Universidad de las Fuerzas Armadas "ESPE"
Ingeniería en Software

Integrantes: Chablay Esteban, Chamorro Myckel, Escobar Isaac, Felix Cristian, González Ariel, Gualotuña Richard.
Materia: Programación Orientada a Objetos.
NRC: 3682.

Introducción a la Herencia en Java

---

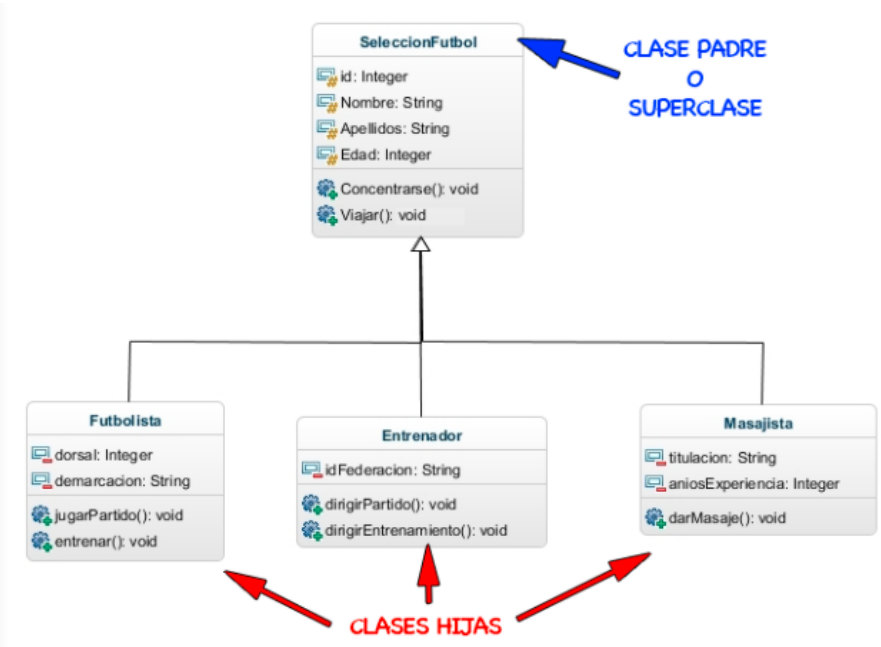
1. Concepto de herencia.

La herencia es uno de los cuatro pilares de la programación orientada a objetos junto con la abstracción, Encapsulación y polimorfismo. Es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente.

El concepto de herencia conduce a una estructura jerárquica de clases, en esta estructura cada clase tiene sólo una clase padre. La clase padre de cualquier clase es conocida como su superclase. La clase hija de una superclase es llamada subclase. La herencia nos permite compartir automáticamente métodos y datos entre las clases, subclases y objetos, pudiendo al mismo tiempo añadir atributos y métodos propios. Existen dos tipos de herencia: Herencia Simple y Herencia Compuesta.

El uso de la herencia al momento de programar nos da ciertas ventajas como la reutilización de código dado que podremos reutilizar los métodos y atributos mencionados en otra clase a la cual esté relacionada así evitando tener que reescribir todos esos métodos en la nueva clase; El mantenimiento de aplicación existente dado que si tenemos una clase con una determinada funcionalidad y tenemos la necesidad de ampliar dicha funcionalidad, no necesitamos modificar la clase existente sino que podemos crear una clase que herede a la primera, adquiriendo toda su funcionalidad y añadiendo la suya propia.

Representación Gráfica de la herencia:



La herencia hace uso de 3 palabras reservadas “extends”, “protected” y “super” definiéndose como:
Extends: esta palabra indica a la clase hija cuál va a ser su clase padre, en nuestro ejemplo podríamos decir que Futbolista extiende de SeleccionFutbol.
Protected: esta palabra nos ayuda a indicar que los atributos y métodos utilizados en la clase padre solo serán visibles desde una de las clases hijas y no desde otra clase.
Super tiene dos formas generales:

- Llama a un constructor de una superclase.
- Se usa para acceder a un miembro de la superclase que ha sido ocultado por un miembro de una subclase.

2. Ejemplo de cómo usar la herencia en Java

El siguiente programa crea una superclase llamada *TwoDimentions*, que almacena el *ancho* y la *altura* de un objeto bidimensional, y una subclase llamada *Triangle*. Observe cómo se usa la palabra clave **extends** para crear una subclase.

Clase TwoDimentions

```java
package ec.edu.espe.inheritance.model;

/**
 *
 * @author Myckel Chamorro EMCL.java ESPE-DCCO
 */
public class TwoDimensions {

    private double base;
    private double height;

    public void showDimension() {
        System.out.println("The base and heigh are: " + getBase() + " and " + getHeight())
    }

    /**
     * @return the base
     */
    public double getBase() {
        return base;
    }

    /**
     * @param base the base to set
     */
    public void setBase(double base) {
        this.base = base;
    }

    /**
     * @return the height
     */
```

Clase Triangle

```java
/**
 *
 * @author Myckel Chamorro EMCL.java ESPE-DCCO
 */
public class Triangle extends TwoDimensions {
    private String style;

    public double area(){
    return getBase()*getHeight()/2;
    }

     public void showStyle(){
     System.out.println("Triangle is: "+getStyle());
    }

    /**
     * @return the style
     */
    public String getStyle() {
        return style;
    }

    /**
     * @param style the style to set
     */
    public void setStyle(String style) {
        this.style = style;
    }

}
```

Main y resultado en pantalla

```
    public static void main(String[] args) {


        Triangle t1=new Triangle();
        Triangle t2=new Triangle();

        t1.setBase(4.0);
        t1.setHeight(4.0);
        t1.setStyle("Style 1");

        t2.setBase(8.0);
        t2.setHeight(12.0);
        t2.setStyle("Style 2");

        System.out.println("Information for T1: ");
        t1.showStyle();
        t1.showDimension();
        System.out.println("His area is: "+t1.area());

        System.out.println();

        System.out.println("Information for T2: ");
        t2.showStyle();
        t2.showDimension();
        System.out.println("His area is: "+t2.area());
    }
}
```

```
Information for T1:
Triangle is: Style 1
The base and heigh are: 4.0 and 4.0
His area is: 8.0

Information for T2:
Triangle is: Style 2
The base and heigh are: 8.0 and 12.0
His area is: 48.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Control de Acceso a Miembros en Herencia

Los modificadores más importantes desde el punto de vista del diseño de clases y objetos, son los que permiten controlar la visibilidad y acceso a los métodos y variables que están dentro de una clase.

Con la variable de instancia de una clase se declara privada () para que se pueda evitar el uso no autorizado. En el siguiente ejemplo muestra la base y la altura se vuelven privados en Dos Dimensiones y con eso para que le trangulo no podrá acceder a ellos
Ejemplo:

```
public class Prueba2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
    class DosDimensiones{
    private double base;
    private double altura;

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

}
```

```
    //Clase para objetos de dos dimensiones
class DosDimensiones{
    private double base;
    private double altura;

    //Métodos de acceso para base y altura
    double getBase(){return base;}
    double getAltura(){return altura;}
    void setBase(double b){base=b;}
    void setAltura (double h){altura=h;}

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


}
```

```
Información para T2:
Triangulo es: Estilo 2
La base y altura es: 8.0 y 12.0
Su área es: 48.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

En la siguiente tabla muestra los niveles de acceso permitido:

	public	protected	(sin modificador)	private
Clase	SI	SI	SI	SI
Subclase en el mismo paquete	SI	SI	SI	NO
No-Subclase en el mismo paquete	SI	SI	SI	NO
Subclase en diferente paquete	SI	SI/NO **(*)**	NO	NO
No-Subclase en diferente paquete (Universo)	SI	NO	NO	NO

Estático:

Puede haber ocasiones en que se puede definir un miembro de clase que se utilizará independientemente de cualquier objeto de esta clase. Solo esto se puede acceder a un miembro de la clase utilizando un objeto de esta clase pero también esto permite crear un miembro que pueda usarse a sí mismo sin referencias en un instancia específica

Este método si tiene restricciones:

- Sólo pueden llamar a otros miembros declarados **static**
- Sólo pueden acceder a datos **static**
- De ninguna manera poder hacer referencia a este o a super

```
    class Test {
    int a; // default access.
    public int b; // public access.
    private int c; // private access.

    // methods to access a c.

    void ECTS (int i) {  int y = 0;
// The value of c.
        c = y;
    }

    int getc () {// The value of c.
        return c;
    }
}
```

4. Constructores y Herencia

Un constructor para una superclase construye la porción de la superclase del objeto, y un constructor para la subclase construye la parte de la subclase.
Esto se debe a que la superclase no tiene conocimiento ni acceso a ningún elemento de la subclase. Por lo tanto cada constructor debe estar separado.

Ejemplo:

Entrada de datos:

```
// Class for two-dimensional objects
//TwoDimensions.java
class TwoDimensions{
    private double base;
    private double height;

    // Access methods for base and height
    double getBase(){return base;}
    double getHeight(){return height;}
    void setBase(double b){base=b;}
    void setHeight (double h){height=h;}

    void showDimension(){
        System.out.println("The base and height is: "+base+" and "+height);
    }
}
```

```java
// A subclass of TwoDimensions for Triangle
//Triagle.java
class Triangle extends TwoDimensions{
    private String style;

    //Constructor
    Triangle(String s, double b, double h){
        setBase(b);
        setHeight(h);
        style=s;
    }

    double area(){
        return getBase()*getHeight()/2;
    }

    void showStyle(){
        System.out.println("Triangle is: "+style);
    }
}
```

```java
/**
 *
 * @author Cristian Félix
 */
public class ThreeSides {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Triangle t1=new Triangle("Style 1",4.0,4.0);
        Triangle t2=new Triangle("Style 2",8.0,12.0);

        System.out.println("Information for T1: ");
        t1.showStyle();
        t1.showDimension();
        System.out.println("Your area is: "+t1.area());

        System.out.println();

        System.out.println("Information for T2: ");
        t2.showStyle();
        t2.showDimension();
        System.out.println("Your area ir: "+t2.area());
    }
}
```

Salida de datos:

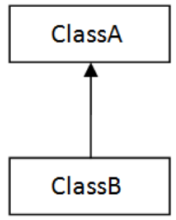
```
Output - ThreeSides (run)
run:
Information for T1:
Triangle is: Style 1
The base and height is: 4.0 and 4.0
Your area is: 8.0

Information for T2:
Triangle is: Style 2
The base and height is: 8.0 and 12.0
Your area ir: 48.0
BUILD SUCCESSFUL (total time: 1 second)
```

Aquí, el constructor del Triangle inicializa los miembros de TwoDimensions que hereda, junto con su propio campo de estilo.

El proceso es un poco más complicado cuando tanto la superclase como la subclase definen constructores ya que debe ejecutarse tanto la superclase como los constructores de la subclase.

5. Tipos de herencia en Java

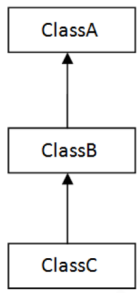
Existen 5 tipos de herencia en la programación, sin embargo, en Java se aplican únicamente 3 de estos conceptos pues no es posible implementar dos de ellos a menos que se haga uso de interfaces. Estos tipos de herencia son:



1) Single

5.1. Herencia única

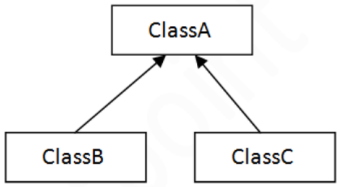
Se da cuando una subclase hereda características únicamente de una clase padre.



2) Multilevel

5.2. Herencia multinivel

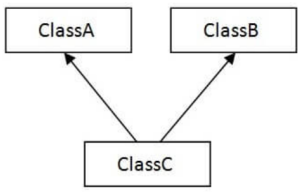
Cuando existen más de dos clases que heredan características entre ellas en forma descendente, este tipo de herencia se conoce como herencia multinivel. Las clases hijas pueden acceder a las características y métodos de cada una de las clases padre de las que hereda su comportamiento. El método reservado super() llama al constructor de la clase padre para poder instanciar un objeto con las características del mismo.



3) Hierarchical

5.3. Herencia jerárquica

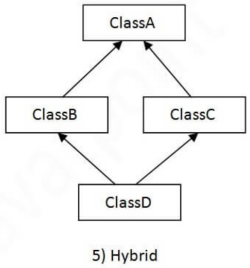
Este tipo de herencia se da cuando una clase hereda información a dos o más clases hijas. En la imagen se puede observar que la clase A es la clase padre tanto para B como para C.



4) Multiple

5.4. Herencia múltiple

Se tiene herencia múltiple cuando una clase hereda características y/o métodos de dos o más clases padre, sin embargo, en Java, este tipo de herencia se da únicamente con el uso de interfaces pues, de la forma tradicional se puede obtener ambigüedad o cualquier tipo de error si las clases padre tienen un mismo método y la clase hija quiere llamar uno de ellos para su implementación.

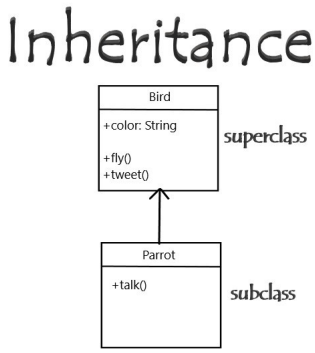


5) Hybrid

5.5. Herencia híbrida

Al igual que sucede en la herencia múltiple, este tipo de herencia se logra únicamente con el uso de interfaces. Se da cuando una clase (clase D en la imagen) hereda características de una clase padre (clase A) por medio de otras clases (clases B y C). Es una combinación de herencia única y herencia simple al mismo tiempo.

6. Datos importantes acerca de la herencia en Java



- **Superclase predeterminada:** excepto la clase **Object**, que no tiene superclase, cada clase tiene una y solo una superclase directa (herencia única). En caso de la ausencia de cualquier otra superclase explícita, cada clase es implícitamente una subclase de clase **Object**.
- **La superclase solo puede ser una**: una superclase puede tener cualquier cantidad de subclases, Pero una subclase solo puede tener una superclase, esto debido a que java no admite herencia múltiple con clases, Aunque con interfaces, la herencia múltiple es compatible con java.
- **Heredar constructores:** una subclase hereda todos los miembros (campos, métodos y clases anidadas) de su superclase los constructores no son miembros, por lo que no son heredados por subclases, pero el constructor de la superclase puede invocarse desde la subclase.
- **Herencia de miembros privados:** una subclase no hereda los miembros privados de su clase principal, Sin embargo, si la superclase tiene métodos públicos o protegidos (como getters y setters) para acceder a sus campos privados, estos también pueden ser utilizados por la subclase.

7. Preguntas

Indique el nombre que se le da a la clase padre.
R: Superclase

La herencia es uno de los pilares de la programación orientada a objetos.

A. Verdadero
B. Falso

¿Cuántas superclases puede admitir java?

A. Ninguna
B. Varias
C. Una

¿Cual es la palabra reservada para la herencia en Java?

A. private
B. exclusive
C. depend
D. extends

¿En qué parte del código se usa la palabra reservada para la herencia?

A. En cada atributo que queremos heredar

B. Junto al nombre de la clase que hereda
C. Importando la clase que se requiere para heredad
D. Ninguna de las anteriores

¿Existen 2 tipos de clases cuales son: ?

A. Públicas y Privadas
B. Solo publicas
C. Solo Privadas
D. Ninguna de las anteriores

¿Completar el siguiente texto?

Estáticos:

Solo se puede acceder a un miembro de la clase utilizando un ……..

A. Clase
B. Objeto
C. Diagrama
D. SubClases

¿Qué constructor es responsable de construir un objeto de la subclase?

A. El de la superclase
B. El de la subclase
C. Ambos
D. Ninguna de las anteriores

Seleccione cuál de las siguientes opciones requiere de interfaces para implementar la herencia en Java:

A. Herencia múltiple
B. Herencia jerárquica
C. Herencia multinivel
D. Herencia única

¿Cuáles son las dos formas generales de la palabra reservada super?

A. Llamar a un constructor de la superclase y acceder a un miembro de la superclase que ha sido ocultado por un miembro de una subclase.
B. Llamar a un constructor de la subclase y acceder a un miembro público de la superclase
C. Llamar a un constructor de la subclase y acceder a un miembro de la subclase que ha sido ocultado por un miembro de la superclase
D. Todas las anteriores

Referencias

Puchol, A. B. (18 de septiembre de 2017). *Abelp.net*. Recuperado el 06 de Julio de 2021, de Abelpt: http://www.abelp.net/apuntesjava/14herencia.html

Walton, A. (26 de abril de 2020). *Javadesdecero*. Recuperado el 06 de Julio de 2021, de Javadesdecero: https://javadesdecero.es/poo/herencia-java-tipos-ejemplos/