# HERENCIA

AUTOR: WALTER MADRIGAL CHAVES

NOVIEMBRE: 2020

San Marcos

# Contenido

INTRODUCCIÓN ..... 2
HERENCIA ..... 3
TIPOS DE HERENCIA ..... 3
VENTAJAS DE LA HERENCIA ..... 5
EXTENDS ..... 6
ÁMBITO PROTECTED ..... 6
INTERFACES ..... 8
CLASES ABSTRACT ..... 9
HERENCIA MÚLTIPLE DE INTERFACES ..... 10
CONCLUSIONES Y RECOMENDACIONES ..... 12
REFERENCIAS BIBLIOGRÁFICAS ..... 12

# INTRODUCCIÓN

El proceso para el desarrollo de software día a día está en constante mejora, a menudo salen a la luz nuevos lenguajes de programación, nuevas técnicas de optimización de código y mejores estructuras para administrar los datos. La programación orientada a objetos es un paradigma que, a pesar de tener años de concebida, se considera bastante eficiente y acorde a las necesidades del mercado.

La Herencia es uno de los pilares fundamentales que dan forma a la programación orientada a objetos, su lógica consiste en crear clases o super clases genéricas llamadas padres que servirán para derivar otras, con esto, tener relaciones constantes y generación de jerarquías dentro del mismo código.

En la siguiente lectura, además de abarcar el tema concreto de herencia en clases, se analizarán términos que complementan su entorno, como lo son: los extends, los ámbitos de acción, interfaces, clases abstract entre otros.

## HERENCIA

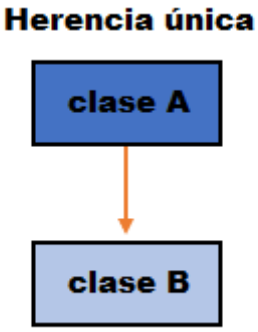
La herencia es una propiedad de POO que permite la creación de objetos a partir de otros ya existentes. Los nuevos objetos obtendrán las características, métodos y atributos de clase principal, esto ayuda a evitar el rediseño, la modificación y verificación de la parte ya implementada.

En el proceso de herencia intervienen o participan dos clases o más clases, en donde la creadora recibe nombres como: clase principal, padre, base, superclase, ancestro, etc. y la clase creada recibe nombre como clase hija, heredada, derivada y subclase.

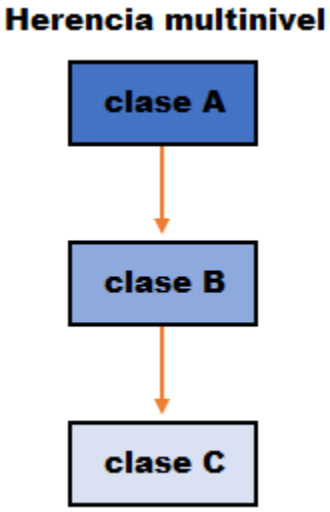
Durante la relación que se da entre la clase padre e hija, puede darse tres posibilidades. Según (Blasco, 2020) La primera es que los atributos y métodos definidos en la superclase puede ser aprovechada por la subclase. La segunda, es que desde la clase padre se puede imponer obligaciones de definición de métodos en otra clase y, por último, se supone que una situación híbrida entre las dos anteriores, se produce cuando la superclase es una clase abstracta.

## TIPOS DE HERENCIA

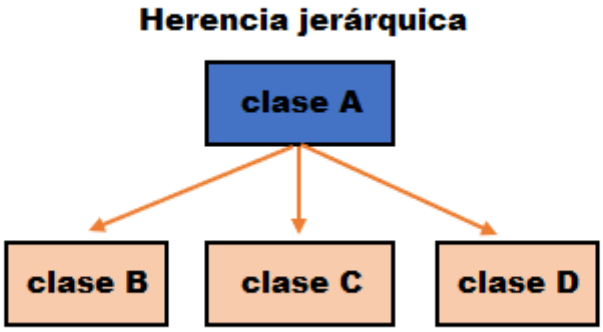
**Herencia única:** las subclases heredan las características de solo una clase padre o superclase.



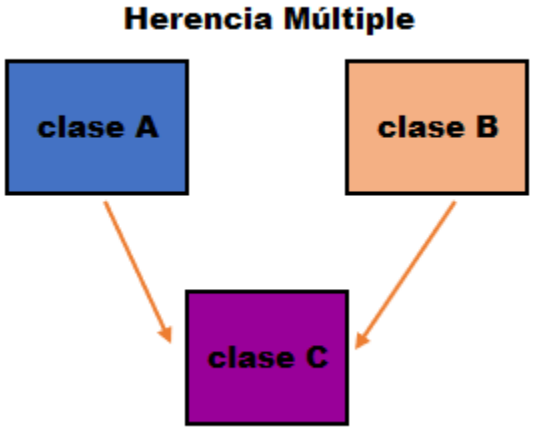

**Herencia Multinivel:** este tipo de herencia se produce cuando una clase base tiene una o más clases hijas y cada una de esas clases hijas puede tener clases derivadas y continuar heredando en cadena. Estas clases constituyen lo que conocido como herencia jerárquica o árbol de herencia.




**Herencia Jerárquica:** es cuando una clase sirve como clase base o padre para más de una subclase.




**Herencia Múltiple (a través de interfaces):** se produce cuando una clase puede tener más de una clase padre y heredar características de todas las clases principales. Java no admite este tipo de herencia por tal motivo para lograrlo es necesario usar Interfaces.




**Herencia Híbrida (a través de Interfaces):** Es una mezcla de dos o más de los tipos de herencia anteriores, al igual que las anteriores se deben aplicar mediante interfaces.

## VENTAJAS DE LA HERENCIA

Las principales ventajas de utilizar la herencia son:

- Posibilidad de reutilización de código (las subclases utilizan el código de

superclases).

- Ahorro en recursos y tiempo.
- Facilita el mantenimiento del código fuente.

## EXTENDS

Extends es una palabra clave que se utiliza para declarar la herencia. En la siguiente imagen se muestra cómo se aplica:

*Imagen 1 Creación de una clase mediante herencia*


Fuente: Elaboración propia

En la imagen anterior se observa la creación de una clase llamada "futbol" a partir de una superclase llamada "Deporte". La primera clase heredará los métodos y atributos de "Deporte", lo anterior es gracias a la utilización de la palabra reservada "extends".

## ÁMBITO PROTECTED

En entregas anteriores se mencionó que los modificadores de acceso son un conjunto de palabras clave que permiten controlar la visibilidad de los objetos (Clases), estado (Propiedades) y funcionalidades (Métodos) de una aplicación desde otras partes de esta. Algunos de los vistos son: tipo privado, público y protected.

En el caso específico del modificador protected es propio del concepto de herencia. Consiste en que los elementos sólo pueden ser accedidos desde su mismo paquete y desde cualquier clase que herede la clase padre en donde se declaró el método o variable como protected.

Lo anterior indica que si una clase quiere acceder a un método o variable marcado como protected deberá heredar de esa clase o moverse a su mismo paquete si es que no se encuentra. Para entender mejor este concepto se presenta el siguiente ejemplo:

*Imagen 2 Creación de clase padre*

```
1   package ejemplo1;
2
3   public class A {
4
5       protected void mostrar(){
6           System.out.println("Java desde Cero");
7       }
8
9   }
```

Fuente: Elaboración propia

En la imagen anterior se crea una clase con el atributo "mostrar" con el modificador protected, lo que significa que únicamente puede ser accedido mediante la herencia. Por lo que se crea en la imagen 3 una clase "B" que hereda de "A" para poder utilizar el atributo "mostrar".

*Imagen 3 Creación de clase heredada*

```
1   package ejemplo2;
2
3   // importar todas las clases en el paquete ejemplo1
4   import ejemplo1.*;
5
6   public class B extends A {
7       public static void main(String[] args) {
8           B obj = new B();
9           obj.mostrar();
10      }
11  }
```

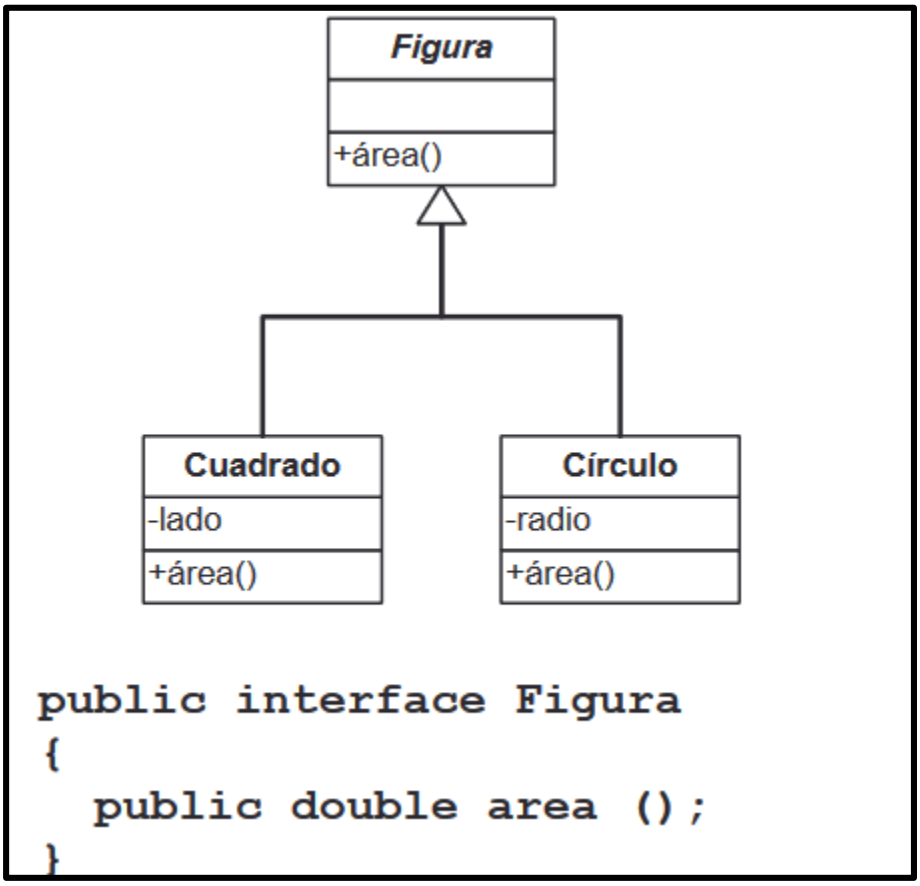
Fuente: Elaboración propia

## INTERFACES

Para (Blasco, 2020), las interfaces son un tipo de superior jerárquico en que se definen "cabeceras" de métodos, sin ninguna implementación de código. Las clases que se definen como subclases de interface mediante implements, deberán definir e implementar, obligatoriamente, todos los métodos cuyas cabeceras han sido definidas en la interface.

Las interfaces en Java sirven de guía para definir un determinado concepto (clase) y lo que está debe realizar, pero sin construir un mecanismo de solución. Consiste en declarar métodos abstractos y constantes que posteriormente puedan ser implementados de diferentes maneras según las necesidades de un programa.

*Imagen 4 Forma general de implementar una interfaz*



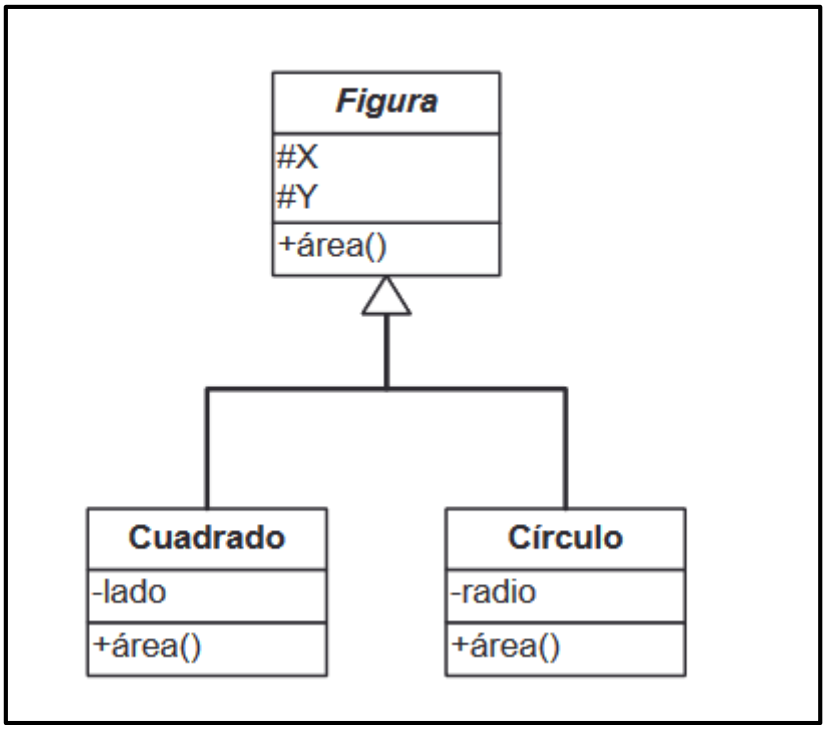

Fuente: https://jarroba.com

En Java, las interfaces se declaran con la palabra reservada interface. En la creación de una interfaz, lo único que puede aparecer son declaraciones de método y definiciones de constantes. Una interfaz no encapsula datos, lo que hace es mostrar los métodos que han de implementar los objetos.

## CLASES ABSTRACT

Son un tipo de clase que no se puede instanciar, solo se utiliza para definir subclases. Se pone en práctica cuando se desea definir una abstracción que englobe objetos de distintos tipos y se quiere hacer uso del polimorfismo.

*Imagen 5 Forma general de una clase abstract*



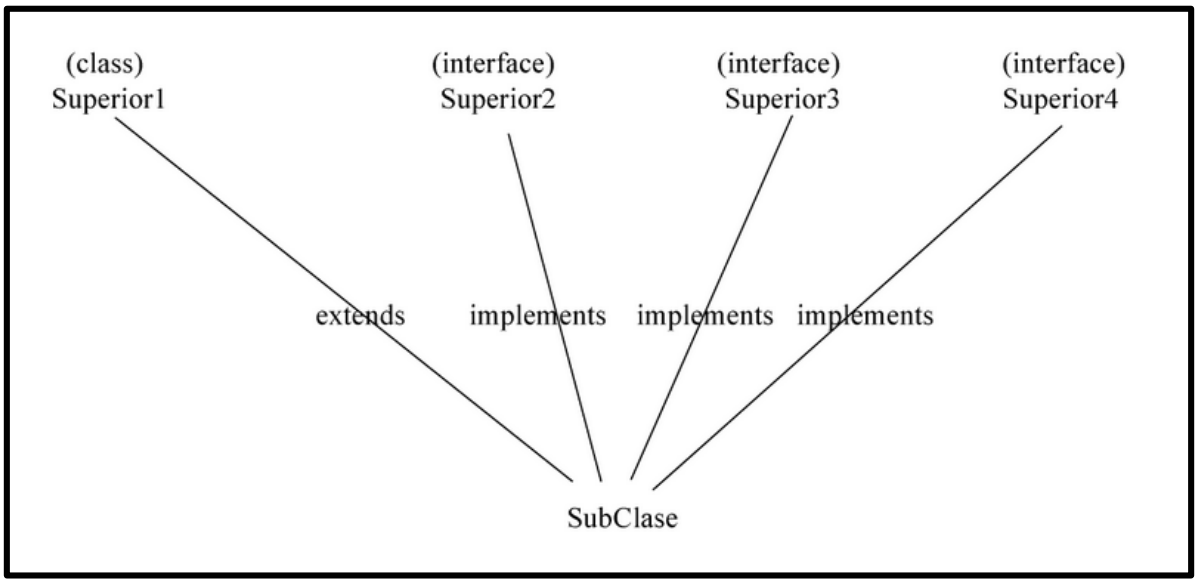

Fuente: https://jarroba.com

La clase "Figura" es de tipo abstracta, porque no tendría sentido calcular su área, pero sí la de un cuadrado o un círculo. Si una subclase de Figura no redefine área (), deberá declararse también como clase abstracta.

## HERENCIA MÚLTIPLE DE INTERFACES

Para (Blasco, 2020), el concepto de herencia múltiple supone que una subclase puede heredar de varias superclases. Una clase puede implementar varios interfaces simultáneamente, pese a que, en Java, una clase sólo puede heredar de otra clase (herencia simple de implementación, múltiple de interfaces).

*Imagen 6 Forma general de herencia múltiple*




Fuente: Programación orienta a objetos (2020)

***Para ahondar más en el tema de Herencia debe realizar la lectura de las páginas 205 a la 235 del libro: Programación orientada a objetos con Java. (2020) de Francisco Blasco.***

## CONCLUSIONES Y RECOMENDACIONES

- La herencia es una muy buena práctica aplicable al desarrollo de software, sus ventajas como: reutilizar código, establecer jerarquías y ahorrar tiempo de implementación, la hacen bastante atractivas.

- Como se analizó en la lectura existen diferentes tipos de herencias clasificadas según su forma de heredar, entonces surge la pregunta ¿Cuál implementar? Se implementa la que más se adapte al problema que se necesita resolver.

- Es importante recordar que el ámbito de modificación protected es propio y se aplica solo en herencia.

- La práctica basada en herencia abre un mundo de posibilidades y combinaciones que ayudan a trabajar en desarrollos complejos con más eficiencia, mostrando código ordenado y estructurado, con lo cual resultará fácil de leer y fácil de mantener.

## REFERENCIAS BIBLIOGRÁFICAS

Abellán, B. (11 de 15 de 2020). *Programacion Orientada a Objetos* . Obtenido de http://picarcodigo.blogspot.com/2012/12/clases-envoltorio.html

Barnes, D., & Kölling, M. (2017). *Programación oerientada a objetos con Java usando Bluej 6a. Ed.* Madrid: Pearson.

Blasco, F. (2020). *Programación orientada a objetos en Java.* Bogotá: Ediciones la U.

Moya, R. (15 de 11 de 2020). *Jarroba*. Obtenido de https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/

San Marcos

**www.usanmarcos.ac.cr**

San José, Costa Rica