

## Objetos

La POO se basa en diseñar programas mediante el uso de *OBJETOS*, identificando componentes o actores que mediante unas determinadas acciones, interaccionando entre ellos, solucionan un problema determinado. Los objetos son por tanto abstracciones de elementos relacionados con el área del problema, diseñados para reflejar su comportamiento. Casi todo puede ser considerado un objeto: el dinero, el helicóptero, una persona, una tabla, un ordenador, un avión, un diccionario, la ciudad o la capa de ozono. Los objetos representan cosas, simples o complejas. Principalmente, un objeto complejo es aquel que está compuesto por otros objetos, como por ejemplo un objeto '*texto*' que está compuesto por otros objetos '*párrafos*' más pequeños.

## Clases

Una clase permite describir objetos similares mediante la definición de sus estructuras de datos y métodos comunes. Las clases son plantillas para objetos, permitiendo la agrupación de objetos que comparten las mismas propiedades y comportamiento. Así, existen rectángulos de varios tamaños, pero todos ellos pertenecen a la clase *rectángulo*, que define los atributos comunes para todos **ancho** y **alto**, a la vez que el método (o función miembro) **calcularArea**, también común para todos.

Ejemplo: Se puede hablar de la clase *Mamífero*, que representa las características comunes a todos los mamíferos. Para identificar a un mamífero particular en esta clase, hay que hablar de "este mamífero" o aquel mamífero".

## Métodos y mensajes

Los métodos no son más que *funciones miembro* de una clase. Se trata de funciones que, automáticamente, tratan con los atributos del objeto como si fuesen variables locales a la función.

Los **métodos**, por tanto, son el mecanismo de acceso y manipulación de los datos del objeto, constituyen la interfaz del objeto con el resto de objetos. Cada método está constituido por un conjunto de instrucciones escritas en un lenguaje de programación estructurado determinado, asociadas a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por este o por sus descendientes.

Ejemplo: Supongamos el conjunto de objetos de los *Empleados* de una empresa. Para un empleado concreto, queremos calcular su salario mensual, pero para ello se necesita conocer el número de horas trabajadas. Se envía un mensaje al objeto, *empleado23*, de activar `calcularSalario()` con unas *horasTrabajadas* determinadas:

```
empleado23.calcularSalario( horasTrabajadas )
```

## Calificadores de acceso de clase en JAVA.

Una declaración de clase puede ir precedida de modificadores que otorgan a la clase ciertas propiedades:

- **public.** Una clase public es públicamente accesible.
- **abstract.** Una clase abstracta se considera incompleta y no se pueden crear instancias de dicha clase. Esto sucede porque la clase contiene métodos abstract que deben ser implementados por una subclase.
- **final.** Una clase final no admite subclases.

Una clase no puede ser a la vez *final* y *abstract*.

Una declaración de clase puede estar precedida por varios modificadores.

### Atributos

Los **atributos de una clase** no son más que *datos miembro* de una clase. Deben cumplir una serie de premisas:

- ☐ *Deben ser significativas dentro del entorno de la aplicación.* Es decir, deben servir para identificar claramente y de manera única a cada uno de los objetos.
- ☐ *El número de atributos debe ser el mínimo para realizar todas las operaciones que requiere la aplicación.*

### Calificadores de Acceso de atributos en JAVA.

Los atributos de clase pueden precederse con modificadores. Pueden ser:

- ☐ **Tipos de calificadores de acceso:**
- ☐ *predeterminado.* Sólo son accesibles desde las clases del paquete
- ☐ *private.* Sólo son accesibles en la propia clase
- ☐ *protected.* Son accesibles en las subclases de la clase y en la propia clase.
- ☐ *public.* Son accesibles en cualquier parte donde la clase sea accesible.
- ☐ **static.** Se crean variables de clase. Cuando se declara un campo `static` en una clase sólo existe una copia de ese campo, independientemente del número de instancias de la clase que se creen.

Un atributo `static` puede ser referido directamente dentro de su propia clase, pero cuando se accede a él externamente, hay que utilizar el nombre de la clase.

Ejemplo:

```
static int numero;  
...  
Vehiculo.numero;
```

- ☐ **final.** Un atributo final es aquel cuyo valor no puede cambiar después de ser inicializado.

Ejemplo:

```
private static final int NUM_COCHES=10;
```

### Variables miembro de clase (static)

Una clase puede tener variables propias de la clase y no de cada objeto. A estas variables se les llama **variables de clase** o variables **static**. Las variables **static** se suelen utilizar para definir constantes comunes para todos los objetos de la clase (por ejemplo **PI** en la clase **Circulo**) o variables que sólo tienen sentido para toda la clase (por ejemplo, un contador de objetos creados como **numCirculos** en la clase **Circulo**).

Las variables de clase son lo más parecido en **Java** a las **variables globales** de C/C++.

Las variables de clase se crean anteponiendo la palabra **static** a su declaración. Para llamarlas se suele utilizar el nombre de la clase (no es imprescindible, pues se puede utilizar también el nombre de cualquier objeto), porque de esta forma su sentido queda más claro. Por ejemplo, **Circulo.numCirculos** es una variable de clase que cuenta el número de círculos creados.

Las variables miembro **static** se inicializan siempre antes que cualquier objeto de la clase.

## MÉTODOS (FUNCIONES MIEMBRO)

### Métodos de objeto o de instancia

Los **métodos** son funciones definidas dentro de una clase. Salvo los métodos **static** o de clase, se aplican siempre a un objeto de la clase por medio del **operador punto**. La primera línea de la definición de un método se llama **declaración** o **header** o **prototipo**; el código comprendido entre las **llaves** {...} es el **cuerpo** o **body** del método.

El **header** consta del cualificador de acceso (**public**, por ejemplo), del tipo del valor de retorno (**int** por ejemplo, **void** si no tiene), del **nombre de la función** y de una lista de **argumentos explícitos** entre paréntesis, separados por comas. Si no hay argumentos explícitos se dejan los paréntesis vacíos.

Los métodos tienen **visibilidad directa** de las variables miembro del objeto que es su **argumento implícito**, es decir, pueden acceder a ellas sin cualificarlas con un nombre de objeto y el operador punto (.).

El **valor de retorno** puede ser un valor de un **tipo primitivo** o una **referencia**. En cualquier caso no puede haber más que un único valor de retorno (que puede ser un objeto o un array). Se puede devolver también una referencia a un objeto por medio de un nombre de **interface**. El objeto devuelto debe pertenecer a una clase que implemente esa interface. A diferencia de las variables miembro, las variables locales no se inicializan por defecto. Si en el **header** del método se incluye la palabra **native** (Ej: `public native void miMetodo() ;`) no hay que incluir el código o implementación del método. Este código deberá estar en una librería dinámica (*Dynamic Link Library* o DLL). Estas librerías son ficheros de funciones compiladas normalmente en lenguajes distintos de **Java** (C, C++, Fortran, etc.). Es la forma de poder utilizar conjuntamente funciones realizadas en otros lenguajes desde código escrito en **Java**.

Un método también puede declararse como **synchronized** (Ej: `public synchronized double miMetodoSynch() { ... }`). Estos métodos tienen la particularidad de que sobre un objeto no pueden ejecutarse simultáneamente dos métodos que estén sincronizados.

### Metodos set y get

Los métodos **get** y **set**, son simples métodos que usamos en las clases para mostrar (**get**) o modificar (**set**) el valor de un atributo. En general se deben crear cuando el atributo sea privado o cuando siendo protegido, no puede ser accedido directamente desde otra clase.

El nombre del método siempre será **get** o **set** y a continuación el nombre del atributo con la primera letra en mayúscula. Su modificador siempre es **public**, ya que queremos mostrar o modificar desde fuera la clase. Por ejemplo, si el atributo es nombre entonces serán: **getNombre** o **setNombre**.

Esta es la sintaxis de cada uno:

```

public tipo_atributo getAtributo () {
    //devuelve el tipo del atributo

    return atributo; // retorna el atributo

}

public void setAtributo (tipo_atributo variable) {

    //no devuelve nada

    this.atributo = variable;

    //asigna al atributo, el valor recibido como parámetro

    //this se usa sólo cuando el nombre del atributo y el nombre de la variable
    recibida como parámetro sean idénticos.


}

```


## UML: características de algunos entornos de desarrollo

### ○ Atributos:

Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:


- **public** (+, 

(3) Java - Conceptos Importantes – Objetos – Clases - Métodos– Página 4 de 5  
 Prof. Cristina Domizio

- **protected** (#, 

#### ○ **Métodos:**

Los métodos u operaciones de una clase son la forma en que ésta interactúa con su entorno, éstos pueden tener las características:

- **public** (+, 

(3) Java - Conceptos Importantes – Objetos – Clases – Métodos– Página 5 de 5  
Prof. Cristina Domizio