introduce cantidades positivas de dinero y solo
 * emiten un ticket si se ha introducido suficiente dinero
 */
public class  MaquinaExpendedora
{
    // 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 verificación de esto
          */
    public MaquinaExpendedora(int precioTicket)
    {
        precio = precioTicket;
        importe = 0;
        total = 0;
    }

    /**
     * Devolver el precio de un billete
     */
    public int getPrecio()
    {
        return precio;
    }

    /**
     * Devolver la cantidad de dinero insertada hasta el momento
     * para el billete
     */
    public int getImporte()
    {
        return importe;
    }

    /**
     * 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);
        }
    }
```

```
    /**
     * Imprimir un ticket si se ha introducido suficiente dinero
     * y reducir el importe restando el precio del ticket. Escribir un mensaje de error si
     * se necesita más dinero
     */
    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 ");
        }

    }

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

}
```

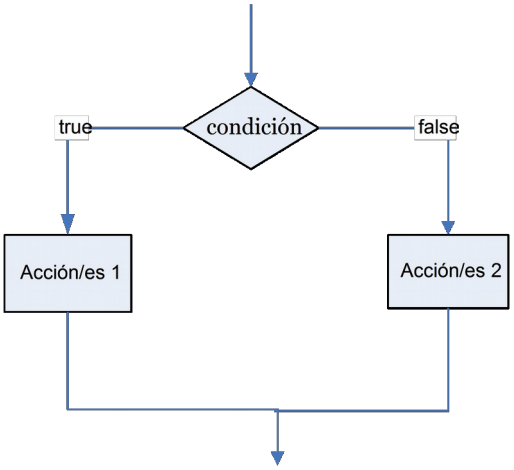
Hasta ahora, cuando desde BlueJ hemos invocado a un método y comenzaba su ejecución, ésta ha sido *secuencial*. Las sentencias contenidas en el método (o constructor) se han ejecutado una detrás de otra hasta que el método concluía.

Muchas veces es necesario alterar este flujo secuencial. Esto se consigue utilizando sentencias de control dentro de los métodos.

Las sentencias o *instrucciones de control* permiten:

- realizar uno u otro conjunto de instrucciones si se cumple una determinada condición → instrucciones **condicionales** (*if, switch*)
- ejecutar un conjunto de instrucciones repetidamente un nº determinado o indeterminado de veces → sentencias **repetitivas** (bucles – *while, do ... while, for*)

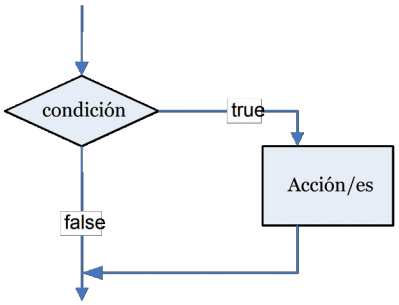
3.9.1.- La instrucción condicional *if*



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
}
```

- Consideraciones sobre la sentencia if

Si el conjunto de acciones a ejecutar es una única sentencia puede omitirse las { } del bloque.

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.

Ej.

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

```
        public void imprimirTicket()
        {

            if (importe  >=  precio)
            {
             ..............................
            }
            else
            {
                System.out.println("# Debe insertar al menos: " +
                            (precio - importe) + " céntimos más ");
            }

    }
```

Ej.

```
        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");
                        }
            }

        }


  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");
          }

}
```

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

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

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

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

Ejer.3.16.

- Abre el proyecto MaquinaExpendedoraMejorada.
- Verifica el comportamiento de la máquina simulada creando varias instancias y llamando al método insertarDinero() con diferentes valores de parámetros actuales
- Verifica el valor del importe antes y después de llamar a insertarDinero*()*
- Prueba el método insertarDinero() introduciendo una cantidad negativa y/o 0. ¿Cambia el valor del importe?
- Analiza el código del método devolverCambio(). ¿Qué tipo de variable es la variable *cambio*? Modifica el código del método devolverCambio() incluyendo únicamente las sentencias: *importe = 0;* y *return importe;* Pruébalo.
- ¿Qué pasa si hacemos *return importe;* y *importe = 0;* Pruébalo.

Ejer.3.17.

- Añade un nuevo método vaciarMaquina() . Este método simula que sacamos todo el dinero de la máquina. El método debe devolver el valor del atributo *total* y ponerlo a 0. ¿Es un accesor o un mutador?
- Reescribe el método imprimirTicket() de forma que:

  - declares una variable local *cantidadPendiente* y la inicialices con la diferencia entre el precio y el importe
  - si el valor de esta variable es >= 0 se imprime un ticket, si no lo es se emite un mensaje de error indicando lo que todavía queda por pagar
  - testea (prueba) este método y comprueba que su comportamiento es el mismo que el de la versión anterior

Ejer.3.18.

- Crea un nuevo proyecto Cantante.
- Añade al proyecto la clase Cantante. Esta clase tiene un atributo *feliz* de tipo booleano.
- Los objetos de esta clase están inicialmente felices. Indica esto en el constructor.
- Añade los siguientes métodos y pruébalos. Indica si son accesores o mutadores.

  - public void cantar() - cuando se le envía el mensaje *cantar()* al objeto se examina su estado de ánimo. Si está feliz se emite el mensaje "*Cantando bajo la lluvia*", si está triste se escribe "*No tiene ánimo para cantar*"
  - public boolean estaTriste() – haz dos versiones de este método , una con la sentencia *if* y otra sin utilizar dicha sentencia
  - public void cambiarEstadoAnimo() – si esta triste se pone feliz y viceversa. Hazlo sin *if*.

Ejer.3.19.

- Crea un nuevo proyecto que incluya una clase Numero. Los objetos de esta clase permiten almacenar un nº y nos indican si ese nº que contienen es menor, mayor o igual que otro que reciben. La clase incluye un único atributo entero, *numero* y exhibe el siguiente comportamiento:

  - el constructor tiene un parámetro que permite inicializar el atributo *numero*
  - public boolean esMayorQue(int otroNumero) - si el valor que guarda el objeto es mayor que el nº pasado como parámetro devuelve *true*, en caso contrario false
  - public boolean esMenorQue(int otroNumero) - si el valor que guarda el objeto es menor que el nº pasado como parámetro devuelve *true*, en caso contrario false
  - public boolean esIgualQue(int otroNumero) - si el nº que se pasa como parámetro es igual que el atributo del objeto devuelve *true*, en caso contrario *false*
  - public String comprobar(int otroNumero) - si el atributo es mayor que el nº que se pasa como parámetro devuelve la cadena "*Más grande*", si es menor devuelve la cadena "*Es menor*", si son iguales devuelve "*Son iguales*"

3.9.2.- Otra sentencia condicional: la instrucción *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;

}
```

- **Consideraciones sobre la sentencia *switch***

La sentencia *switch* consta de una expresión que se evalúa a un valor entero (*int*, *byte*, *short*) o *char* . (*)

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.

Ej.

```
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;
}
```

(*) Una de las mejoras que introdujo la versión Java 7 fue la posibilidad de utilizar expresiones String en esta sentencia.

```
String dia = "Martes";
switch (dia)
{
   case "Lunes": System.out.println("Hoy es Lunes");
                 break;
   case "Martes": System.out.println("Hoy es Martes");
                 break;
   case "Miércoles": System.out.println("Hoy es Miércoles");
                 break;
   case "Jueves": System.out.println("Hoy es Jueves");
                 break;
   case "Viernes": System.out.println("Hoy es Viernes");
                 break;
   default:
                 System.out.println("Día no laborable");

}
```

Ejer.3.20. Escribe una sentencia *switch* que analice el valor de una variable *mes* y devuelva los días que tiene en la variable *diasMes*. Cuando se trate del mes 2 (febrero) habrá que tener en cuenta si el año (contenido en la variable *anio*) es o no bisiesto (un año es bisiesto si es múltiplo de 4) .