Encapsulamiento, Herencia y Polimorfismo

Encapsulamiento

El encapsulamiento consiste en **controlar el acceso a los datos** que conforman un objeto o instancia de una clase, es decir, debemos indicar qué métodos y atributos son públicos para que puedan ser accedidos por otras entidades e incluso ser modificados. En términos de una clase Java, significa configurar la clase para que solamente los métodos de esa clase con sus variables puedan referir a las variables de instancia.

Setters y Getters

- **Getter: Es un método que, al ser llamado, retorna el valor de una variable.**
- **Setter: Es un método que, al ser llamado, define o configura el valor de una variable.**

Las variables de instancia usualmente se definen como ***private***, mientras que los métodos setter y getter son ***públicos*** (public, ya que forma parte del principio de POO Encapsulamiento), dándole la posibilidad a otras clases de interactuar con la clase en cuestión, sin exponer sus métodos y atributos públicamente.

```java
public class Libro {

    private String titulo;
    private String autor;

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }
}
```

Paquetes en Java (Packages)

Un paquete es un grupo de clases similares, interfaces y sub-paquetes. Pueden ser categorizados como:

- Paquetes incorporados **(Built-in package)**
- Paquetes definidos por el usuario **(User-defined package)**
- Existen varios paquetes incorporados como**java**,**lang**,**awt**,**javax**, etc.

Ventajas:
1. Se utilizan para categorizar las clases y las interfaces para que puedan ser fácilmente mantenidas
2. Proveen protección al acceso
3. Evitan problemas con la colisión de nombres

Modificadores de Acceso

Java ofrece 4 alternativas para modificadores de acceso:

- **public** → Puede ser llamado o accedido desde cualquier clase
- **private** → Puede ser llamado solamente dentro de la misma clase
- **protected** → Puede ser llamado desde clases en el mismo paquete o subclase
- **default (package private)** → Puede ser llamado solamente desde clases en el mismo paquete. No existe una palabra reservada para el acceso por defecto. Simplemente debes omitir el modificador de acceso

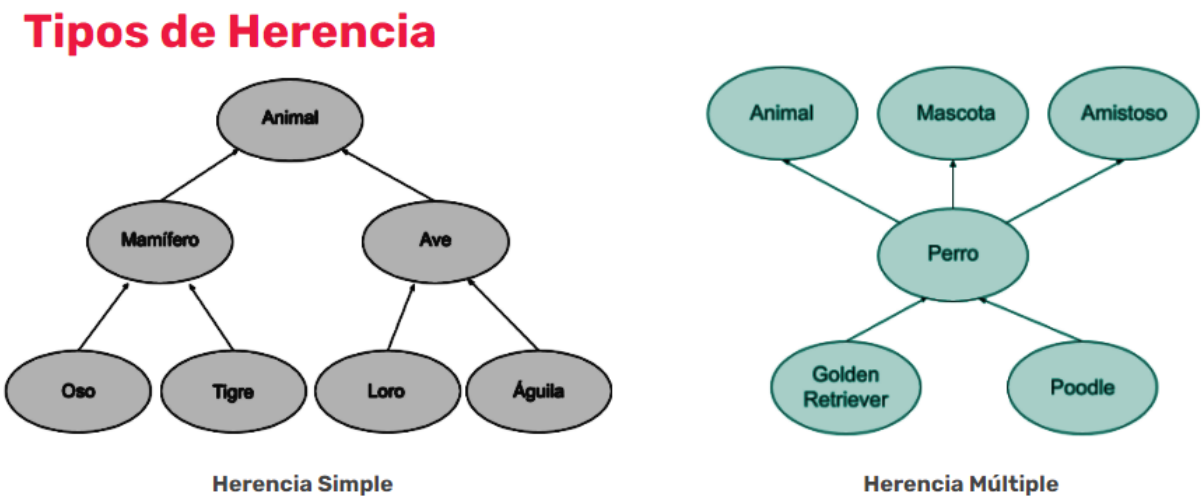
Modificador	Misma clase	Mismo paquete	Subclase	Otro Paquete
private	SI	NO	NO	NO
default	SI	SI	NO	NO
protected	SI	SI	SI	NO
public	SI	SI	SI	SI

Herencia

Cuando creamos una clase en Java, podemos definir que la misma **herede de otra clase existente**. Nos referimos al proceso en el cual la *clase hija* o *subclase* **incluirá automáticamente** cualquier comportamiento (métodos), propiedades (atributos) públicos o protegidos de otra clase existente.

- Cuando una clase hereda o deriva de una clase padre la llamamos clase hija o **subclase**.
- Cuando nos referimos a una clase de la cual clases hijas van a heredar, la llamamos **clase padre** o **superclase**.

- Java soporta **herencia simple**, es decir, que una clase puede heredar solamente de una clase padre de manera directa.
- Puedes heredar **de una clase** tantas veces como quieras, esto permitirá a sus hijos tener acceso a los miembros de la clase padre.
- Java **no soporta herencia múltiple** (donde una clase puede heredar de múltiples clases padre), sin embargo, puede lograrse algo similar a este tipo de herencia, lo cual veremos más adelante en este curso.



La clase Object:

Toda clase en Java deriva (directa o indirectamente) de la clase Object. Es decir, todas las clases **heredan de Object**.

- **boolean** equals(Object object): Este método indica si otro objeto es “igual a” el actual.

“Ejemplo”.equals(“Ejemplo”) → true

- **String** toString(): Este método retorna una representación String del objeto.

“Ejemplo”.toString() → **Ejemplo**

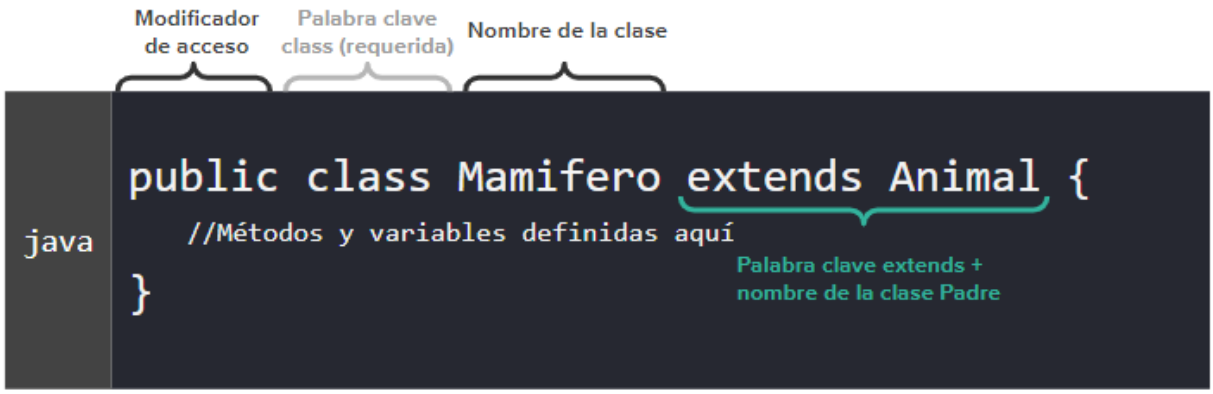
- **Class<?>** getClass(): Retorna la clase en tiempo de ejecución del objeto.

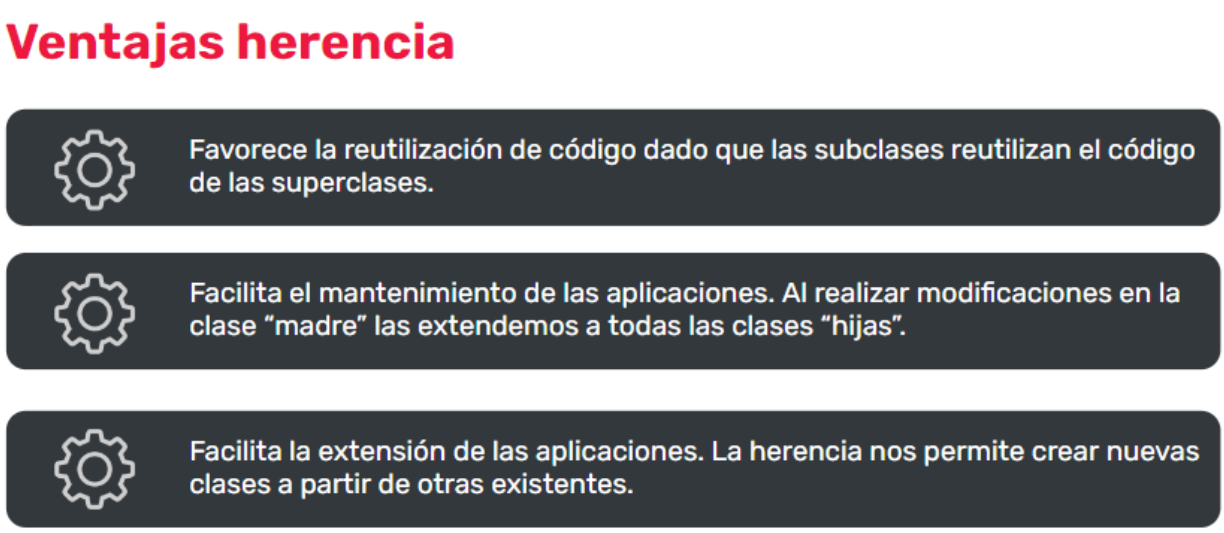
Persona persona = new Persona();

persona.getClass()→class Persona

Herencia:

Podemos heredar o derivar de una clase agregando el nombre de la clase padre en la definición de la misma utilizando la palabra reservada **extends**.





Sobreescritura (Overriding):

Es la forma por la cual una clase que hereda de otra puede **re-definir** los métodos de la clase padre, permitiendo crear nuevos métodos que tengan el mismo nombre de su superclase pero aplicando comportamientos diferentes. Podemos identificar un método

sobreescrito ya que tiene la anotación **@Override**, no es obligatoria en todos los casos pero sí es recomendable utilizarla.

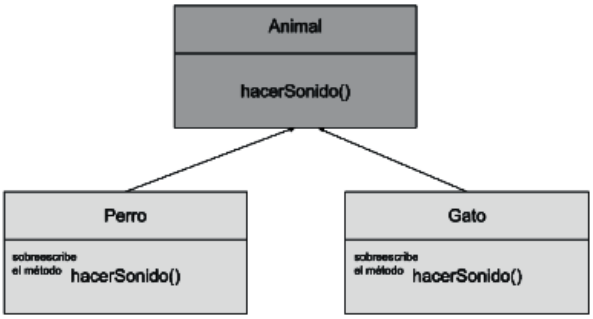
Las condiciones para poder sobreescribir un método son las siguientes:

- La firma del mismo debe ser **IGUAL** al método original de la clase padre, es decir, debe tener el mismo nombre, tipo y cantidad de argumentos como parámetros
- El tipo de retorno debe ser el mismo
- No debe tener un nivel de acceso más restrictivo que el original (por ejemplo si en la clase padre es protected, en la clase hijo no puede ser private)
- No deben ser métodos static ni final (ya que static representa métodos globales y final constantes)

Polimorfismo

"Es uno de los **pilares** de la programación orientada a objetos, refiere a la **propiedad** de un objeto de tomar **diferentes formas**".

Ejemplo de Polimorfismo




```java
public class Animal {

    public void hacerSonido() {
        System.out.println("El animal hace un sonido");
    }
}
```

```java
public class Gato extends Animal {

    @Override
    public void hacerSonido() {
        System.out.println("Miau");
    }
}
```

```java
public class Perro extends Animal {

    @Override
    public void hacerSonido() {
        System.out.println("Guau");
    }
}
```