

UNIDAD DIDÁCTICA 4

DESARROLLO DE CLASES

**MÓDULO PROFESIONAL:
PROGRAMACIÓN**



CESUR
Tu Centro Oficial de FP

Índice

RESUMEN INTRODUCTORIO	2
INTRODUCCIÓN	2
CASO INTRODUCTORIO.....	2
1. CLASES Y OBJETOS EN JAVA	4
1.1 Clase.....	4
1.2 Atributos	6
1.3 Métodos.....	6
1.4 Constructores.....	9
2. OBJETO.....	15
2.1 Mensaje.....	17
2.2 Declarar un objeto.....	18
2.3 Ejemplarizar una clase.....	18
2.4 Inicializar un objeto	19
3. CONCEPTO DE HERENCIA. TIPOS. UTILIZACIÓN DE CLASES HEREDADAS	27
3.1 Relación de asociación («uso», usa, cualquier otra relación)	28
3.2 Relación de herencia (generalización / especialización, es un)	32
3.3 Relación de agregación (todo / parte, forma parte de)	36
3.4 Relación de composición.....	42
4. LIBRERÍAS DE CLASES. CREACIÓN. INCLUSIÓN Y USO DE LA INTERFACE	46
RESUMEN FINAL	51

RESUMEN INTRODUCTORIO

A lo largo de esta unidad veremos la definición de clase y objeto y la diferencia que existe entre estos dos conceptos en Java. 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.

Conocer y manejar el desarrollo de clases en Java es fundamental en la programación orientada a objetos (POO). Las clases permiten reutilizar código, abstraer complejidades, organizar el programa de manera modular, aplicar herencia y polimorfismo, y garantizar la encapsulación de datos, promoviendo así la mantenibilidad y depuración eficiente del software. Es esencial para crear aplicaciones eficientes, escalables y fáciles de mantener.

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 el estudio de la unidad, sabrás identificar los principales atributos y métodos que representan a una clase, serás capaz de modelar cualquier objeto del mundo real utilizando el concepto de clase y sus elementos, conocerás los distintos tipos de relaciones existentes entre los objetos.

1. CLASES Y OBJETOS EN JAVA

Para guiar a tu equipo en la implementación del nuevo proyecto en Java, decides realizar una formación de una semana, en la que tratarás los temas fundamentales de la POO en Java. Lo más importante es saber cómo crear una clase, añadirle atributos y crear métodos que realicen la funcionalidad esperada. También introduces el concepto de constructor para que en las próximas clases puedas entrar de lleno con la explicación de creación de objetos en Java.

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.



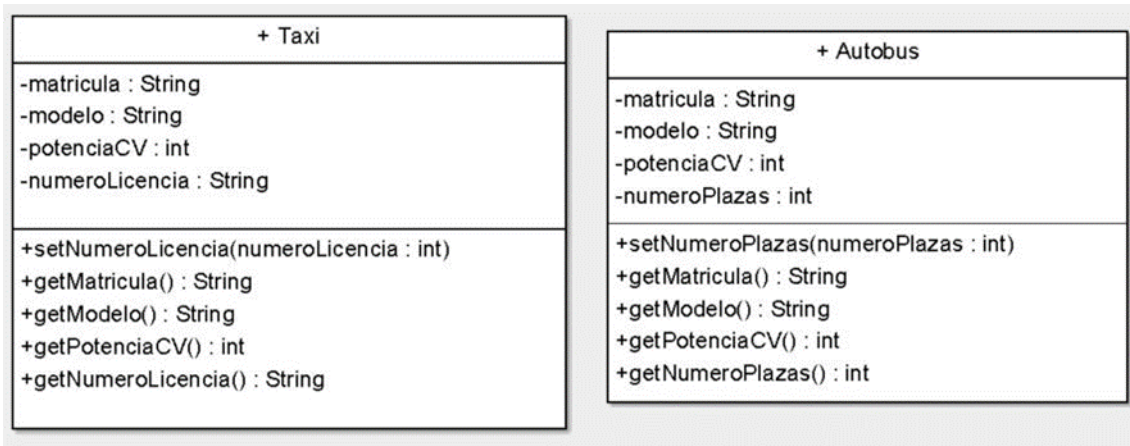
¿SABÍAS QUE...?

La Programación Orientada a Objetos (POO) es un paradigma de programación que ha sido ampliamente adoptado en diversos lenguajes de programación, como Java, C++, Python y C#. En todos ellos, los conceptos fundamentales de la POO, como clases, objetos, herencia, encapsulamiento y polimorfismo, son parte integral de su sintaxis y funcionalidad.

1.1 Clase

En la unidad anterior se ha visto el concepto de clase como la agrupación o colección de objetos que comparten una estructura común y un comportamiento común formada por atributos y métodos, pero desde un punto de vista conceptual mediante los diagramas de clase UML.

A modo de repaso se muestra la siguiente imagen que representa una clase Taxi y otra clase Autobus con sus correspondientes atributos y métodos en notación UML.



Clases en notación UML

Fuente: Elaboración propia.

Para representar estas clases, dadas en notación UML, a su correspondiente código Java, debemos utilizar unas reglas y sintaxis.

Toda clase comienza con una llave y cierra con una llave, todo lo que hay en el interior pertenece a la clase. En un mismo archivo con extensión .java pueden existir más de una clase (no recomendado), pero normalmente tendremos una sola clase en un archivo dependerá de los requerimientos software, etc.

Las clases se componen de atributos, constructores y métodos.

```
public class NombreClase {
    //atributos

    //constructores

    //métodos
}
```

Estructura de una clase en Java

Fuente: Elaboración propia



RECUERDA

En Java, es una convención que el nombre del archivo que contiene una clase coincida exactamente con el nombre de la clase y que los nombres de clases comiencen con la primera letra mayúsculas (notación UpperCamelCase o PascalCase).

1.2 Atributos

Como se explicó en la unidad anterior un atributo es una característica de un objeto que permite describir y almacenar el estado del objeto al que pertenecen.

En Java la sintaxis para declarar un atributo es la siguiente:

[modificador] tipoDeDato nombreDeAtributo;

Ej: **private** String **matricula**;

- [modificador]: es un modificador de acceso opcional que determina la visibilidad y el alcance del atributo. Puede ser public, protected, private o no tener un modificador, lo que indica que el atributo es public.
- tipoDeDato: es el tipo de dato del atributo, como int, double, String, etc. También puede ser el nombre de una clase si el atributo es de tipo objeto.
- nombreDeAtributo es el nombre que se le asigna al atributo.

1.3 Métodos

Igualmente también se trató en la unidad anterior el concepto de método. Los métodos son bloques de código que se definen dentro de una clase y que contienen instrucciones para realizar una tarea específica.

Los métodos en Java tienen la siguiente sintaxis básica:

```
[modificador] tipoRetorno nombreMetodo([parámetros]) {  
    // Instrucciones a ejecutar  
    // Retorno de valor (si se ha establecido un tipoRetorno)  
}
```

```
Ej: public int sumar(int a, int b) {  
    int resultado = a + b;  
    return resultado;  
}
```

- [modificador]: es un modificador de acceso opcional que define la visibilidad del método. Puede ser public, private, protected o no tener ningún modificador (en este caso, es public).
- tipoRetorno es el tipo de dato que el método devuelve como resultado (en este caso debe incorporar una instrucción return de retorno de valor), o void si no devuelve ningún valor.
- nombreMetodo es el nombre dado al método, que se utiliza para invocarlo posteriormente.
- [parámetros]: son los valores de entrada que puede recibir el método. Pueden ser opcionales y si hay más de uno, se separan por comas.

En el ejemplo del método anterior, el método se llama sumar, tiene dos parámetros a y b de tipo int y devuelve un valor de tipo int. En el cuerpo del método, se realiza la suma de a y b, y el resultado se asigna a la variable resultado. Luego, se utiliza la instrucción return para devolver el valor resultante.

Los métodos pueden tener diferentes propósitos y funcionalidades, como realizar cálculos, modificar el estado de un objeto, interactuar con otros objetos, realizar operaciones de entrada/salida, entre otros. También pueden tener modificadores adicionales como static, final, abstract, etc., que proporcionan diferentes comportamientos y restricciones.

Un método **setter**, es un método utilizado para **establecer** el valor de un atributo privado de una clase (los atributos privados solo pueden ser accedidos por métodos de la misma clase en la que están declarados). Para reconocerlos fácilmente se les suele añadir el prefijo 'set' al principio del nombre del método. Estos métodos recibirán uno o más parámetros.

Un método **getter**, es un método utilizado para **retornar** el valor de un atributo privado de una clase (los atributos privados solo pueden ser accedidos por métodos de la misma clase en la que están declarados). Estos métodos no suelen recibir parámetros.



EJEMPLO PRÁCTICO

Ejemplo de la clase **Autobus** en Java que se debe guardar en el archivo **Autobus.java**. El método **setNumeroPlazas(int numeroPlazas)** es un método setter que se encarga de insertar el valor recibido como parámetro dentro de un atributo de la clase.

La clase **Autobus** tiene 5 métodos (4 métodos getter y un método setter).
Analiza y añade el resto de métodos setter que faltan.

```
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

Ejemplo de la clase `Taxi` en Java, almacenada en el archivo `Taxi.java`. La clase `Taxi` tiene 5 métodos (4 métodos getter y un método setter).

Analiza y añade el resto de métodos setter que faltan.

```
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;  
    }  
}
```

1.4 Constructores

Un **constructor** es uno de los métodos más importantes que se encuentra en una clase y que se ejecuta la primera vez que se crea o instancia un objeto. Por convención el método constructor debe tener el mismo nombre de la clase a la que pertenece el objeto. Es un método que recibe cero o más parámetros y lo usual es que inicialice los valores de los atributos del objeto por primera vez.

En lenguajes como Java se puede definir más de un método constructor, que normalmente se diferencian entre sí por la cantidad o tipo diferente en los parámetros que reciben.

Para crear o instanciar un objeto se utiliza el operador `new` seguido del nombre del constructor (que puede tener parámetros o no) si no hay definido constructor entonces se usa el constructor por defecto (nombre de la clase seguido de dos paréntesis Ej: `NombreClase()`).

Cuando en una clase no definimos un constructor, Java le asigna uno llamado por defecto (default), este constructor no tiene argumentos. En el ejemplo anterior almacenado en el archivo `Calculadora.java`, nos fijamos en la siguiente instrucción dentro del método `main` incluido en la clase `Main`,

```
Calculadora calculadora = new Calculadora();
```

se ha usado el constructor por defecto (no había sido definido en la clase `Calculadora`, pero existe por defecto), se utiliza para crear un objeto de la clase `Calculadora` y así poder acceder a los atributos y métodos del objeto.

Cuando se invoca al constructor por defecto (sin parámetros) al crear un objeto y no se inicializan los atributos dentro de este constructor, los atributos se inicializan a los valores por defecto correspondientes a su tipo, esto es, para números enteros se inicializan a cero, para reales a 0.0, de tipo boolean a `false` y para tipo carácter o `String` a `null`.

Características de un constructor de objetos:

- Primer método que se ejecuta cuando se crea un objeto.
- No puede retornar datos
- Solo se ejecuta una sola vez cuando se crea el objeto.
- El objetivo de este método es inicializar atributos o realizar cualquier validación oportuna antes de inicializar los datos.
- Pueden coexistir varios constructores (mismo nombre ~ sobrecarga de métodos) dentro de una misma clase, pero deben diferir en el número o tipo de parámetros.
- El constructor por defecto no tiene parámetros y no hace falta declararlo salvo cuando ya se han declarado otros constructores, entonces, para poder usarlo sí es necesario declararlo explícitamente.

Método destructor 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.



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. En este caso, 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).

En el ejemplo de la clase Autobus y Taxi anterior, no se ha declarado constructor con parámetros en ninguna de ellas. Esto quiere decir, que por defecto existirá el constructor sin parámetros que podríamos utilizar para instanciar objetos, y sería el siguiente...

Para la clase Taxi, el constructor sin parámetros (por defecto, aunque no se haya declarado):

```
public Autobus() {  
  
}
```

Para la clase Taxi, el constructor sin parámetros (por defecto, aunque no se haya declarado):

```
public Taxi() {  
  
}
```



EJEMPLO PRÁCTICO

Declaración de constructores (el primero sin parámetros y un segundo con cuatro parámetros) de la clase Taxi en Java. En el ejemplo no se han mostrado los atributos ni otros métodos, solo constructores.

```
public Taxi() {  
  
}  
  
public Taxi(String matricula, String modelo,  
            int potenciaCV, String numeroLicencia) {  
  
    this.matricula = matricula;  
    this.modelo = modelo;  
    this.potenciaCV = potenciaCV;  
    this.numeroLicencia = numeroLicencia;  
}
```



EJEMPLO PRÁCTICO

Definición de constructores (el primero sin parámetros y un segundo con cuatro parámetros) de la clase Autobus en Java. En el ejemplo no se han mostrado los atributos ni otros métodos, solo constructores.

```
//constructor sin parámetros  
public Autobus() {  
  
}  
  
//constructor con todos los parámetros  
public Autobus(String matricula, String modelo,  
               int potenciaCV, int numeroPlazas) {  
    this.matricula = matricula;  
    this.modelo = modelo;  
    this.potenciaCV = potenciaCV;  
    this.numeroPlazas = numeroPlazas;  
}
```



NOTA DE INTERÉS

Completa las clases Taxi y Autobus, añadiendo los atributos y métodos setter y getter que faltan, te servirá de guía fijarte en los parámetros que incluye cada constructor.



EJEMPLO PRÁCTICO

Ejemplo de clase Vehiculo en Java, que se debe guardar en el archivo Vehiculo.java. En este ejemplo se han mostrado atributos, constructor con parámetros y dos métodos de tipo getter.

Añade el resto de métodos getter y setter que harían falta.

```
public class Vehiculo {  
    //atributos  
    private String matricula;  
    private String modelo;  
    private int potenciaCV;  
    //constructor  
    public Vehiculo(String matricula, String modelo, int pCV) {  
        this.matricula=matricula;  
        this.modelo=modelo;  
        this.potenciaCV=pCV;  
    }  
    //métodos  
    public String getMatricula() {  
        return matricula;  
    }  
  
    public String getModelo() {  
        return modelo;  
    }  
}
```

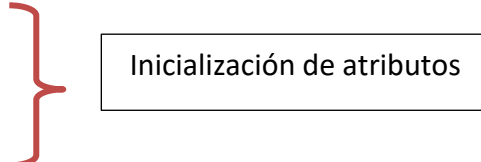
La clase Vehiculo declarada anteriormente tiene 3 atributos que son, matricula, modelo y potenciaCV, que tienen un tipo de determinado (String e int). El método constructor de esta clase es el método (que tiene el mismo nombre que la clase):

Vehiculo(String matricula, String modelo, int pCV)

Recibe 3 parámetros que se corresponden con los atributos declarados en la clase (pueden tener mismo nombre o diferente a los atributos de la clase). En el momento de la creación o instanciación de un objeto de tipo Vehiculo, se reciben los valores de estos parámetros y el método constructor se encarga de almacenarlos en sus correspondientes atributos dentro del objeto, es decir, inicializa los atributos con los valores recibidos. Este comportamiento se habrá indicado mediante el código que se incluye dentro del constructor.

Antes de asignar los valores recibidos por el constructor a los atributos del objeto, se puede realizar algún tipo de comprobación de los datos y en caso de que no sean datos válidos no realizar la inicialización (dependerá de las especificaciones del problema a resolver).

```
public Vehiculo(String matricula, String modelo, int pCV) {  
    this.matricula=matricula;  
    this.modelo=modelo;  
    this.potenciaCV=pCV;  
}
```



La palabra clave **this** se utiliza en Java como una referencia al objeto actual (con el que se está trabajando), se puede utilizar para hacer referencia a los atributos y métodos de un objeto que pertenece a una clase. Se usa para distinguir entre los atributos de la instancia actual y los parámetros o variables locales que puedan tener el mismo nombre

```
this.matricula=matricula
```

De esta forma se indica que el **parámetro** llamado **matricula**, recibido en el constructor, se va a almacenar en el **atributo** del objeto que tiene el mismo nombre **matricula** (this se usa con atributos y métodos) que se ha instanciado o creado.

Entonces, después de la creación del objeto usando su constructor con parámetros, el objeto tendrá los atributos matricula, modelo y potenciaCV inicializados y con un valor.

Por último tenemos dos métodos más, cuyos nombres empiezan por "get.." (**getter**) y se encargan de retornar el valor almacenado en un atributo de la clase.

- El método **String getMatricula()** retorna el valor que tenga almacenado el atributo matricula del objeto.
- El método **String getModelo()** retorna el valor que tenga almacenado el atributo modelo del objeto.



VÍDEO DE INTERÉS

Para comprender qué es una clase en Java, formato y reglas para crearlas visualiza este vídeo.



2. OBJETO

Como jefe de proyecto en tu empresa, te enfrentas al desafío de explicar cómo la programación orientada a objetos, con sus conceptos clave como mensaje, declaración de objetos, ejemplarización de una clase e inicialización de un objeto, será fundamental para conseguir el éxito.

La asimilación de estos conceptos serán la base de este emocionante proyecto software, lo cual garantizará que el equipo modele de manera efectiva los elementos clave del sistema.

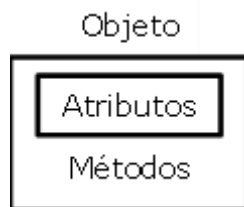
Un objeto es una entidad caracterizada por sus **atributos** propios y cuyo comportamiento está determinado por las **acciones** o métodos que pueden modificarlo, así como también las acciones que requiere de otros objetos. Un objeto tiene identidad e inteligencia y constituye una unidad que **oculta** 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.

Cada objeto puede ser considerado como un proveedor de servicios utilizado por otros objetos que son sus clientes. Cada objeto puede ser a la vez proveedor y cliente. De ahí

que un programa pueda ser visto como un conjunto de relaciones entre proveedores clientes. Los servicios ofrecidos por los objetos son de dos tipos:

1. Los datos, que llamamos **atributos**.
2. Las acciones o funciones, que llamamos **métodos**



El **proceso** que se realiza cuando se **crea un objeto** es el siguiente:

- Asignación de memoria al objeto.
- Invocación de un constructor.
- Inicialización de atributos del objeto.

Características Generales:

- **Un objeto se identifica por un nombre o un identificador único que lo diferencia de los demás.** Ejemplo: el objeto Cuenta de Ahorros número 12345 es diferente al objeto Cuenta de Ahorros número 25789. En este caso el identificador que los hace únicos es el número de la cuenta.
- **Un objeto posee estados.** El estado de un objeto está determinado por los valores que poseen sus atributos en un momento dado.
- **Un objeto tiene un conjunto de métodos.** El comportamiento general de los objetos dentro de un sistema se describe o representa mediante sus operaciones o métodos. Los métodos se utilizarán para obtener o cambiar el estado de los objetos, así como para proporcionar un medio de comunicación entre objetos.
- **Un objeto tiene un conjunto de atributos.** Los atributos de un objeto contienen valores que determinan el estado del objeto durante su tiempo de vida. Se implementan con variables, constantes y estructuras de datos (similares a los campos de un registro).
- **Los objetos soportan encapsulamiento.** La estructura interna de un objeto normalmente está oculta a los usuarios de este. Los datos del objeto están

disponibles solo para ser manipulados por los propios métodos del objeto. El único mecanismo que lo conecta con el mundo exterior es el paso de mensajes.

- **Un objeto tiene un tiempo de vida dentro del programa o sistema que lo crea y utiliza.** Para ser utilizado en un programa el objeto debe ser creado con una instrucción particular (New) y al finalizar su utilización es destruido con el uso de otra instrucción o de manera automática.
- **Un objeto es una instancia de una clase.** Cada objeto concreto dentro de un sistema es miembro de una clase específica y tiene el conjunto de atributos y métodos declarados en la misma.



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.

2.1 Mensaje

Es la petición de un objeto a otro para solicitar la ejecución de alguno de sus métodos o para obtener el valor de un atributo público.

Estructuralmente, un mensaje consta de 3 partes:

- **Identidad del receptor:** Nombre del objeto que contiene el método a ejecutar.
- **Nombre del método a ejecutar:** Solo los métodos declarados públicos.
- **Lista de Parámetros** que recibe el método (cero o más parámetros)

Su sintaxis es la siguiente:

```
<Variable_Objeto>.<Nombre_Método> ( [<Lista de Parámetros> ] );
```

Cuando el objeto receptor recibe el mensaje, comienza la ejecución del código contenido dentro del método invocado, recibiendo y/o devolviendo los valores de los parámetros correspondientes, si los tiene, ya que son opcionales: ([]).

```
System.out.println("Plazas disponibles en bus1: " +  
                    bus1.getNumeroPlazas());
```

En esta sentencia podemos ver que el objeto que contiene el método a ejecutar es bus1, el método a ejecutar es getNumeroPlazas(), este método no recibe parámetros pero devolverá un valor que será un número entero que corresponde al número de plazas del autobus.

2.2 Declarar un objeto

Declarar una variable para contener un objeto es exactamente igual que declarar una variable que va a contener un tipo primitivo.

```
tipo nombre
```

La declaración de un objeto aparece 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;
```

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**.

2.3 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. La siguiente sentencia, crea un objeto de la clase Date (clase del paquete java.util).

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

Esta sentencia realiza tres acciones declaración, ejemplarización e inicialización.



EJEMPLO PRÁCTICO

En este ejemplo se puede ver como integrar la sentencia anterior dentro de una clase en Java, usando un método main para que esta sea ejecutable.

```
import java.util.Date;

public class EjemploFechaActual {
    public static void main(String[] args) {
        // Declaración, ejemplarización e inicialización de la
        // clase Date
        Date fechaActual = new Date();
        System.out.println("Fecha actual: " + fechaActual);
    }
}
```

2.4 Inicializar un objeto

Las clases proporcionan métodos para inicializar los nuevos objetos de ese tipo. El proceso de inicialización de un objeto se puede realizar mediante un constructor o mediante un método llamado setter (visto anteriormente)

Inicialización de objetos mediante un constructor:

Una clase puede 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**. En el ejemplo práctico realizado con la clase Rectangle se ha realizado la creación del objeto rect e inicializado a la vez con su constructor de 4 parámetros,

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

Inicialización de objetos mediante un método conocido como setter:

También se puede inicializar un objeto usando un método setter. La diferencia de inicializar un objeto con el constructor a hacerlo con un método setter es que con el método constructor solo realizará la inicialización la primera vez que se crea el objeto, mientras que utilizando un método setter podríamos hacerlo siempre que sea necesario.

En el caso de que los atributos de la clase sean declarados como privados mediante el uso del modificador `private`, es indispensable usar los `setter` para asignar o cambiar el valor de los atributos de un objeto, durante la ejecución del programa, ya que estos serían los únicos que tienen el privilegio de acceder a estos atributos.



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);
```



PARA SABER MÁS

Visualiza este enlace para saber más sobre la clase `Rectangle`, métodos constructores y métodos concretos que forman parte de esta clase.



Aquí tienes un ejemplo de `setter/getter`, constructor sin parámetros y constructor con parámetros usados de forma conjunta. La clase `Autobus` está almacenada en el archivo `Autobus.java`

```
public class Autobus {  
    private String matricula;  
    private String modelo;  
    private int potenciaCV;  
    private int numeroPlazas;  
    //constructor sin parámetros  
    public Autobus() {  
    }  
    //constructor con todos los parámetros  
    public Autobus(String matricula, String modelo,  
                    int potenciaCV, int numeroPlazas) {  
        this.matricula = matricula;  
        this.modelo = modelo;  
        this.potenciaCV = potenciaCV;  
        this.numeroPlazas = numeroPlazas;  
    }  
    //getter y setter  
    public String getMatricula() {  
        return matricula;  
    }  
    public void setMatricula(String matricula) {  
        this.matricula = matricula;  
    }  
    public String getModelo() {  
        return modelo;  
    }  
    public void setModelo(String modelo) {  
        this.modelo = modelo;  
    }  
    public int getPotenciaCV() {  
        return potenciaCV;  
    }  
    public void setPotenciaCV(int potenciaCV) {  
        this.potenciaCV = potenciaCV;  
    }  
    public int getNumeroPlazas() {  
        return numeroPlazas;  
    }  
    public void setNumeroPlazas(int numeroPlazas) {  
        this.numeroPlazas = numeroPlazas;  
    }  
}
```

El ejemplo anterior de la clase Autobus es más completo e incorpora un constructor sin parámetros, un constructor con todos los parámetros y los métodos getter y setter correspondientes a los atributos declarados en la clase. El constructor inicializa los atributos cuando se crea el objeto y con los métodos setter podremos inicializarlos o modificarlos en cualquier momento durante la ejecución del programa. Los métodos getter se usarán para extraer los datos almacenados en los atributos del objeto.

Podemos instanciar un objeto de la clase Autobus, de dos formas, bien usando el constructor sin parámetros o bien con el constructor con parámetros.

Instanciación utilizando constructor sin parámetros:

```
Autobus bus1 = new Autobus();
```

Lo que va a ocurrir es que se va a crear el objeto **bus1**, pero no vamos a poder pasarle datos al constructor (porque no tiene parámetros dicho constructor) para que se almacenen en los correspondientes atributos del objeto bus1, es decir, si en un momento dado queremos que por ejemplo, el usuario introduzca los datos del objeto bus1, por teclado y dichos datos queremos que se introduzcan en los atributos de este objeto, no tenemos forma de hacerlo en este caso, porque hemos usado el constructor sin parámetros.

¿Cómo podemos solucionarlo?, pues bien, como la clase Autobus tiene declarados los métodos setter para cada uno de sus atributos, entonces, los encargados de inicializar los valores de los atributos deben ser los métodos setter correspondientes a cada uno de los atributos.

Instanciación utilizando constructor con parámetros:

```
Autobus bus2 = new Autobus("5432PVG", "Volvo", 190, 54);
```

Ahora se ha usado el constructor con parámetros por tanto, mediante código incluido dentro del constructor con parámetros, establecemos que los parámetros recibidos sean almacenados en los correspondientes atributos. Esto será muy útil, cuando queremos preguntar al usuario por los datos del autobús, de esta forma los datos introducidos en el objeto serán los que haya introducido el usuario durante la ejecución del programa y no, unos datos fijos como se ve en este ejemplo. Ampliaremos sobre este caso en particular en próximas unidades.

En el próximo ejemplo se utiliza una clase con método main para instanciar objetos de la clase Autobus anterior, utilizando un constructor con parámetros y sin parámetros. También se utiliza un método setter para modificar un atributo después de la creación del objeto.


```
public class MainAutobus {
    public static void main(String[] args) {
        /*instanciación de objeto utilizando constructor
        * sin parámetros para el 1º bus*/
        Autobus bus1 = new Autobus();

        //inicialización de atributos objetos con setter
        bus1.setMatricula("1254GXP");
        bus1.setModelo("Mercedes B");
        bus1.setNumeroPlazas(12);
        bus1.setPotenciaCV(156);

        //visualización datos en pantalla
        System.out.println("Primer bus");
        System.out.println("-----");
        System.out.println("Matricula: " + bus1.getMatricula());
        System.out.println("Modelo: " + bus1.getModelo());
        System.out.println("PotenciaCV: " +
            bus1.getPotenciaCV());

        /*instanciación de objeto utilizando constructor
        con parámetros para el 2º bus*/
        Autobus bus2 = new Autobus("5432PVG", "Volvo", 190, 54);

        //visualización datos en pantalla
        System.out.println("\nSegundo bus");
        System.out.println("-----");
        System.out.println("Matricula: " + bus2.getMatricula());
        System.out.println("Modelo: " + bus2.getModelo());
        System.out.println("PotenciaCV: " +
            bus2.getPotenciaCV());

        //modificación de datos después de la creación del objeto
        bus1.setModelo("Volkswagen");

        //visualización datos en pantalla
        System.out.println("\n**Modelo modificado en primer
            bus**\n");
        System.out.println("Modelo: " + bus1.getModelo());

        //visualización datos del 2º bus modificado en pantalla
        System.out.println("\nprimer bus");
        System.out.println("-----");
        System.out.println("Matricula: " + bus1.getMatricula());
        System.out.println("Modelo: " + bus1.getModelo());
        System.out.println("PotenciaCV: " +
            bus1.getPotenciaCV());
    }
}
```


Salida en pantalla:

Primer bus

Matricula: 1254GXP

Modelo: Mercedes B

PotenciaCV: 156

Segundo bus

Matricula: 5432PVG

Modelo: Volvo

PotenciaCV: 190

****Modelo modificado en primer bus**\n**

Modelo: Volkswagen

primer bus

Matricula: 1254GXP

Modelo: Volkswagen

PotenciaCV: 156

Se puede observar que instanciar e inicializar un objeto a la vez (en una misma línea utilizando el constructor con parámetros) ahorra líneas de código. Si en algún momento después de la creación del objeto es necesario modificar algún dato solo hay que llamar al método setter correspondiente (ya no se podría usar el constructor nuevamente para inicializar el objeto porque el objeto ya está creado).



RECUERDA

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 (Sobrecarga de constructores). Cada constructor puede inicializar el nuevo objeto de una forma diferente.



ARTÍCULO DE INTERÉS

Se recomienda visitar el siguiente documento sobre uso de constructores, métodos y objetos.



EJEMPLO PRÁCTICO

Crea una nueva clase que contenga un método **main**(ejecutable) desde el que se instanciarán los objetos **vehiculo1** y **vehiculo2** de la clase **Vehiculo**, utiliza el operador **new** e invoca al **constructor** de la clase y pasa los parámetros que se definieron en su momento al declarar el constructor. Una forma de ejecutar la clase es haciendo clic derecho encima de **MainVehiculo.java**, escogiendo del menú **Run as** y a continuación **Java application**.

```
public class MainVehiculo {  
    public static void main(String[] args) {  
        Vehiculo vehiculo1 = new Vehiculo("2345BCD", "Porsche 911  
                                           540");  
        Vehiculo vehiculo2 = new Vehiculo("8765FCD", "Fiat Panda", 69);  
  
        System.out.println("Matrícula "+vehiculo1.getModelo()+" - "  
                           +vehiculo1.getMatricula()  
                           +vehiculo1.getPotenciaCV());  
        System.out.println("Matrícula "+vehiculo2.getModelo()+" - "  
                           +vehiculo2.getMatricula()  
                           +vehiculo2.getPotenciaCV());  
    }  
}
```

La salida por pantalla será:

```
Matrícula Porsche 911 - 2345BCD540  
Matrícula Fiat Panda - 8765FCD69
```



EJEMPLO PRÁCTICO

Ejemplo de creación (instanciación) de objetos en Java de la clase Autobus y Taxi con el constructor por defecto(sin parámetros) y usando métodos setter. La clase MainTransporte puede almacenarse en el archivo MainTransporte.java

```
public class MainTransporte {  
    public static void main(String[] args) {  
  
        //instanciación de objetos  
        Autobus bus1 = new Autobus();  
        Autobus bus2 = new Autobus();  
  
        Taxi taxi1 = new Taxi();  
        Taxi taxi2 = new Taxi();  
  
        //inicialización de atributos objetos de tipo Autobus  
        bus1.setNumeroPlazas(45);  
        bus2.setNumeroPlazas(20);  
  
        //inicialización de atributos objetos de tipo Taxi  
        taxi1.setNumeroPlazas(5);  
        taxi2.setNumeroPlazas(7);  
  
        System.out.println("Plazas disponibles en bus1: "+  
                           bus1.getNumeroPlazas());  
        System.out.println("Plazas disponibles en bus2: "+  
                           bus2.getNumeroPlazas() + "\n");  
  
        System.out.println("Plazas disponibles en taxi1: "+  
                           taxi1.getNumeroPlazas());  
        System.out.println("Plazas disponibles en taxi2: "+  
                           taxi2.getNumeroPlazas());  
    }  
}
```

Salida por pantalla:

```
Plazas disponibles en bus1: 45  
Plazas disponibles en bus2: 20
```

```
Plazas disponibles en taxi1: 5  
Plazas disponibles en taxi2: 7
```



RECUERDA

Para poder invocar a un determinado método desde un objeto, antes, éste debe estar declarado en la clase de la cual se instancia (crea) el objeto. El objeto se podrá instanciar, bien usando el/los constructor/es con parámetros (también debe estar declarado) o bien el constructor por defecto (sin parámetros).

3. CONCEPTO DE HERENCIA. TIPOS. UTILIZACIÓN DE CLASES HEREDADAS

Mientras presentas el enfoque de programación orientada a objetos en Java a tu equipo, es importante destacar cómo Java va más allá de las simples clases y objetos. Les explicas que, además de las clases y objetos que hemos discutido previamente, Java proporciona herramientas avanzadas para manejar las relaciones entre los objetos. Les explicarás que en este sistema, la relación de herencia será fundamental para modelar las entidades que comparten características comunes. También profundizarás en la relación entre un objeto principal (contenedor) y sus objetos secundarios (contenidos), diferenciando entre la relación de composición que es una relación más fuerte con una dependencia más estrecha entre objetos y la agregación que es una relación más débil y con más independencia entre ellos.

Las clases en Java no suelen operar de forma independiente, sino que están diseñadas para relacionarse entre sí, permitiendo compartir atributos y métodos.

La capacidad de establecer jerarquías entre clases es una característica fundamental que distingue la programación orientada a objetos de los enfoques tradicionales. Esta característica facilita la extensión y reutilización del código existente, evitando la necesidad de reescribirlo repetidamente.

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

Asociación

(entre otras, la relación
Usa-a)

Herencia

(Generalización /
Especialización o Es-un)

Agregación

(Todo / Parte o Forma-
parte-de)

Composición

(Es parte elemental de)

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

Una relación de asociación 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 que será utilizada en una relación de asociación.

```
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;  
        this.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) {  
        this.dni = dni;  
    }  
}
```



EJEMPLO PRÁCTICO

Ejemplo de clase CuentaBancaria en Java. Existe una relación de **Asociación** entre la clase Cliente anterior y la clase CuentaBancaria. Esta relación podría ser de composición o agregación dependerá de la dependencia de una clase con respecto a la otra según el contexto y el diseño específico de la aplicación a realizar.

```
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;  
    }  
}
```

Ejemplo de clase CuentaBancaria en Java. Existe una relación de Asociación entre la clase Cliente anterior y la clase CuentaBancaria. Esta relación podría ser de composición o agregación dependerá de la dependencia de una clase con respecto a la otra según el contexto y el diseño específico de la aplicación a realizar.

En esta relación la clase CuentaBancaria usa la clase Cliente. Por tanto primero tendremos que instanciar un objeto de la clase Cliente y pasarle los datos nombre, apellidos y dni dentro de su constructor. A continuación, instanciamos un objeto de la clase CuentaBancaria y le pasamos los datos de la cuenta, que serían el número de cuenta, el saldo y el objeto cliente (objeto instanciado de la clase Cliente que ya tiene datos).



EJEMPLO PRÁCTICO

Este ejemplo muestra cómo se crea un objeto de tipo Cliente, se inicializa con datos y se pasa como parámetro en el constructor de la clase CuentaBancaria.

```
public class MainCuentaBancaria {  
    public static void main(String[] args) {  
        Cliente cliente1 = new Cliente("Marta", "López", "44635695E");  
        CuentaBancaria cuenta1 = new CuentaBancaria("122", 10000,  
                                                    cliente1);  
  
        System.out.println("Datos de la cuenta bancaria:");  
        System.out.println("-----\n");  
        System.out.println("Número de cuenta: "+  
                           cuenta1.getNumCuenta() + "\n");  
        System.out.println("Saldo: "+ cuenta1.getSaldo() + "\n");  
        System.out.println("\nDatos del titular:");  
        System.out.println("-----\n");  
        System.out.println("Nombre: "+  
                           cuenta1.getTitular().getNombre() + "\n");  
        System.out.println("Apellidos: "+  
                           cuenta1.getTitular().getApellidos() + "\n");  
        System.out.println("DNI: "+ cuenta1.getTitular().getDni()  
                           + "\n");  
    }  
}
```




ENLACE DE INTERÉS

En este enlace podrás ver más ejemplos sobre las relaciones de asociación en la POO visitando la web:



3.2 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.

En Java para indicar que una clase hereda de otra(extiende) se utiliza la palabra reservada `extends` en la declaración de la clase. En el constructor de la clase hija (la clase que hace uso de `extends`) pasamos todos los parámetros tanto los que tuviera la clase hija(propios) como los que hereda del padre. Después, dentro del constructor de la clase hija haremos la inicialización de atributos propios y una llamada al constructor del padre para pasarle los parámetros que solo correspondan a la clase padre para que los almacene.



EJEMPLO PRÁCTICO

Traduce el diagrama de clases con notación UML realizado en la unidad anterior a código Java. Se trataba del diseño de un sistema de gestión de animales que puede aplicarse por ejemplo, en un zoológico, una protectora de animales, etc. Empieza creando una clase Animal que será superclase de todas las demás. Define sus atributos, constructor con parámetros, getter y setter.

```
public class Animal {  
    private String nombre;  
    private int edad;  
    private String genero;  
    private String tamanho;  
  
    public Animal(String nombre, int edad, String genero,  
                  String tamanho) {  
        this.nombre = nombre;  
        this.edad = edad;  
        this.genero = genero;  
        this.tamanho = tamanho;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public int getEdad() {  
        return edad;  
    }  
  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
  
    public String getGenero() {  
        return genero;  
    }  
  
    public void setGenero(String genero) {  
        this.genero = genero;  
    }  
  
    public String getTamanho() {  
        return tamanho;  
    }  
}
```

```
    }  
  
    public void setTamanho(String tamanho) {  
        this.tamanho = tamanho;  
    }  
}
```



EJEMPLO PRÁCTICO

En este caso concreto, nuestra clase Mamífero posee atributos propios y hereda de la clase Animal 3 atributos(nombre, edad, genero y tamanho), el valor de estos atributos los recoge en el momento de la creación de un objeto de tipo Mamífero mediante el constructor con parámetros, por tanto, dentro este constructor de la clase hija(Mamífero) hay que invocar al constructor de la clase padre para pasarle estos datos(llamada a **super(nombre, numPatas, tamanho)**) y que el constructor de la clase padre(Animal) los almacene porque son atributos que ella gestiona.

```
package ud4.animal.mamifero;  
  
public class Mamifero extends Animal {  
  
    //atributos propios de la clase Mamifero  
    private int gestacion;  
    private int cantidadPatas;  
    private boolean tienePelo;  
  
    //constructor con parámetros  
    public Mamifero(String nombre, int edad, String genero,  
                    String tamanho, int gestacion, int cantidadPatas,  
                    boolean tienePelo) {  
  
        //llamada a constructor clase padre(Animal)  
        super(nombre, edad, genero, tamanho);  
  
        //inicialización atributos propios de clase Mamifero  
        this.gestacion = gestacion;  
        this.cantidadPatas = cantidadPatas;  
        this.tienePelo = tienePelo;  
    }  
    //getter y setter  
    public int getGestacion() {  
        return gestacion;  
    }  
  
    public void setGestacion(int gestacion) {  
        this.gestacion = gestacion;  
    }  
}
```

```
public int getCantidadPatas() {  
    return cantidadPatas;  
}  
  
public void setCantidadPatas(int cantidadPatas) {  
    this.cantidadPatas = cantidadPatas;  
}  
  
public boolean isTienePelo() {  
    return tienePelo;  
}  
  
public void setTienePelo(boolean tienePelo) {  
    this.tienePelo = tienePelo;  
}  
}
```



PARA SABER MÁS

Accede a este enlace para ampliar tus conocimientos sobre la Herencia en Java, se pueden visitar las siguientes referencias web:



ENLACE DE INTERÉS

En este enlace puedes consultar más información sobre la herencia en programación orientada a objetos por medio de ejemplos





¿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**.



VÍDEO DE INTERÉS

Consulta este vídeo muy interesante sobre ejercicios para practicar la herencia y la abstracción



VÍDEO DE INTERÉS

En este vídeo podrás seguir practicando la herencia a través de ejercicios



3.3 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.

Características:

- Es una relación débil.
- La clase contenedora o todo tiene una referencia a la clase contenida o parte.
- La vida útil de la clase contenida no depende de la clase contenedora.
- La clase contenida o parte puede estar relacionada con más de una clase contenedora o todo.
- En UML se representaba con un simbolo de rombo vacío (sin color) próximo a la clase contenedora o todo.

Por ejemplo, un aula puede tener varios estudiantes. La relación entre el aula y los estudiantes es de agregación, ya que los estudiantes pueden existir independientemente del aula y pueden estar relacionados con otras aulas.



EJEMPLO PRÁCTICO

Tu nuevo cliente que es dueño de una cadena de hoteles te ha encargado una aplicación que gestione las habitaciones de que dispone cada hotel.

Primero comienzas por crear una clase Habitación con los atributos número, área y tipo, donde tipo, se refiere por ejemplo a si es de tipo suite, individual, doble, etc. Para todos los atributos creas sus correspondientes métodos getters y setters, así como un constructor con parámetros.

```
public class Habitacion {  
  
    private int numero;  
    private double area;  
    private String tipo;  
  
    public Habitacion(int numero, double area, String tipo) {  
        this.numero = numero;  
        this.area = area;  
        this.tipo = tipo;  
    }  
  
    public int getNumero() {  
        return numero;  
    }  
    public void setNumero(int numero) {  
        this.numero = numero;  
    }  
    public double getArea() {  
        return area;  
    }  
    public void setArea(double area) {  
        this.area = area;  
    }  
    public String getTipo() {  
        return tipo;  
    }  
    public void setTipo(String tipo) {  
        this.tipo = tipo;  
    }  
}
```



EJEMPLO PRÁCTICO

De momento vamos a partir de la idea de que cada Hotel solo tiene una habitación(sería algo muy raro, normalmente hay más de una), porque todavía no se ha tratado el concepto de arrays que veremos en próximas unidades y será entonces cuando podremos aplicarlo a las relaciones entre clases.

De cada hotel de la cadena se quiere almacenar su nombre, dirección, nombre del gerente o responsable y las características de la habitación. Se declaran estos atributos, el constructor con parámetros y los métodos getters y setters. El atributo habitacion es un objeto que pertenece a la clase Habitacion que ya hemos creado anteriormente.

```
package ud4.hotel.habitacion;

public class Hotel {
    private String nombre;
    private String direccion;
    private String nombreGerente;
    private Habitacion habitacion;

    public Hotel(String nombre, String direccion, String nombreGerente,
        Habitacion habitacion) {
        this.nombre = nombre;
        this.direccion = direccion;
        this.nombreGerente = nombreGerente;
        this.habitacion = habitacion;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public String getNombreGerente() {
        return nombreGerente;
    }

    public void setNombreGerente(String nombreGerente) {
        this.nombreGerente = nombreGerente;
    }

    public Habitacion getHabitacion() {
        return habitacion;
    }
}
```



```
public void setHabitacion(Habitacion habitacion) {  
    this.habitacion = habitacion;  
}  
}
```



EJEMPLO PRÁCTICO

Necesitas una clase que contenga un método main para ejecutar la aplicación. Por tanto, te dispones a crear cada una de las habitaciones que pueden formar parte de los hoteles del cliente.

Cada hotel dispone de una serie de habitaciones que tienen una numeración, un tamaño y son de un tipo. Cuando amplíes la aplicación, en un futuro, puedes incluir otra clase Cliente para contemplar aquellos clientes que se registran en un hotel y en una habitación determinada.

De momento solo creas tres objetos correspondientes a las habitaciones que se encuentran en el ala sur(El cliente nos ha comentado que clasifica las habitaciones de los hoteles según la orientación hacia el norte, sur, etc). A continuación se crean otros tres objetos para tres hoteles diferentes(el cliente gestiona una cadena de hoteles). A cada objeto de tipo Hotel se le pasarán como parámetros dentro del constructor los datos del hotel y de la habitación(que es un objeto).

Recuerda que de momento vamos a partir de la idea de que cada hotel solo tiene una habitación (porque todavía no se ha tratado el uso de arrays).

```
public class MainHotel {  
  
    public static void main(String[] args) {  
        //Crear una instancia de Habitacion  
        Habitacion habitacionSur1=new Habitacion(1, 6, "individual");  
        Habitacion habitacionSur2=new Habitacion(2, 11, "doble");  
        Habitacion habitacionSur3=new Habitacion(3, 20, "suite");  
  
        Hotel hotelNh=new Hotel("NH Collection", "Avenida Diego  
                                Martinez, 8", "Francisco Vidal",habitacionSur1);  
        Hotel hotelAlhambra=new Hotel("Alhambra Palace", "Plaza  
                                Arquitecto García, 1", "Marta Fernández",habitacionSur2);  
        Hotel hotelMarbella=new Hotel("Marbella Club", "Avenida.  
                                Bulevar Principe, s/n", "Rocío Gómez",habitacionSur3);  
  
        //visualizar en pantalla datos almacenados  
        System.out.println("HOTEL: " );  
        System.out.println("-----" );  
        System.out.println("Nombre del hotel: "+ hotelNh.getNombre()  
    );  
  
        System.out.println("Dirección: "+ hotelNh.getDireccion() );  
        System.out.println("Gerente: "+ hotelNh.getNombreGerente());  
    }  
}
```

```

System.out.println("\nDatos Habitación:");
System.out.println("-----" );
System.out.println("Número habitación: "+
                    hotelNh.getHabitacion().getNumero());
System.out.println("Metros cuadrados: "+
                    hotelNh.getHabitacion().getArea());
System.out.println("Tipo: "+ hotelNh.getHabitacion().getTipo()
                    + "\n\n");
System.out.println("HOTEL:" );
System.out.println("-----" );
System.out.println("Nombre del hotel: "+
                    hotelAlhambra.getNombre());
System.out.println("Dirección: "+
                    hotelAlhambra.getDireccion());
System.out.println("Gerente: "+
                    hotelAlhambra.getNombreGerente());
System.out.println("\nDatos Habitación:");
System.out.println("-----" );
System.out.println("Número habitación: "+
                    hotelAlhambra.getHabitacion().getNumero()
                    );
System.out.println("Metros cuadrados: "+
                    hotelAlhambra.getHabitacion().getArea());
System.out.println("Tipo: "+ hotelNh.getHabitacion().getTipo()
                    + "\n\n");
System.out.println("HOTEL:" );
System.out.println("-----" );
System.out.println("Nombre del hotel: "+
                    hotelMarbella.getNombre());
System.out.println("Dirección: "+
                    hotelMarbella.getDireccion());
System.out.println("Gerente: "+
                    hotelMarbella.getNombreGerente());
System.out.println("\nDatos Habitación:");
System.out.println("-----" );
System.out.println("Número habitación: "+
                    hotelMarbella.getHabitacion().getNumero()
                    );
System.out.println("Metros cuadrados: "+
                    hotelMarbella.getHabitacion().getArea());
System.out.println("Tipo: "+
                    hotelMarbella.getHabitacion().getTipo() +
                    "\n\n");
    }
}

```

Salida por pantalla:

```

HOTEL:
-----
Nombre del hotel: NH Collection
Dirección: Avenida Diego Martinez, 8

```

Gerente: Francisco Vidal

Datos Habitación:

Número habitación: 1

Metros cuadrados: 6.0

Tipo: individual

HOTEL:

Nombre del hotel: Alhambra Palace

Dirección: Plaza Arquitecto García, 1

Gerente: Marta Fernández

Datos Habitación:

Número habitación: 2

Metros cuadrados: 11.0

Tipo: individual

HOTEL:

Nombre del hotel: Marbella Club

Dirección: Avenida. Bulevar Principe, s/n

Gerente: Rocío Gómez

Datos Habitación:

Número habitación: 3

Metros cuadrados: 20.0

Tipo: suite

3.4 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.

Características:

- Es una relación fuerte.
- La clase contenedora o todo es responsable de crear y destruir la clase contenida o parte.
- La vida útil de la clase contenida o parte depende de la clase contenedora o todo.

- La clase contenida o parte solo puede estar relacionada con una clase contenedora o todo.
- Se representa en diagrama UML con un rombo sombreado de negro del lado de la clase contenedora o todo.

Por ejemplo, un coche tiene un motor. La relación entre el coche y el motor es de composición, ya que el motor no puede existir sin el coche y el coche es responsable de crear y gestionar el motor.



EJEMPLO PRÁCTICO

Imagina que estás trabajando como desarrollador de software para una empresa que se dedica a ofrecer soluciones de gestión de aparcamientos.

Tu jefe te ha asignado la tarea de diseñar e implementar un sistema para gestionar los aparcamientos de diferentes edificios y espacios públicos. Con este sistema, la empresa podrá gestionar fácilmente las plazas de aparcamiento en diferentes ubicaciones, lo que permitirá a los usuarios encontrar aparcamiento de manera más eficiente y mejorar la experiencia general de los visitantes en esos lugares.

En tu diseño, creas dos clases importantes: PlazaAparcamiento y Parking. La clase PlazaAparcamiento representa cada plaza de aparcamiento individual, y tiene atributos como `numero` (número de identificación de la plaza) y `tamanho` (tamaño de la plaza). Además, implementas métodos para obtener y establecer estos atributos y un constructor con parámetros.

```
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

Para este sistema de gestión, vas a contemplar por primera vez la idea de que un Parking puede tener mas de una PlazaAparcamiento(relación de uno a muchos, hasta ahora en este tipo de relación solo considerabais relaciones de uno a uno).

La clase Parking se encarga de gestionar todas las plazas de aparcamiento de un lugar específico. Tiene un atributo numPlazas para almacenar el número total de plazas disponibles en el aparcamiento y un atributo plazas que es un array de objetos PlazaAparcamiento para almacenar todas las plazas individuales. La clase Parking también tiene métodos para obtener y establecer el número de plazas, así como para obtener y establecer el array de plazas.

En próximas unidades profundizaremos en este tipo de relación entre clases con el uso de arrays y se explicará como crear una clase main que gestione relaciones entre clases de uno a muchos. De momento solo se plantea como crear las clases.

```
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;  
    }  
  
}
```



RECUERDA

La diferencia clave entre **agregación** y **composición** radica en el grado de dependencia y acoplamiento entre clases. La agregación es una relación más flexible, donde la clase contenida o parte puede existir de manera independiente, mientras que la composición es una relación más fuerte y la clase contenida o parte depende completamente de la clase contenedora o todo.



VÍDEO DE INTERÉS

En el siguiente video encontrarás una explicación completa sobre la programación orientada a objetos que trata conceptos clave como son la abstracción, encapsulamiento, herencia y polimorfismo.



4. LIBRERÍAS DE CLASES. CREACIÓN. INCLUSIÓN Y USO DE LA INTERFACE

Por último das un repaso al conjunto de librerías en Java para que sepan del conjunto de clases, interfaces, etc. que ya están implementadas en Java y que pueden usar para facilitar el trabajo y hacerlo más ágil. Recomendas que cuando comiencen a programar tengan a mano la documentación de la versión que estén usando de la API de Java y así poder consultarla en caso de dudas sobre los métodos que contiene una determinada clase.

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.

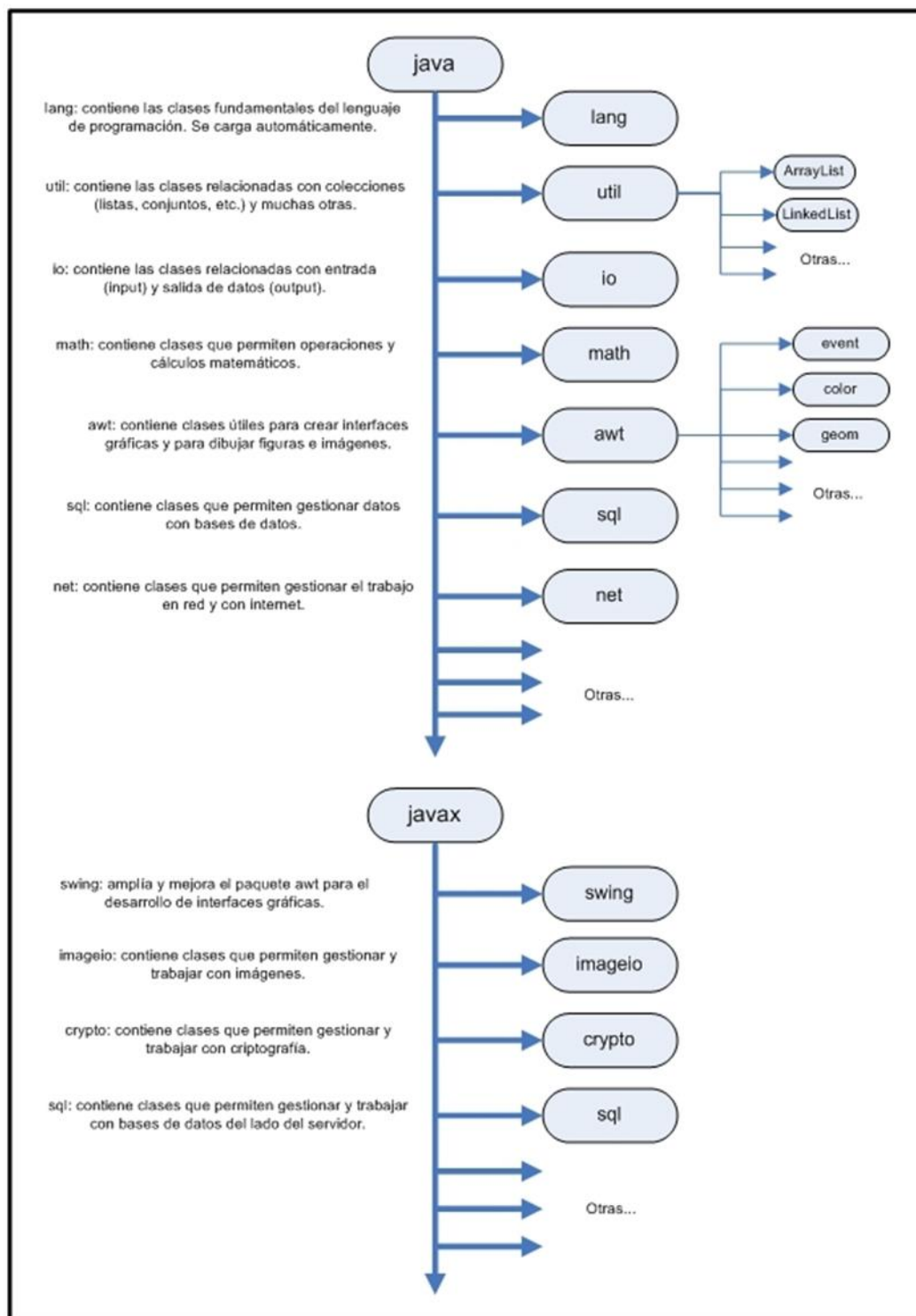
Los nombres de las librerías responden a un **esquema jerárquico** y se basan en la **notación de punto**.

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:

- 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.
- 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.

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`.



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



PARA SABER MÁS

Visita la web de Oracle para conocer toda la jerarquía de paquetes de la 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**.



VÍDEO DE INTERÉS

Para saber más y completar tu aprendizaje, visualiza como se implementan las clases y relaciones en un sistema de gestión de clientes y cuentas bancarias



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.

Una clase se concibe como una descripción estática o plantilla que define nuevos tipos de objetos, compuestos por atributos y métodos. Los atributos describen las características o propiedades de un objeto y los métodos las acciones o funciones que el objeto puede realizar.

Un objeto es una instancia particular de una clase que se debe declarar, ejemplarizar (instanciar) e inicializar.

Por otro lado es necesario que se realice una petición(mensaje) de un objeto a otro para solicitar la ejecución de un método o para obtener el valor de un atributo público. Todo mensaje consta de la identidad del receptor, el nombre del método y una lista de parámetros.

En la programación orientada a objeto existe un concepto muy importante llamado herencia que permite crear nuevas clases basadas en clases existentes. Además existen otros tipos de relaciones como son, la relación de asociación, agregación y composición.

Para poder programar en Java es indispensable conocer la biblioteca estándar de Java, esta está compuesta por cientos de clases como System, String, Scanner, ArrayList, HashMap, etc. Estas nos permiten hacer casi cualquier cosa, con lo que podemos reutilizar código para no tener que volver a crearlo de nuevo. Las clases que queramos utilizar deben incluirse en el desarrollo de nuestro programa.

En general, hemos visto que en esta unidad se ofrece una base sólida para comprender los principios fundamentales de la programación orientada a objetos en Java, lo que permitirá a los desarrolladores crear programas más eficientes y estructurados.