Herencia

Cuando hablamos de herencia en programación no nos referimos precisamente a que algún familiar lejano nos ha podido dejar una fortuna, ya nos gustaría. En realidad se trata de uno de los pilares fundamentales de la programación orientada a objetos. Es el mecanismo por el cual una clase permite heredar las características (atributos y métodos) de otra clase.

La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados.

Estoy seguro que cuando has leído "reutilizar" se te ha hecho la boca agua ¿verdad? No hay nada mejor en programación que poder usar el mismo código una y otra vez para hacer nuestro desarrollo más rápido y eficiente. El concepto de herencia ofrece mucho juego. Gracias a esto, lograremos un código mucho más limpio, estructurado y con menos líneas de código, lo que lo hace más legible.

En Java tenemos que tener claro cómo llamar a la clase principal de la que heredamos y aquella que hereda de ella, así, clase que se hereda se denomina superclase. La clase que hereda se llama subclase. Por lo tanto, una subclase es una versión especializada de una superclase. Hereda todas las variables y métodos definidos por la superclase y agrega sus propios elementos únicos.

Terminología importante:

- **Superclase:** la clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal).
- **Subclase:** la clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos, además de los campos y métodos de la superclase.
- **Reutilización:** la herencia respalda el concepto de "reutilización", es decir, cuando queremos crear una clase nueva y ya hay una clase que incluye parte del código que queremos, podemos derivar nuestra nueva clase de la clase existente. Al hacer esto, estamos reutilizando los campos/atributos y métodos de la clase existente.

Declara una jerarquía de herencia

En Java, cada clase solo puede derivarse de otra clase. Esa clase se llama superclase, o clase padre. La clase derivada se llama subclase o clase secundaria.

Utiliza la palabra clave extends para identificar la clase que extiende su subclase. Si no declara una superclase, su clase amplía implícitamente la clase Object. El objeto es la raíz de todas las jerarquías de herencia; Es la única clase en Java que no se extiende de otra clase.

Ejemplo de cómo usar la herencia en Java

Para entender el concepto mejor, crearemos una superclase llamada DosDimensiones, que almacena el ancho y la altura de un objeto bidimensional, y una subclase llamada Triángulo que usaremos la palabra clave extends para crear esa subclase.

```
//Clase para objetos de dos dimensiones

class DosDimensiones{

    double base;
    double altura;

    void mostrarDimension(){
        System.out.println("La base y altura es: "+base+" y "+altura);
    }

}

//Una subclase de DosDimensionespara Triangulo

class Triangulo extends DosDimensiones{

    String estilo;
```

```
    double area(){
        return base*altura/2;
    }

    void mostrarEstilo(){
        System.out.println("Triangulo es: "+estilo);
    }
}
```

Clase Principal

```
class Principal{
    public static void main(String[] args) {
        Triangulo t1=new Triangulo();
        t1.base=4.0;
        t1.altura=4.0;
        t1.estilo="Estilo 1";
        System.out.println("Información para T1: ");
        t1.mostrarEstilo();
        t1.mostrarDimension();
        System.out.println("Su área es: "+t1.area());
    }
}
```

En el programa anterior, cuando se crea un objeto de clase Triangulo, una copia de todos los métodos y campos de la superclase adquiere memoria en este objeto. Es por eso que, al usar el objeto de la subclase, también podemos acceder a los miembros de una superclase.

Herencia y modificadores de acceso

Los modificadores de acceso definen qué clases pueden acceder a un atributo o método. esto podría servir por ejemplo para ser usados para proteger la información o mejor dicho definir cómo nuestro programa accede a ella. Es decir, los modificadores de acceso afectan a las entidades y los atributos a los que puede acceder dentro de una jerarquía de herencia.

Aunque así de pronto esto pueda parecer complejo lo mejor, para entenderlo, es resumir sus características en una descripción general rápida de los diferentes modificadores:

- Solo se puede acceder a los atributos o métodos privados (private) dentro de la misma clase.
- Se puede acceder a los atributos y métodos sin un modificador de acceso dentro de la misma clase, y por todas las demás clases dentro del mismo paquete.
- Se puede acceder a los atributos o métodos protegidos (protected) dentro de la misma clase, por todas las clases dentro del mismo paquete y por todas las subclases.
- Todas las clases pueden acceder a los atributos y métodos públicos.

Como puedes ver en esta lista, una subclase puede acceder a todos los atributos y métodos públicos y protegidos de la superclase. Siempre que la subclase y la superclase pertenecen al mismo paquete, la subclase también puede acceder a todos los atributos y métodos privados del paquete de la superclase.

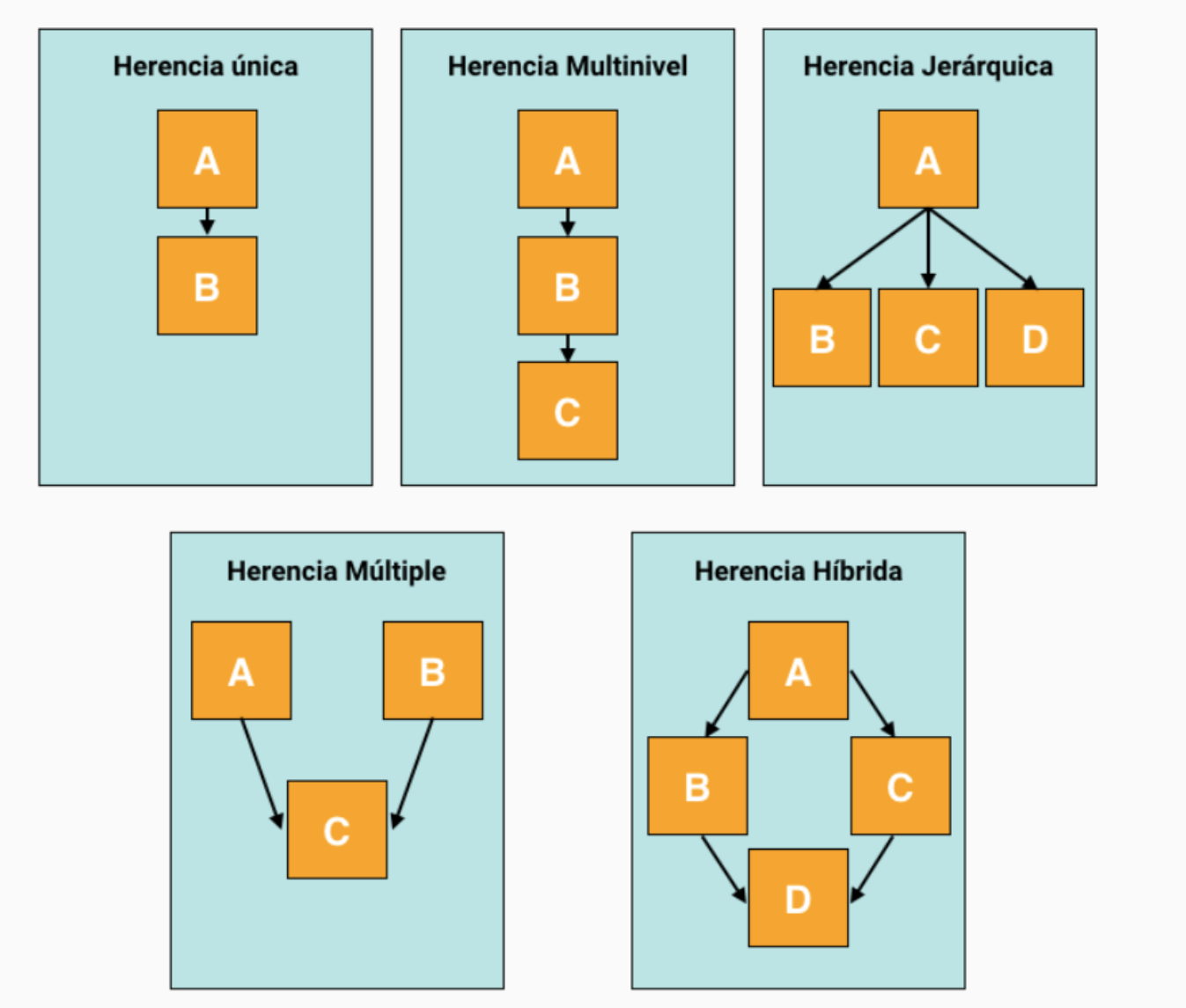
La siguiente tabla muestra el acceso a los miembros permitido por cada modificador:

Especificador	Clase	SubClase	Paquete	Todos
public	sí	sí	sí	sí
private	sí	no	no	no
protected	sí	sí	sí	no
no declarado	sí	no	sí	no

Tipos de herencia en Java

Java, como la mayoría de lenguajes de programación modernos, dispone de diferentes tipos de herencia que podemos usar para hacer todavía más eficiente nuestro programa al añadir características o atributos procedentes de diferentes clases. Lo que sí debemos tener en cuenta es que solo puede existir una superclase, como veremos en los siguientes tipos de herencia:

- **Herencia única:** donde las subclases heredan las características de solo una superclase.
- **Herencia Multinivel:** una clase derivada heredará una clase base y, además, la clase derivada también actuará como la clase base de otra clase.
- **Herencia Jerárquica:** una clase sirve como una superclase (clase base) para más de una subclase.
- **Herencia Múltiple (a través de interfaces):** una clase puede tener más de una superclase y heredar características de todas las clases principales. Pero Java no admite herencia múltiple con clases, así que para lograrlo tenemos que usar Interfaces.
- **Herencia Híbrida (a través de Interfaces):** Es una mezcla de dos o más tipos de herencia anteriores. Como Java no admite herencia múltiple con clases, la herencia híbrida tampoco es posible con clases, pero como en el ejemplo anterior, podemos lograr el mismo resultado a través de Interfaces.



En resumen, la herencia nos abre un mundo de posibilidades y combinaciones que nos ayudan a trabajar en desarrollos complejos con más eficiencia, mostrando código ordenado y estructurado que resultará no solo fácil de leer, sino también de mantener en un futuro dentro de su ciclo de desarrollo.

Bibliografía:

- Extraído de <<¿Qué es la herencia en programación orientada a objetos?>>. 5 de junio de 2019. Consultado el 28 de noviembre del 2022