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

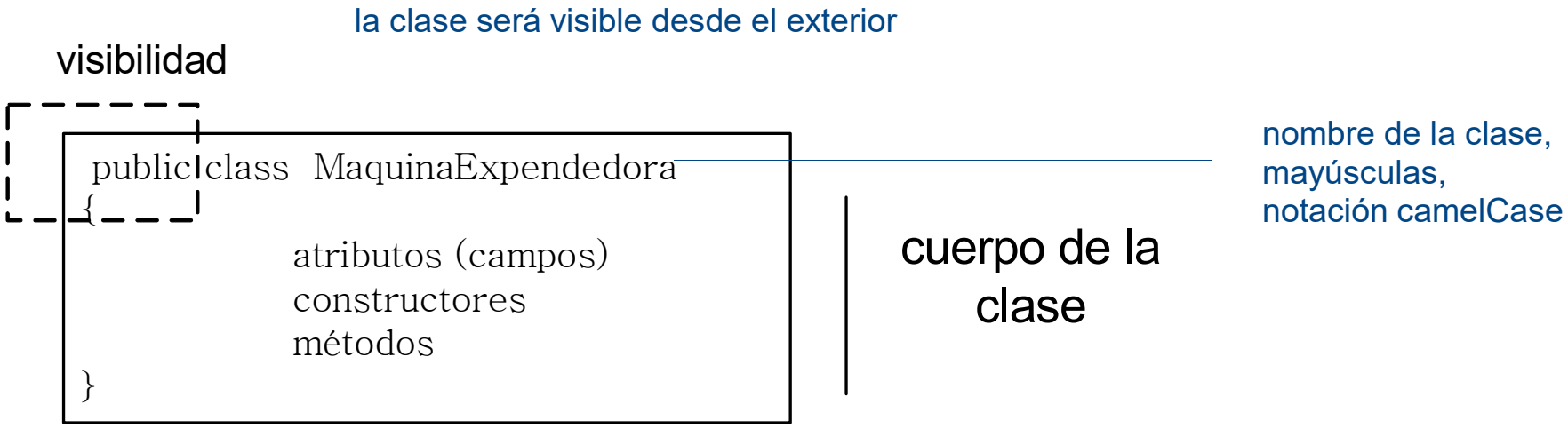
CLASE

- Plantilla para describir objetos.
- Define atributos (estado) y métodos (comportamiento) para todos los objetos de esa clase

OBJETO

- Instancia particular de una clase.
- Pueden haber múltiples instancias de una clase.
- Cada instancia su propio estado.
- Todas se comportan igual → métodos.
- Los métodos describen el comportamiento de los objetos

Visibilidad (*public* o *private*)



La visibilidad indicará si la clase, atributo o métodos serán visibles desde el exterior. En esta unidad veremos 2 tipos:

- **public**: será visible desde el exterior.
- **private**: NO será visible desde el exterior.

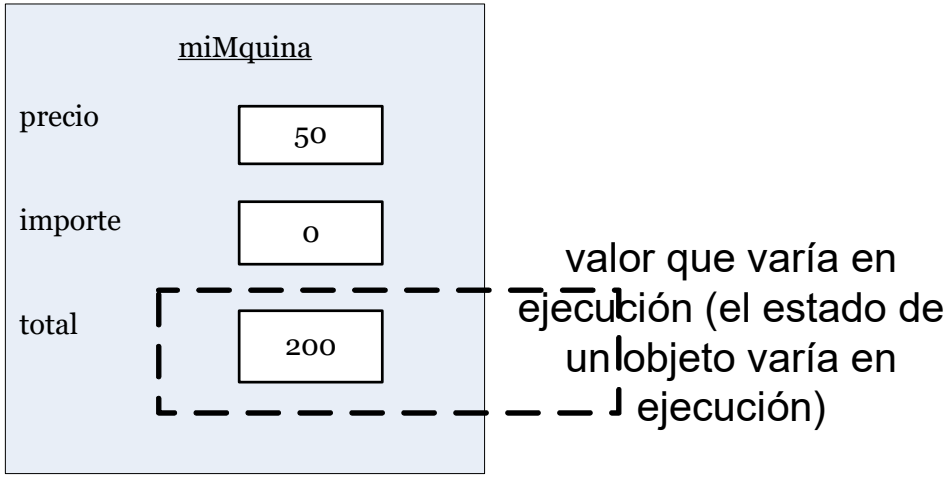
Atributos

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)

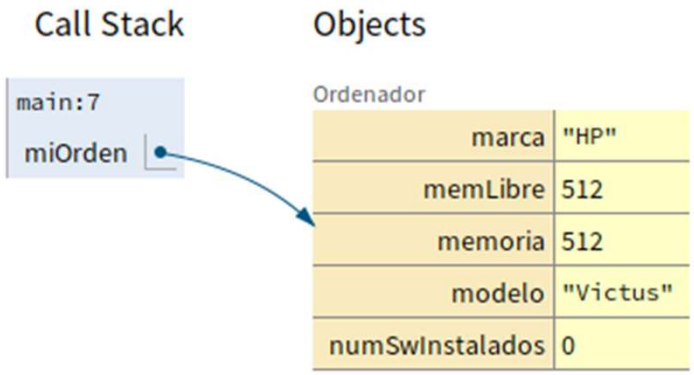


Atributos

```java
public class Ordenador {  2 usages
    // ATRIBUTOS DE LA CLASE
    public String marca;  1 usage
    public String modelo;  1 usage
    public int memoria;  3 usages
    public int memLibre;  6 usages
    public int numSwInstalados;  5 usages

    public Ordenador(String queMarca, String queModelo, int queMemoria){
        this.marca = queMarca;
        this.modelo = queModelo;
        this.memoria = queMemoria;
        this.memLibre = this.memoria;
        this.numSwInstalados = 0;
    }
    // MÉTODOS DE LA CLASE
    public void instalarSW(int memoriaAInstalar){  1 usage
        memLibre = memLibre - memoriaAInstalar;
        numSwInstalados = numSwInstalados + 1;
    }

    public void Formatear(){  no usages
        numSwInstalados = 0;
        memLibre = memoria;
    }

    public int obtenerMemLibre(){  no usages
        return memLibre;
    }
}
```

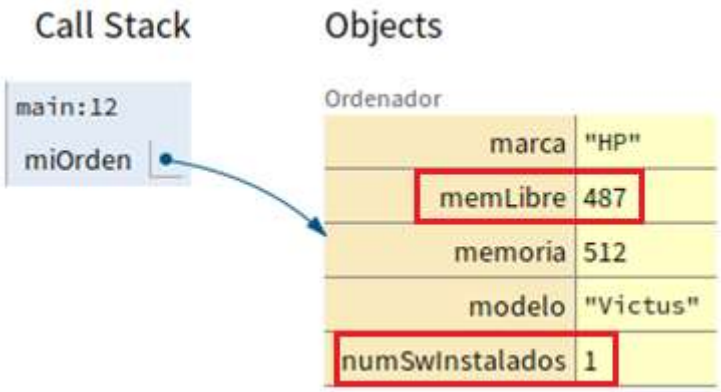
Atributos

```
public class Main {
    public static void main(String[] args) {   args: []

        Ordenador miOrden = new Ordenador( queMarca: "HP",  queModelo: "Victus",  queMemoria: 512);   mi
        System.out.println(miOrden.memLibre);   miOrden: Ordenador@798
        System.out.println(miOrden.numSwInstalados);
        miOrden.instalarSW( memoriaAInstalar: 25);


    }
}
```



Atributos

```
public class Main {
    public static void main(String[] args) {  args: []

        Ordenador miOrden = new Ordenador( queMarca: "HP",  queModelo: "Victus",  queMemoria: 512);
        System.out.println(miOrden.memLibre);
        System.out.println(miOrden.numSwInstalados);
        miOrden.instalarSW( memoriaAInstalar: 25);   miOrden: Ordenador@798


    }
}
```



Atributos

- Por cada atributo
  - visibilidad (private – solo visible dentro de la clase)
  - tipo del atributo (int, double, ….)
- 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

Cuando 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 Ordenador {  2 usages
    // ATRIBUTOS DE LA CLASE
    public String marca;  1 usage
    public String modelo;  1 usage
    public int memoria;  3 usages
    public int memLibre;  6 usages
    public int numSwInstalados;  5 usages

    public Ordenador(String queMarca, String queModelo, int queMemoria){
        this.marca = queMarca;
        this.modelo = queModelo;
        this.memoria = queMemoria;
        this.memLibre = this.memoria;
        this.numSwInstalados = 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, ….)

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 class Ordenador {  2 usages
    // ATRIBUTOS DE LA CLASE
    public String marca;  1 usage
    public String modelo;  1 usage
    public int memoria;  3 usages
    public int memLibre;  6 usages
    public int numSwInstalados;  5 usages

    public Ordenador(String queMarca, String queModelo, int queMemoria){
        this.marca = queMarca;
        this.modelo = queModelo;
        this.memoria = queMemoria;
        this.memLibre = this.memoria;
        this.numSwInstalados = 0;
    }
```

Paso de parámetros

- **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.

```
public class Ordenador {  2 usages
    public String marca;  1 usage
    public String modelo;  1 usage
    public int memoria;  3 usages
    public int memLibre;  6 usages
    public int numSwInstalados;  5 usages

    public Ordenador(String queMarca, String queModelo, int queMemoria){
        this.marca = queMarca;
        this.modelo = queModelo;
        this.memoria = queMemoria;
        this.memLibre = this.memoria;
        this.numSwInstalados = 0;
    }
```

PARÁMETROS FORMALES

```
Ordenador miOrden = new Ordenador( queMarca: "HP", queModelo: "Victus", queMemoria: 512);
System.out.println(miOrden.memLibre);
System.out.println(miOrden.numSwInstalados);
miOrden.instalarSW( memoriaAInstalar: 25);
```

PARÁMETROS ACTUALES

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.

Parámetros formales y parámetros actuales.

Paso de parámetros en Java – **por valor**

se copia el valor de los parámetros actuales en los formales (los actuales no de modifican)

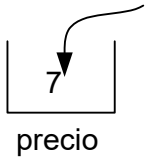
Sentencia de asignación

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

Variables locales

```
// Metodo que devuelve la suma de la memoria total
//  y la libre
public int sumaTotalLibre(){  no usages
    return memoria + memLibre;
}
```

```
// Metodo que devuelve la suma de la memoria total
//  y la libre
public int sumaTotalLibre(){  no usages
    int suma;
    suma = memoria + memLibre;
    return suma;
}
```

int suma → ***Variable local*** del método sumaTotalLibre.
No es visible fuera del método sumaTotalLibre.

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

```
// Metodo que devuelve la suma de la memoria total
//  y la libre
public int sumaTotalLibre(){  no usages
    int suma;
    suma = memoria + memLibre;
    return suma;
}
```

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

Atributos/variables locales/Parámetros

VISIBILIDAD	Atributos	Parámetros	Variables locales
En toda la clase	SI	NO (Sólo en su método)	NO (Sólo en su método)
Fuera de la clase	Sólo si son púbicos	NO	NO

Ejer 3.3 a 3.8

Realizar los ejercicios 3.3 a 3.8

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.

Métodos

```
// MÉTODOS DE LA CLASE
public void instalarSW(int memoriaAInstalar){
    memLibre = memLibre - memoriaAInstalar;
    numSwInstalados = numSwInstalados + 1;
}

public void Formatear(){  1 usage
    numSwInstalados = 0;
    memLibre = memoria;
}
```

Estructura de un método

- **Cabecera** – signatura 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
}
```

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 Formatear()*
- 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

Valor de retorno de un método

```java
// Metodo que devuelve la suma de la memoria total
//   y la libre
public int sumaTotalLibre(){  no usages
    int suma;
    suma = memoria + memLibre;
    return suma;
}
```

```java
public int obtenerMemLibre(){
    return memLibre;
}
```

Métodos accesores

Los métodos entran en alguna de las siguientes categorías:

- **métodos accesores**
- **métodos mutadores**
- **otros métodos**

Métodos **accesores** (***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 accesores

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

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

Añade al proyecto Bombilla una accesor para la potencia y un accesor para conocer la situación de la bombilla.

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)

Realizar ejercicios 3.10 – 3.12

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

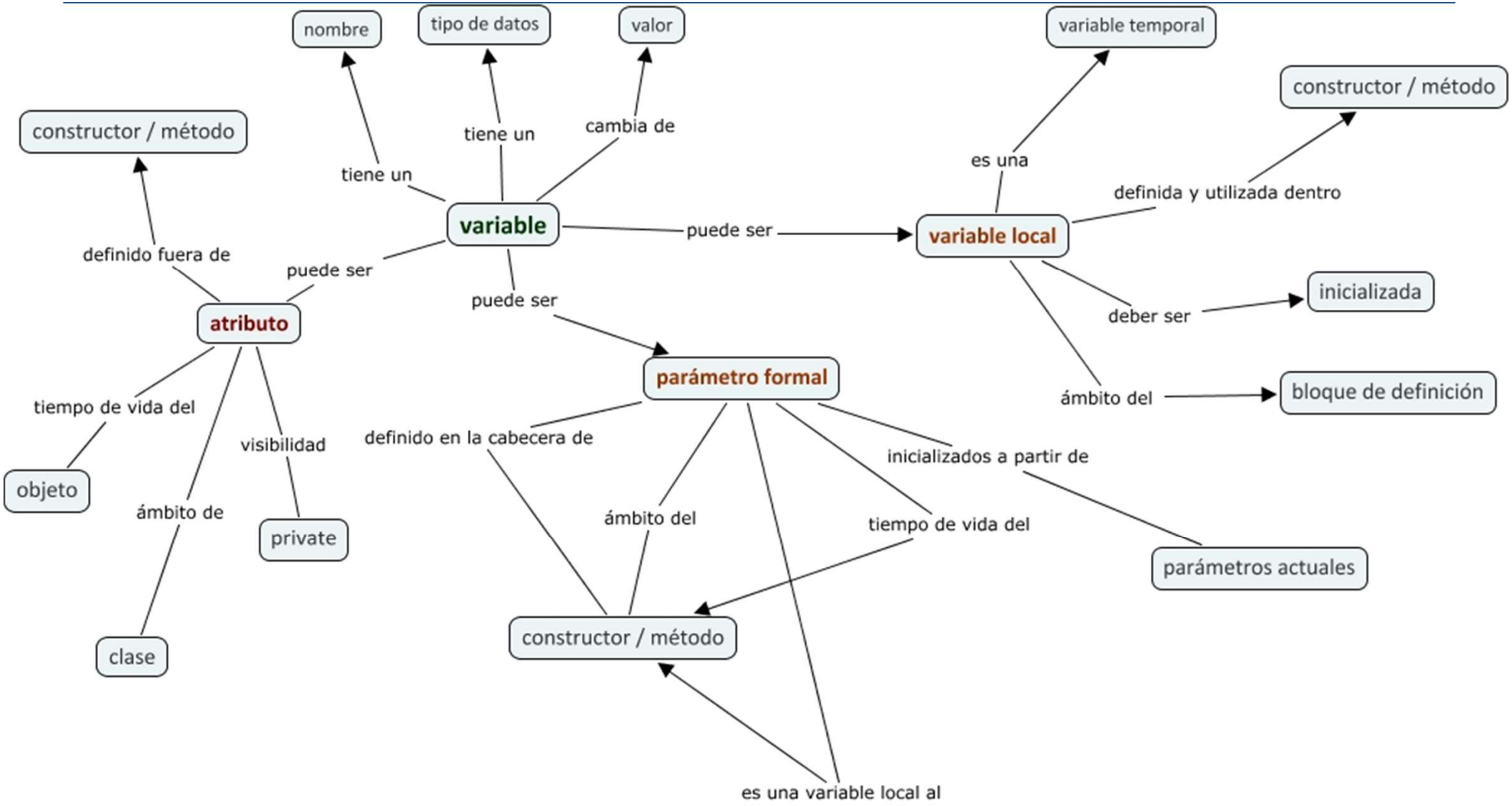
Constantes en Java

- 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;***

Constantes en Java

- 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



Cuestionario UT3

- Ejercicios
- AD01 Libro
- AD02Hucha
- AD03 Contador
- AD04FacturaLuz
- AD05 ConversorFarenheit
- AD06 Descomposición en monedas

Estructuras de control condicionales

- 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)

Estructuras de control condicionales

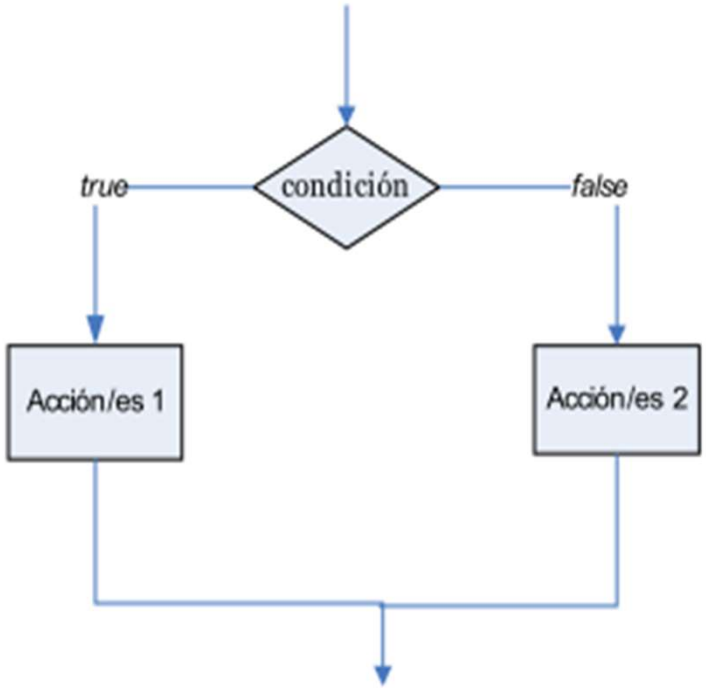
Recordemos el proyecto de nuestro ordenador en el método ***InstalarSW*** → ¿Se ha tenido en cuenta el caso en el que la ***memoriaLibre*** sea menor a la ***memoria del software a instalar***?

En este caso no se podría instalar.

Estructuras de control condicionales

```java
public class Main {
    public static void main(String[] args) {
        int edad;

        edad = 10;

        if (edad < 13){
            System.out.println("Se trata de un niño");
        } else if (edad < 18) {
            System.out.println("Se trata de un adolescente");
        } else if (edad < 65) {
            System.out.println("Se trata de un adulto");
        }else {
            System.out.println("Se trata de un jubilado");
        }

    }
}
```

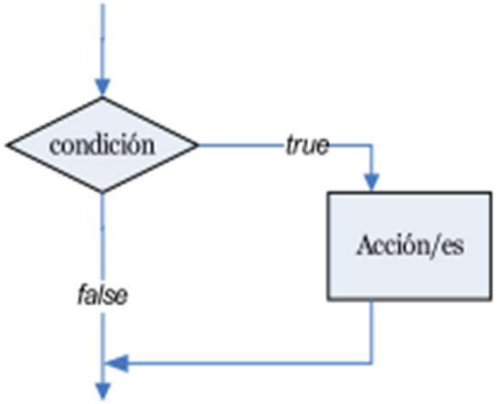
Sentencia if



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

```
public boolean  metodoMisterio(int valor)  {

     return (valor >= 0);
}
```

Ejemplos sentencia if

- Completar clase EjemplosCondicional
- Ejer 3.16 a 3.19

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

Consideraciones sobre la sentencia switch

- 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

```java
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;
}
```

```java
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;
}
```