

## **UNIDAD 4: DESARROLLO DE CLASES**

### **Módulo Profesional: Programación**

# Índice

RESUMEN INTRODUCTORIO.....	3
INTRODUCCIÓN.....	3
CASO INTRODUCTORIO .....	4
1. CONCEPTO DE CLASE Y OBJETO .....	5
1.1 Clase .....	5
1.2 Objeto.....	7
2. ESTRUCTURA Y MIEMBROS DE UNA CLASE. DIAGRAMAS DE CLASE.....	9
3. CREACIÓN DE ATRIBUTOS .....	10
4. CREACIÓN DE MÉTODOS .....	11
5. CREACIÓN DE CONSTRUCTORES .....	12
6. UTILIZACIÓN DE CLASES Y OBJETOS .....	14
6.1 Crear objetos en java .....	14
6.2 Declarar un objeto .....	14
6.3 Ejemplarizar una clase.....	15
6.4 Inicializar un objeto.....	16
7. CONCEPTO DE HERENCIA. TIPOS. UTILIZACIÓN DE CLASES HEREDADAS .....	17
7.1 Relación de herencia (generalización / especialización, es un) .....	17
7.2 Relación de agregación (todo / parte, forma parte de) .....	20
7.3 Relación de composición .....	21
7.4 Relación de asociación («uso», usa, cualquier otra relación).....	23
8. LIBRERÍAS DE CLASES. CREACIÓN. INCLUSIÓN Y USO DE LA INTERFACE .....	27
RESUMEN FINAL .....	31

## RESUMEN INTRODUCTORIO

A lo largo de esta unidad veremos la definición de clase y objeto y la diferencia que existe entre estos dos conceptos. Posteriormente veremos cómo se crean los atributos, métodos y los métodos constructores. A continuación, se detallará la utilización de clases y objetos: cómo crear objetos en Java, declararlos, ejemplarizar una clase e inicializar un objeto. Finalmente volveremos sobre el concepto de herencia y los tipos de relaciones que pueden existir entre clases, para finalizar con las librerías de clases, su creación e inclusión. Para cada uno de los apartados veremos ejemplos en el lenguaje de programación Java.

## INTRODUCCIÓN

Los lenguajes de Programación Orientada a Objetos (POO) ofrecen medios y herramientas para describir los objetos manipulados por un programa. Los paradigmas de la programación han ido evolucionando desde enfoques como el imperativo hasta la programación orientada a objetos. La POO es una forma especial de programar que es más cercana a la forma en la que se podría expresar las cosas en la vida real. Por ejemplo, se puede pensar en un animal para tratar de modelizarlo en un esquema de POO. Se podría decir que el animal es el elemento principal que tiene una serie de propiedades, como podrían ser el tamaño, qué comen o si viven en el agua o no. Además, un Animal puede hacer cosas como desplazarse, comer o hacer ruido.

## CASO INTRODUCTORIO

En la empresa en la que trabajas te han propuesto como candidato al puesto de jefe de proyecto. Te han asignado un equipo de trabajo y tienes que defender la realización del próximo proyecto utilizando la programación orientada a objetos. Debes destacar ante tu equipo las posibilidades que ofrece el concepto de clase para poder crear cualquier tipo de objeto y los elementos de las clases (atributos y métodos) que os van a permitir modelar las características de los objetos y las acciones que se van a poder realizar sobre la información que contenga cada clase o familia de objetos.

Al finalizar la unidad el alumnado:

- ✓ Identificará los principales atributos y métodos que representan a una clase.
- ✓ Será capaz de modelar cualquier objeto del mundo real utilizando el concepto de clase y sus elementos.
- ✓ Conocerá los distintos tipos de relaciones existentes entre los objetos.

# 1. CONCEPTO DE CLASE Y OBJETO

La programación Orientada a Objetos es una metodología que basa la estructura de los programas en torno a los **objetos**.

Los lenguajes de POO ofrecen medios y herramientas para describir los objetos manipulados por un programa. Más que describir cada objeto individualmente, estos lenguajes proveen una construcción (**Clase**) que describe a un conjunto de objetos que poseen las mismas propiedades.



## ¿SABÍAS QUE...?

Simula 67 fue el primer lenguaje de programación orientado a objetos. Simula 67 fue lanzado oficialmente por sus autores Ole Johan Dahl y Kristen Nygaard en mayo de 1967, en la Conferencia de Trabajo en Lenguajes de Simulación IFIO TC 2, en Lysebu cerca de Oslo.

## 1.1 Clase

Se ha visto en la unidad anterior que una clase puede definirse como la agrupación o colección de objetos que comparten una estructura común y un comportamiento común.

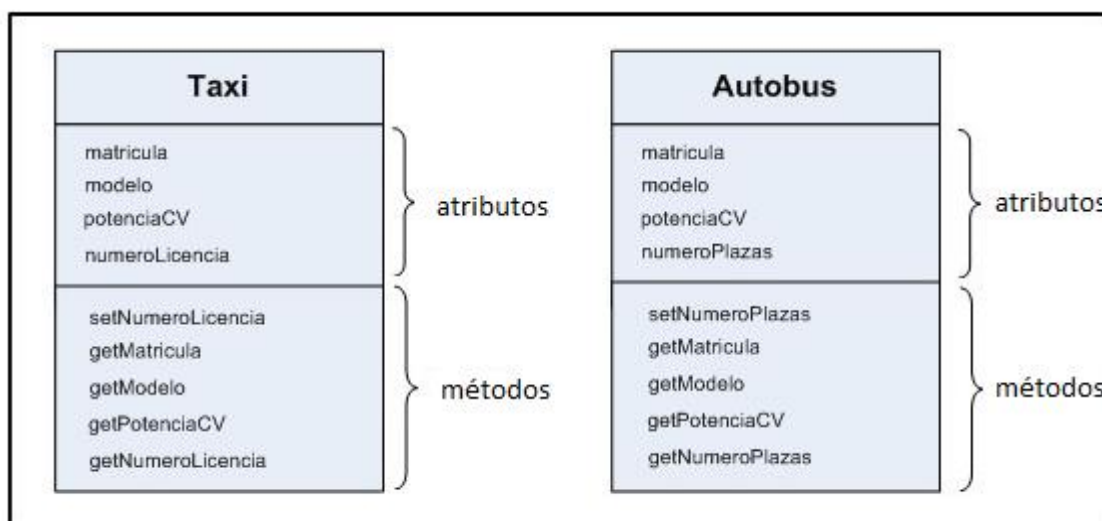
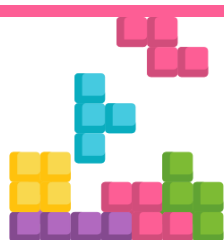


Imagen: ejemplo clases (POO)



## EJEMPLO PRÁCTICO

La clase Autobus en Java:

```
public class Autobus {  
    private String matricula;  
    private String modelo;  
    private int potenciaCV;  
    private int numeroPlazas;  
  
    public void setNumeroPlazas(int numeroPlazas) {  
        this.numeroPlazas = numeroPlazas;  
    }  
  
    public String getMatricula() {  
        return matricula;  
    }  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public int getPotenciaCV() {  
        return potenciaCV;  
    }  
  
    public int getNumeroPlazas() {  
        return numeroPlazas;  
    }  
}
```



### EJEMPLO PRÁCTICO

La clase Taxi en Java:

```
public class Taxi {  
    private String matricula;  
    private String modelo;  
    private int potenciaCV;  
    private String numeroLicencia;  
  
    public void setNumeroLicencia(String numeroLicencia) {  
        this.numeroLicencia = numeroLicencia;  
    }  
  
    public String getMatricula() {  
        return matricula;  
    }  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public int getPotenciaCV() {  
        return potenciaCV;  
    }  
  
    public String getNumeroLicencia() {  
        return numeroLicencia;  
    }  
}
```

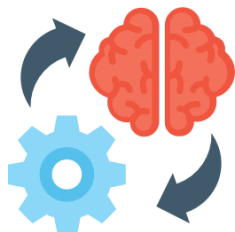
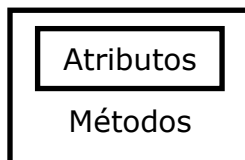
## 1.2 Objeto

Se ha visto en la unidad anterior que un objeto es una entidad caracterizada por sus **atributos** propios y cuyo comportamiento está determinado por las **acciones** o funciones que pueden modificarlo, así como también las acciones que requiere de otros objetos. Un objeto tiene identidad e inteligencia y constituye una unidad que **oculta** tanto datos como la **descripción** de su manipulación. Puede ser definido como una encapsulación y una abstracción: una **encapsulación** de atributos y servicios, y una **abstracción** del mundo real.

Para el contexto del Enfoque Orientado a Objetos (EOO) un objeto es una entidad que **encapsula** datos (**atributos**) y **acciones** o funciones que los

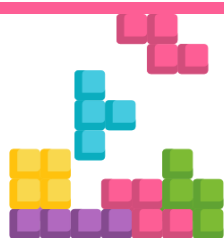
manejan (**métodos**). También para el EOO un objeto se define como una **instancia** o particularización de una clase.

#### Objeto



#### RECUERDA

Un objeto es una entidad (tangible o intangible) que posee características y acciones que realiza por sí solo o interactuando con otros objetos. Consta de atributos y métodos.



#### EJEMPLO PRÁCTICO

Se pueden crear tantos objetos Autobus y/o Taxi como se necesiten instanciándolos con el operador new en Java:

```
Autobus bus1 = new Autobus();
Autobus bus2 = new Autobus();
Taxi taxi1 = new Taxi();
Taxi taxi2 = new Taxi();
```



## 2. ESTRUCTURA Y MIEMBROS DE UNA CLASE. DIAGRAMAS DE CLASE

Una clase es una plantilla que contiene la descripción general de una colección de objetos. Consta de atributos y métodos que resumen las características y el comportamiento comunes de un conjunto de objetos.

Todo objeto (también llamado instancia de una clase), pertenece a alguna clase. Mientras un objeto es una entidad concreta que existe en el tiempo y en el espacio, una clase representa solo una abstracción.

Todos aquellos objetos que pertenecen a la misma clase son descritos o comparten el mismo conjunto de atributos y métodos.

Todos los objetos de una clase tienen el mismo formato y comportamiento, son diferentes únicamente en los valores que contienen sus atributos. Todos ellos responden a los mismos mensajes.

Su sintaxis algorítmica es:

```
Clase <Nombre de la Clase>  
...  
FClase <Nombre de la Clase>;
```

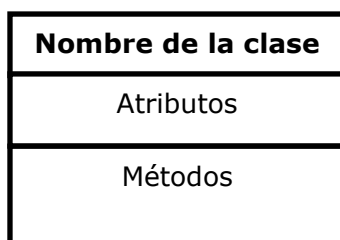


Imagen: representación de una clase

### 3. CREACIÓN DE ATRIBUTOS

Son los datos o variables que **caracterizan** al objeto y cuyos valores en un momento dado indican su estado.

Un **atributo** es una característica de un objeto. Mediante los atributos se define información oculta dentro de un objeto, la cual es manipulada solamente por los métodos definidos sobre dicho objeto. Un atributo consta de un nombre y un valor. Cada atributo está asociado a un tipo de dato, que puede ser simple (entero, real, lógico, carácter, *string* (cadena de caracteres)) o estructurado (*array*, registro, archivo, lista, etc.)

Su sintaxis algorítmica es:

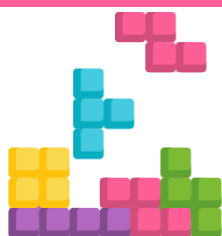
**<Modo de Acceso> <Tipo de dato> <Nombre del Atributo>;**

Los **modos de acceso** son:

**Público:** Atributos (o Métodos) que son accesibles fuera de la clase. Pueden ser llamados por cualquier clase, aun si no está relacionada con ella. Este modo de acceso **también se puede** representar con el símbolo +.

**Privado:** Atributos (o Métodos) que sólo son accesibles dentro de la implementación de la clase. **También** se puede representar con el símbolo –.

**Protegido:** Atributos (o Métodos) que son accesibles para la propia clase y sus clases hijas (subclases). También se puede representar con el símbolo #.



#### EJEMPLO PRÁCTICO

Declaración de atributos de la clase Autobus en Java:

```
private String matricula;
private String modelo;
private int potenciaCV;
private int numeroPlazas;
```

## 4. CREACIÓN DE MÉTODOS

Son las operaciones (acciones o funciones) que se aplican sobre los objetos y que permiten crearlos, cambiar su estado o consultar el valor de sus atributos.

Los métodos constituyen la secuencia de acciones que implementan las operaciones sobre los objetos. La implementación de los métodos no es visible fuera del objeto.

La **sintaxis algorítmica** de los métodos expresados como funciones y acciones es:

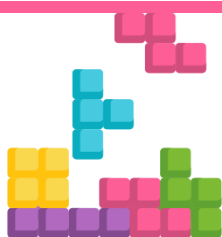
Para **funciones** se pueden usar cualquiera de estas dos sintaxis:

`<Modo de Acceso> Función <Nombre> [(Lista Parámetros)]:  
<Descripción del Tipo de datos>`

Para **acciones**:

`<Modo de Acceso> Acción <Nombre> [(Lista Parámetros)]` donde los parámetros son opcionales

Ejemplo: Un rectángulo es un objeto caracterizado por los atributos Largo y Ancho, y por varios métodos, entre otros Calcular su área y Calcular su perímetro.



### EJEMPLO PRÁCTICO

Ejemplo de creación de métodos en la clase Taxi en Java:

```
public void setNumeroLicencia(String numeroLicencia) {  
    this.numeroLicencia = numeroLicencia;  
}  
  
public String getMatricula() {  
    return matricula;  
}  
  
public String getModelo() {  
    return modelo;  
}  
  
public int getPotenciaCV() {  
    return potenciaCV;  
}  
  
public String getNumeroLicencia() {  
    return numeroLicencia;  
}
```

## 5. CREACIÓN DE CONSTRUCTORES

Cada objeto o instancia de una clase debe ser creada explícitamente a través de un método u operación especial denominado **Constructor**.

Los atributos de un objeto toman valores iniciales dados por el constructor. Por convención el método constructor tiene el mismo nombre de la clase (aunque esto no es obligatorio en pseudocódigo) y no se le asocia un modo de acceso (**es público**).

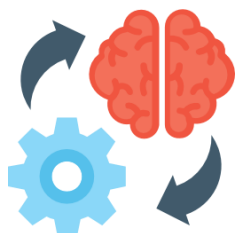
Algunos lenguajes proveen un método constructor por defecto para cada clase y/o permiten la definición de más de un método constructor.

### Métodos destructores de objetos:

Los objetos que ya no son utilizados en un programa, ocupan inútilmente espacio de memoria, que es conveniente recuperar en un momento dado.

Según el lenguaje de programación utilizado esta tarea es dada al programador o es tratada automáticamente por el procesador o soporte de ejecución del lenguaje.

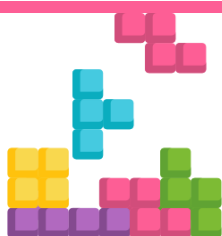
En la **notación algorítmica** NO tomaremos en cuenta ese problema de administración de memoria, por lo tanto no definiremos formas para destruir objetos. En cambio, al utilizar lenguajes de programación si debemos conocer los métodos destructores suministrados por el lenguaje y utilizarlos a fin de eliminar objetos una vez no sean útiles.



### RECUERDA

Si se definen constructores personalizados para una clase, el constructor por defecto (sin parámetros) para esa clase deja de ser generado por el compilador. Si se necesita tener un constructor sin parámetros se tiene que crear para poder utilizarlo.

Si se ha creado un constructor con parámetros y no se ha implementado el constructor por defecto, el intento de utilización del constructor por defecto producirá un error de compilación (el compilador no lo hará por nosotros).



### EJEMPLO PRÁCTICO

Definición de constructores (el primero sin parámetros, y el segundo con cuatro parámetros) de la clase Taxi en Java:

```
public Taxi() {

}

public Taxi(String matricula, String modelo,
            int potenciaCV, String numeroLicencia) {

    this.matricula = matricula;
    this.modelo = modelo;
    this.potenciaCV = potenciaCV;
    this.numeroLicencia = numeroLicencia;
}
```

## 6. UTILIZACIÓN DE CLASES Y OBJETOS

A lo largo de los siguientes apartados se estudiará cómo realizar la creación de objetos en java, así como a declarar e inicializar un objeto. También, se verá cómo ejemplarizar una clase en java.

### 6.1 Crear objetos en java

En Java, se crea un objeto mediante la instanciación de una clase o, en otras palabras, ejemplarizando una clase.

Frecuentemente, se verá la creación de un objeto Java con una sentencia similar a esta.

```
Date hoy = new Date();
```

Esta sentencia crea un objeto de la clase Date (Date es una clase del paquete java.util). Esta sentencia realmente realiza tres acciones: declaración, ejemplarización e inicialización.

**Date hoy** es una declaración de variable que sólo le dice al compilador que el nombre **hoy** se va a utilizar para referirse a un objeto cuyo tipo es Date, el operador **new** ejemplariza la clase Date (creando un nuevo objeto Date), y **Date()** inicializa el objeto.



#### ARTÍCULO DE INTERÉS

Se recomienda visitar el artículo de Oracle sobre creación de objetos:

<http://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html>

### 6.2 Declarar un objeto

La declaración de un objeto es una parte innecesaria de la creación de un objeto. Las declaraciones aparecen frecuentemente en la misma línea que la creación del objeto. Como cualquier otra declaración de variable, las declaraciones de objetos pueden aparecer solitarias como esta.

Date hoy;

De la misma forma, declarar una variable para contener un objeto es exactamente igual que declarar una variable que va a contener un tipo primitivo.

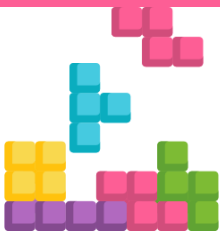
### tipo nombre

donde tipo es el tipo de dato del objeto y nombre es el nombre que va a utilizar el objeto. En Java, las clases e interfaces son como tipos de datos. Entonces tipo puede ser el nombre de una clase o de un interface.

Las declaraciones notifican al compilador que se va a utilizar nombre para referirse a una variable cuyo tipo es tipo. **Las declaraciones no crean nuevos objetos.** `Date hoy` no crea un objeto `Date`, sólo crea un nombre de variable para contener un objeto `Date`. Para ejemplarizar la clase `Date`, o cualquier otra clase, se utiliza el operador **new**.

## 6.3 Ejemplarizar una clase

El operador **new** ejemplariza una clase mediante la asignación de memoria para el objeto nuevo de ese tipo. Este operador **new** necesita un sólo argumento: una llamada al método **constructor**. Los métodos constructores son métodos especiales proporcionados por cada clase Java que son responsables de la inicialización de los nuevos objetos de ese tipo. El operador **new** crea el objeto, el **constructor** lo inicializa.



### EJEMPLO PRÁCTICO

Ejemplo del uso del operador **new** para crear un objeto `Rectangle` (`Rectangle` es una clase del paquete `java.awt`).

```
new Rectangle(0, 0, 100, 200);
```

En el ejemplo, `Rectangle(0, 0, 100, 200)` es una llamada al constructor de la clase `Rectangle`.

El operador **new** devuelve una referencia al objeto recién creado. Esta referencia puede ser asignada a una variable del tipo apropiado.

```
Rectangle rect = new Rectangle(0, 0, 100, 200);
```

## 6.4 Inicializar un objeto

Las clases proporcionan métodos constructores para inicializar los nuevos objetos de ese tipo. Una clase podría proporcionar **múltiples constructores** para realizar diferentes tipos de inicialización en los nuevos objetos.

Cuando se vea la implementación de una clase, se reconocerán los constructores porque tienen el **mismo nombre** que la clase y **no tienen tipo de retorno**. Si se vuelve a la creación del objeto Date en la sección inicial, se puede ver que el constructor utilizado no tenía ningún argumento.

```
Date()
```

Un constructor que no tiene ningún argumento, como el mostrado arriba, es conocido como **constructor por defecto**. Al igual que Date, la mayoría de las clases tienen **al menos** un constructor, el constructor por defecto.

Si una clase tiene varios constructores, todos ellos tienen el mismo nombre pero se deben diferenciar en el número o el tipo de sus argumentos. Cada constructor inicializa el nuevo objeto de una forma diferente. Junto al constructor por defecto, la clase Date proporciona otro constructor que inicializa el nuevo objeto con un nuevo año, mes y día.

Ejemplo: `Date cumpleaños = new Date(1963, 8, 30);`

El compilador puede diferenciar los constructores a través del tipo y del número de sus argumentos.



### ARTÍCULO DE INTERÉS

Se recomienda visitar el artículo de Oracle sobre uso de objetos:

<http://docs.oracle.com/javase/tutorial/java/javaOO/usingobject.html>

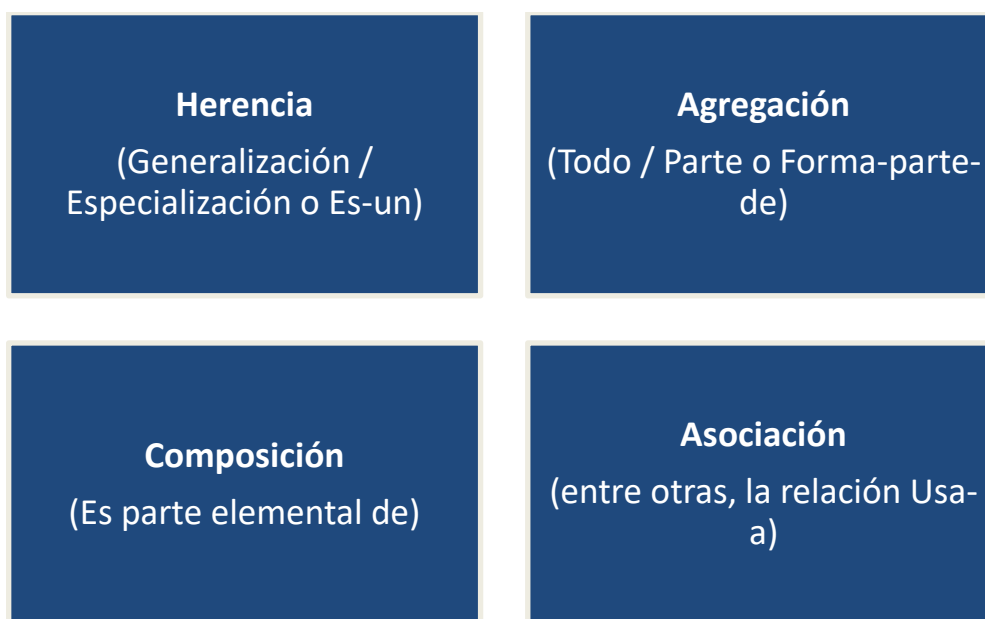


## 7. CONCEPTO DE HERENCIA. TIPOS. UTILIZACIÓN DE CLASES HEREDADAS

Las clases no se construyen para que trabajen de manera aislada, la idea es que ellas se puedan **relacionar entre sí**, de manera que puedan compartir atributos y métodos sin necesidad de reescribirlos.

La posibilidad de establecer **jerarquías** entre las clases es una característica que diferencia esencialmente la programación orientada a objetos de la programación tradicional, ello debido fundamentalmente a que permite **extender** y **reutilizar** el código existente sin tener que reescribirlo cada vez que se necesite.

Los cuatro tipos de relaciones entre clases estudiados en la unidad anterior son:



### 7.1 Relación de herencia (generalización / especialización, es un)

Es un tipo de jerarquía de clases, en la que cada subclase contiene los atributos y métodos de una (herencia simple) o más superclases (herencia múltiple).

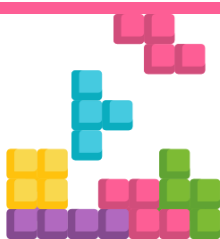
Mediante la herencia las instancias de una clase hija (o subclase) pueden acceder tanto a los atributos como a los métodos públicos y protegidos de la clase padre (o superclase). Cada subclase o clase hija en la jerarquía es siempre una extensión (esto es, conjunto estrictamente más grande) de la(s) superclase(s) o clase(s) padre(s) y además incorporar atributos y métodos propios, que a su vez serán heredados por sus hijas.



### EJEMPLO PRÁCTICO

Ejemplo de clase Animal en Java:

```
public class Animal {  
  
    private String nombre;  
    private int numPatas;  
    private String tamanho;  
  
    public Animal(String nombre, int numPatas, String tamanho){  
        this.nombre = nombre;  
        this.numPatas = numPatas;  
        this.tamanho = tamanho;  
    }  
  
    public String getnombre() {  
        return this.nombre;  
    }  
  
    public void setNombre(String nombre){  
        this.nombre = nombre;  
    }  
  
    public int getNumPatas(){  
        return this.numPatas;  
    }  
  
    public void setNumPatas(int numPatas){  
        this.numPatas = numPatas;  
    }  
  
    public String getTamanho(){  
        return this.tamanho;  
    }  
  
    public void setTamanho(String tamanho){  
        this.tamanho = tamanho;  
    }  
}
```



### EJEMPLO PRÁCTICO

Ejemplo de clase Mamifero en Java, que hereda de la clase Animal:

```
public class Mamifero extends Animal {  
  
    public Mamifero(){  
  
    }  
  
    public Mamifero(String nombre, int numPatas, String tamanho) {  
        super(nombre,numPatas,tamanho);  
    }  
  
    public void rugir() {  
        System.out.println("Argggghhhhhhhhh!!!!");  
    }  
  
    public void dormir() {  
        System.out.println("Duermo como un mamífero");  
    }  
  
    public void comer() {  
        System.out.println("Como como un mamífero");  
    }  
}
```

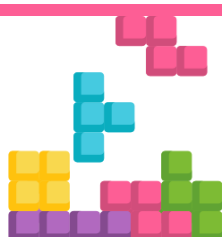


### ¿SABÍAS QUE...?

Hay dos tipos de herencia: **Simple** y **Múltiple**. La primera indica que se pueden definir nuevas clases solamente a partir de una clase inicial mientras que la segunda indica que se pueden definir nuevas clases a partir de dos o más clases iniciales. **Java** sólo permite **herencia simple**.

## 7.2 Relación de agregación (todo / parte, forma parte de)

Es una relación que representa a los objetos compuestos por otros objetos. Indica Objetos que a su vez están formados por otros. El objeto en el nivel superior de la jerarquía es el todo y los que están en los niveles inferiores son sus partes o componentes.



### EJEMPLO PRÁCTICO

Ejemplo de clase Habitación en Java:

```
public class Habitación {
    private double metrosCuadrados;
    private int numVentanas;

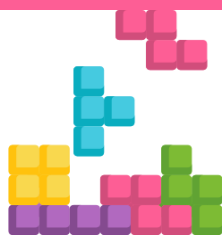
    public Habitación(double metrosCuadrados, int numVentanas) {
        this.metrosCuadrados = metrosCuadrados;
        this.numVentanas = numVentanas;
    }

    public double getMetrosCuadrados() {
        return metrosCuadrados;
    }

    public void setMetrosCuadrados(double metrosCuadrados) {
        this.metrosCuadrados = metrosCuadrados;
    }

    public int getNumVentanas() {
        return numVentanas;
    }

    public void setNumVentanas(int numVentanas) {
        this.numVentanas = numVentanas;
    }
}
```



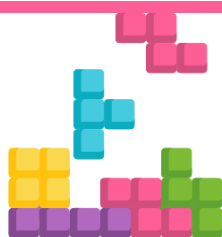
### EJEMPLO PRÁCTICO

Ejemplo de clase Casa en Java. Existe una relación de **Agregación** entre Casa y Habitación:

```
public class Casa {  
  
    Habitacion comedor = new Habitacion(30, 2);  
    Habitacion dormitorio1 = new Habitacion(25, 2);  
    Habitacion banho1 = new Habitacion(15, 1);  
  
}
```

## 7.3 Relación de composición

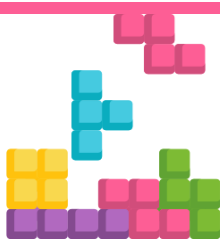
Un componente es parte esencial de un elemento. La relación es más fuerte que el caso de agregación, al punto que, si el componente es eliminado o desaparece, la clase mayor deja de existir.



## EJEMPLO PRÁCTICO

Ejemplo de clase PlazaAparcamiento en Java:

```
public class PlazaAparcamiento {  
  
    private String numero;  
    private double tamanho;  
  
    public PlazaAparcamiento(String numero, double tamanho) {  
        this.numero = numero;  
        this.tamanho = tamanho;  
    }  
  
    public String getNumero() {  
        return numero;  
    }  
  
    public void setNumero(String numero) {  
        this.numero = numero;  
    }  
  
    public double getTamanho() {  
        return tamanho;  
    }  
  
    public void setTamanho(double tamanho) {  
        this.tamanho = tamanho;  
    }  
  
}
```



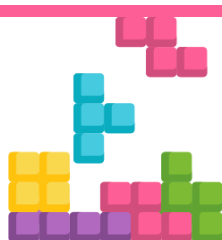
### EJEMPLO PRÁCTICO

Ejemplo de clase Parking en Java. Esta clase está compuesta de un número de objetos PlazaAparcamiento:

```
public class Parking {  
  
    private int numPlazas;  
    private PlazaAparcamiento[] plazas;  
  
    public Parking(int numPlazas) {  
        this.numPlazas = numPlazas;  
        plazas = new PlazaAparcamiento[numPlazas];  
    }  
  
    public int getNumPlazas() {  
        return plazas.length;  
    }  
  
    public void setNumPlazas(int numPlazas) {  
        this.numPlazas=numPlazas;  
    }  
  
    public PlazaAparcamiento[] getPlazas() {  
        return plazas;  
    }  
  
    public void setPlazas(PlazaAparcamiento[] plazas) {  
        this.plazas = plazas;  
    }  
  
}
```

## 7.4 Relación de asociación («uso», usa, cualquier otra relación)

Es una asociación que se establece cuando dos clases tienen una dependencia de utilización, es decir, una clase utiliza atributos y/o métodos de otra para funcionar. Estas dos clases no necesariamente están en jerarquía, es decir, no necesariamente una es clase padre de la otra, a diferencia de las otras relaciones de clases.

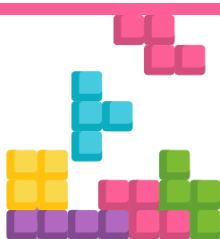


## EJEMPLO PRÁCTICO

Ejemplo de clase Cliente en Java:

```
public class Cliente {  
    private String nombre;  
    private String apellidos;  
    private String DNI;  
  
    public Cliente(String nombre, String apellidos, String dNI) {  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
        DNI = dNI;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getApellidos() {  
        return apellidos;  
    }  
  
    public void setApellidos(String apellidos) {  
        this.apellidos = apellidos;  
    }  
  
    public String getDNI() {  
        return DNI;  
    }  
  
    public void setDNI(String dNI) {  
        DNI = dNI;  
    }  
}
```





### EJEMPLO PRÁCTICO

Ejemplo de clase CuentaBancaria en Java. Existe una relación de **Asociación** entre la clase Cliente y la clase CuentaBancaria:

```
public class CuentaBancaria {
    private String numCuenta;
    private long saldo;
    private Cliente titular;

    public CuentaBancaria(String numCuenta, long saldo, Cliente titular) {
        this.numCuenta = numCuenta;
        this.saldo = saldo;
        this.titular = titular;
    }

    public String getNumCuenta() {
        return numCuenta;
    }

    public void setNumCuenta(String numCuenta) {
        this.numCuenta = numCuenta;
    }

    public long getSaldo() {
        return saldo;
    }

    public void setSaldo(long saldo) {
        this.saldo = saldo;
    }

    public Cliente getTitular() {
        return titular;
    }

    public void setTitular(Cliente titular) {
        this.titular = titular;
    }
}
```



### ENLACE DE INTERÉS

Se puede ampliar información y ver más ejemplos sobre las relaciones de asociación en la POO visitando la web:

<http://www.cristalab.com/tutoriales/programacion-orientada-a-objetos-asociacion-vs-composicion-c893371/>

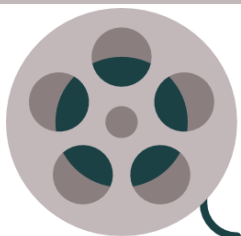


### ENLACE DE INTERÉS

Si se desea ampliar sus conocimientos sobre la Herencia en Java, se pueden visitar las siguientes referencias web:

<http://codejavu.blogspot.com.es/2013/05/herencia-en-java.html>

<https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/>



### VIDEO DE INTERÉS

A continuación, se dispone de varios enlaces a vídeos acerca de la herencia en Java

[https://www.youtube.com/watch?v=vu\\_PXCOCF3I](https://www.youtube.com/watch?v=vu_PXCOCF3I)

<https://www.youtube.com/watch?v=CYNLhg42O9c>

## 8. LIBRERÍAS DE CLASES. CREACIÓN. INCLUSIÓN Y USO DE LA INTERFACE

En Java y en varios lenguajes de programación más, existe el concepto de librería. Una **librería** en Java se puede entender como un conjunto de clases, que poseen una serie de métodos y atributos. Lo realmente interesante de estas librerías para Java es que facilitan muchas operaciones. De una forma más completa, las librerías en Java permiten **reutilizar código**, es decir que se puede hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar esas funcionalidades.



La **biblioteca estándar de Java** está compuesta por cientos de clases como System, String, Scanner, ArrayList, HashMap, etc. que nos permiten hacer casi cualquier cosa. Para programar en Java se tiene que recurrir continuamente a consultar la documentación del API de Java.

Los nombres de las librerías responden a un **esquema jerárquico** y se basan en la **notación de punto**.

Ejemplo: Por ejemplo, el nombre completo para la clase ArrayList sería `java.util.ArrayList`. Se permite el uso de `*` para nombrar a un conjunto de clases. Por ejemplo, `java.util.*` hace referencia al conjunto de clases dentro del paquete `java.util`, donde tenemos ArrayList, LinkedList y otras clases.

Para utilizar las librerías del API, existen dos situaciones:

- a) Hay librerías o clases que se usan siempre pues constituyen elementos fundamentales del lenguaje Java como la clase String. Esta clase, perteneciente al paquete `java.lang`, se puede utilizar directamente en cualquier programa Java ya que se carga automáticamente.
- b) Hay librerías o clases que no siempre se usan. Para usarlas dentro de nuestro código hemos de indicar que requerimos su carga mediante una sentencia `import` incluida en cabecera de clase.

Ejemplo: Por ejemplo, `import java.util.ArrayList;` es una sentencia que incluida en cabecera de una clase nos permite usar la clase ArrayList del API de Java. Escribir `import java.util.*;` nos permitiría cargar todas las clases del paquete `java.util`.



### PARA SABER MÁS

Se recomienda visitar la web de Oracle para conocer toda la jerarquía de paquetes de la API de Java:

<https://docs.oracle.com/javase/10/docs/api/index.html?overview-summary.html>

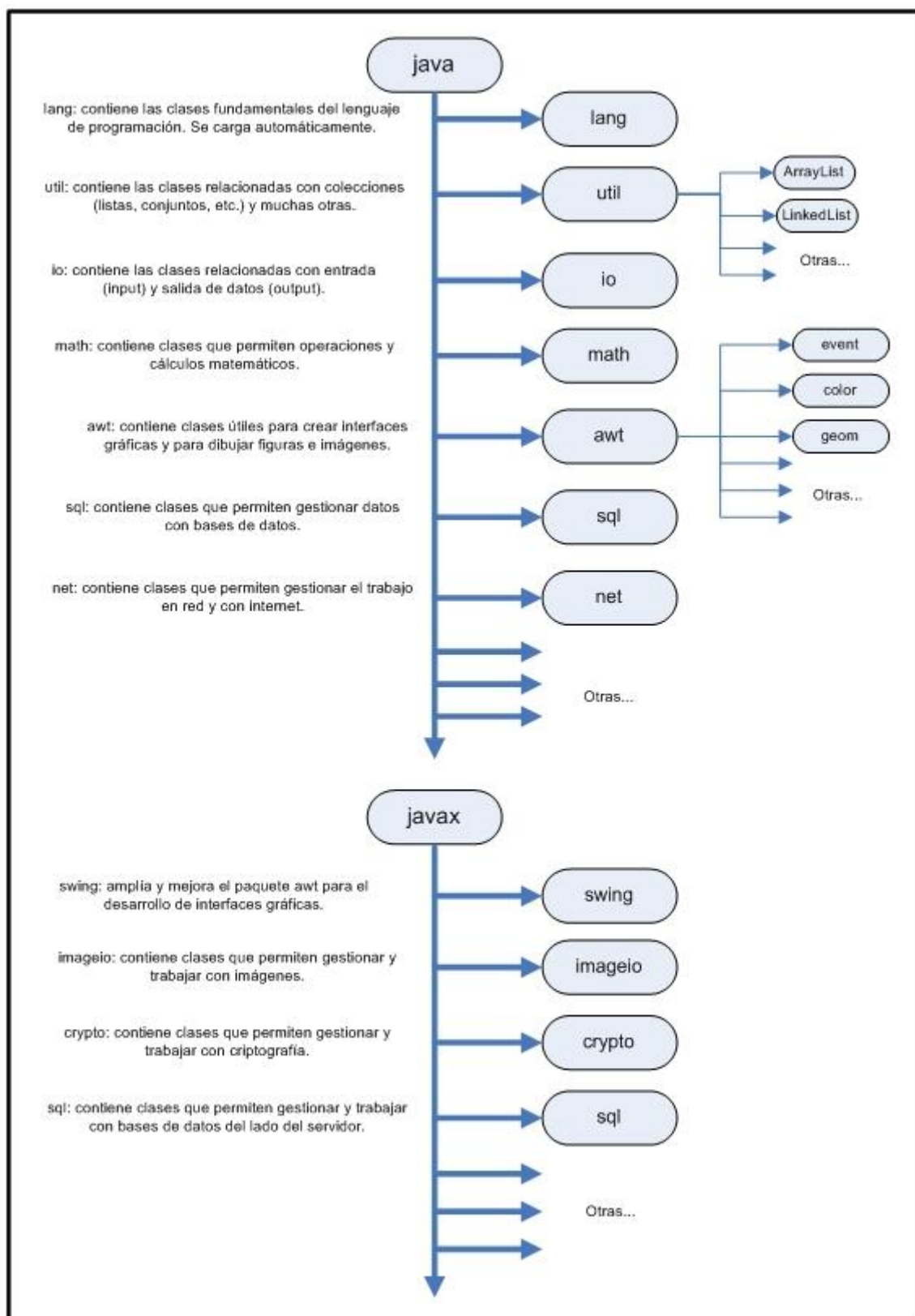


Imagen: Esquema orientativo de la organización de librerías en el API de Java

## Creación de clases

Lo más interesante de las librerías en Java, es que cualquier programador puede crearlas y hacer uso de ellas en sus proyectos. Básicamente un paquete en Java puede ser una librería, sin embargo una librería Java completa puede estar conformada por muchos paquetes más. Al importar un paquete podemos hacer uso de las clases, métodos y atributos que lo conforman.

Para crear un paquete en Java recomendamos seguir los siguientes pasos:

1. Poner un **nombre** al paquete.
2. Crear una **estructura jerárquica de carpetas** equivalente a la estructura de subpaquetes. La ruta de la raíz de esa estructura jerárquica deberá estar especificada en el **ClassPath** de Java.
3. Especificar a qué paquete pertenecen la clase (o clases) del archivo .java mediante el uso de la sentencia **package**.

## RESUMEN FINAL

La Programación Orientada a Objetos constituye una buena opción a la hora de resolver un problema, sobre todo cuando éste es muy extenso. En este enfoque de programación, se facilita evitar la repetición de código, no sólo a través de la creación de clases que hereden propiedades y métodos de otras, sino además a través de que el código es reutilizable por sistemas posteriores que tengan alguna similitud con los ya creados.

En esta unidad se ha comenzado analizando la programación orientada a objetos para posteriormente estudiar las unidades de modularidad de la POO: clases y objetos, así como las relaciones entre ellas. A continuación, se han identificado los fundamentos básicos de este enfoque y uno de los principales conceptos del paradigma: la herencia.

Finalmente se ha tratado el concepto de librería de clase y cómo se crean esas librerías y esas clases.