

CURSO DE FUNDAMENTOS DE JAVA

HERENCIA EN JAVA



Ing. Ubaldo Acosta

Por el experto: Ing. Ubaldo Acosta



CURSO DE FUNDAMENTOS DE JAVA

www.globalmentoring.com.mx

Hola, te saluda nuevamente Ubaldo Acosta. Espero que estés listo para comenzar con esta lección.

Vamos a estudiar el tema de la herencia en Java.

¿Estás listo? ¡Vamos!

CONCEPTO DE HERENCIA



En esta lección vamos a estudiar el tema de herencia.

Una de las formas en que entendemos la herencia, son las características que se mantienen de generación en generación en una familia. Por ejemplo los abuelos tenían cabello rubio, y un de los hijos hereda esta característica, a su vez el hijo tiene a su vez un hijo que también hereda el cabello rubio.

Por otro lado los mismos abuelos pudieron haber tenido una hija con cabello castaño y lacio, y los nietos pueden heredar también esta misma característica de tener el cabello castaño y lacio.

En la programación orientada a objetos, el concepto de herencia es exactamente igual. Definiremos una jerarquía de clases que nos permitirán heredar características entre clases Padre y clases Hijas.

Veamos a continuación algunas características particulares de Java.

HERENCIA EN JAVA

**REPRESENTAR
COMPORTAMIENTO
EN COMÚN**

**ES UNA
CARACTERÍSTICA
DE LA POO**

**EVITAR
DUPLICACIÓN
DE CÓDIGO**

**DISEÑAR
CLASES MÁS
PARECIDAS AL
MUNDO REAL**

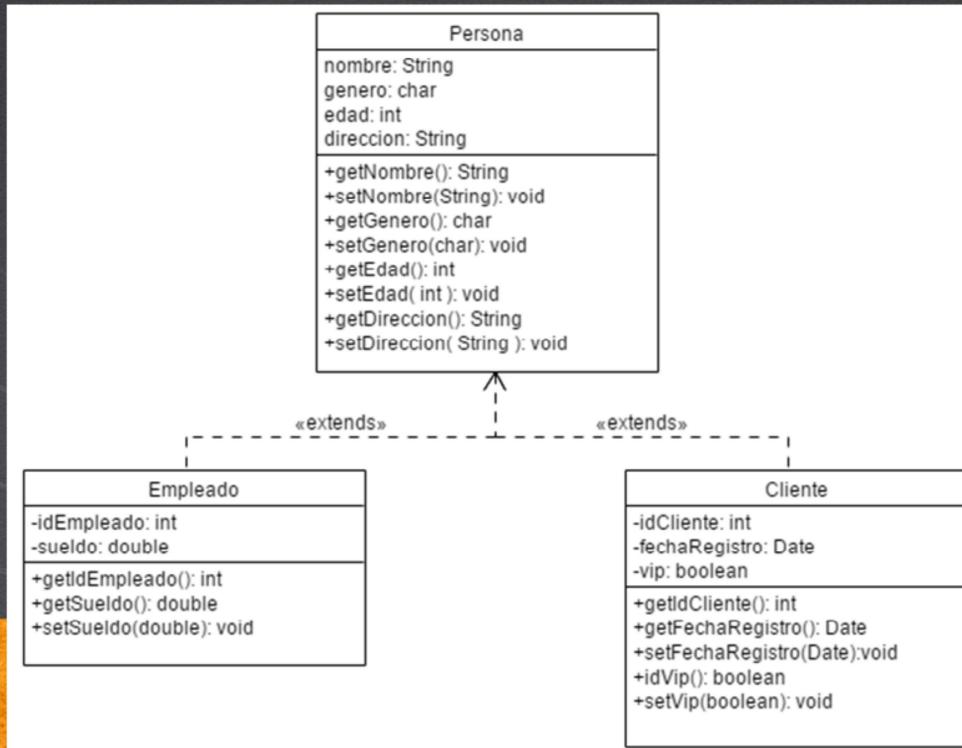
CURSO DE FUNDAMENTOS DE JAVA

www.globalmentoring.com.mx

La herencia es uno de los conceptos más importantes en la programación orientada a objetos (POO), y mencionaremos algunas de las razones más importantes por la cuales aplicaremos este concepto en el diseño de nuestras clases.

- La herencia nos permitirá representar características o comportamiento en común entre clases, permitiendo definir en la clase Padre los atributos o métodos que sean comunes a las clases hijas, las cuales heredarán estos atributos o métodos definidos en la clase Padre.
- Lo anterior permite evitar duplicar el código entre la clase Padre y las clases Hijas, por lo que cumplimos con la reutilización de código que es uno de los principales objetivos de la POO.
- Si aplicamos el concepto de Herencia, podemos modelar de una manera más sencilla sistemas del mundo real. Si ya la definición de objetos es un gran aporte, diseñar una jerarquía de clases, nos facilitará aún más el modelado de clases.

EJEMPLO DE HERENCIA EN JAVA



Vamos a ver un ejemplo de herencia en Java. Podemos observar una clase llamada **Persona**, la cual como cualquier persona tiene muchas características, pero nos vamos a enfocar a unas cuantas. Una persona tiene un nombre, un genero, una edad y una dirección, vamos a capturar estas características en cada uno de los atributos que corresponden en la clase Java, puede tener muchos más, pero nos vamos a enfocar en estos para simplificar el ejemplo.

Por otro lado, si estamos tratando de modelar un sistema de ventas, podríamos tener algunas variantes de una persona en nuestro sistema, por ejemplo la clase de **Empleado** y **Cliente**.

A diferencia de una simple **Persona**, un **Empleado** tiene sueldo, y su actividad puede definir el tipo de empleado que es. Vendedor, Gerente, sin embargo no vamos a ir a más detalle en este ejemplo, vamos a dejar el ejemplo de herencia sólo hasta la relación de **Persona – Empleado**. Un empleado también tiene un identificador, el cual es único para cada empleado, y la clase **Empleado** tiene los métodos para cada atributo.

Por otro lado, podemos tener un cliente que se registra en el sistema de Ventas, este **Cliente** puede tener como atributos un identificador generado de manera muy particular para él, también podemos tener una fecha de alta o de registro, y también podemos indicar si es un cliente especial o VIP, para tales casos, asignamos las variables descritas en la clase **Cliente**, así como los métodos respectivos.

En este ejemplo no estamos indicando cómo llegamos al diseño de estas clases, sin embargo el proceso general es identificar los actores u objetos (entidades que generalmente serán nuestras clases) que interactuarán en nuestro sistema, y a partir de las necesidades de nuestro sistema podemos ir anotando cuales son las características (atributos) y funcionalidad (métodos) que debe tener cada entidad registrada.

Una vez que se han encontrado estas entidades con sus atributos y métodos, vemos si existen características o funcionalidad en común, y allí es precisamente como llegamos a este tipo de diseños. Podemos observar que la clase **Persona** es la clase Padre y que las clases **Empleado** y **Cliente** extienden de la clase **Persona**, es decir, que heredan las características de la clase **Persona**.

Esto permite que no tengamos que repetir el código ya definido en la clase **Persona**, y una vez que creamos un objeto de tipo **Empleado** o **Cliente**, en automático también tendrá las características de la clase **Persona**. En la siguiente lámina veremos un ejemplo más visual de esto.

EJEMPLO DE HERENCIA EN JAVA

Clase Empleado Completa

Clase Persona
 + getNombre()
 + setNombre()
 + getGenero(), etc

Clase Empleado
 +getIdEmpleado()
 +getSueldo(), etc

emp1: Empleado

idEmpleado: int=12345
 sueldo:25000

El objeto creado de tipo Empleado contiene los métodos de la clase Persona que son heredables:

+getNombre()
 +setNombre()
 +getGenero(), etc

Además tiene sus propios métodos:

+getIdEmpleado()
 +getSueldo()
 +setSueldo()

En esta lámina podemos observar del lado izquierdo la definición de dos clases con herencia. Este es una forma en que podemos imaginar visualmente lo que ocurre con la definición de clases una vez que manejamos el concepto de herencia.

Lo que visualizamos es que la clase Persona tiene atributos privados y varios métodos públicos. Los atributos privados no se heredan, al igual que los métodos privados. Es decir, mientras usemos el modificador de acceso private no se heredara el atributo o método marcado con este modificador.

Pero el modificador public si se hereda. Más adelante mencionaremos lo que sucede con los modificadores de tipo package y protected, de momento manejaremos solo estos dos modificadores, private y public para simplificar la explicación del concepto de herencia.

Como observamos la figura muestra la definición “completa” de la clase Empleado, la cual al heredar de la clase Persona, hereda las características que no son privadas, es decir, no hereda los atributos ya que se definieron como private, pero sí hereda los métodos definidos como públicos. Esto permite que cuando creamos un objeto de tipo empleado, como el descrito del lado derecho de la lámina, podemos observar que este objeto de tipo Empleado tiene acceso tanto a los métodos heredados de la clase Persona, así como a los métodos definidos en la misma clase Empleado.

Debemos notar que puede ser heredada ya sea atributos o métodos, lo que lo restringe es el modificador de acceso, ya sea private o public. Recordemos que son 4 modificadores de acceso en total, pero por ahora simplificaremos con sólo estos dos.

En resumen, cuando visualicemos una clase que hereda de otra, podemos imaginarnos a esta clase como el conjunto de sus propios atributos y métodos, así como la suma de los atributos o métodos heredables de la clase o clases Padre, ya que puede recibir herencia de una clase Padre, o una clase “Abuelo”, es decir, una clase Padre de su propia clase Padre.

SINTAXIS HERENCIA EN JAVA

Sintaxis Herencia en Java:

```

class Persona {...} //Definición de la clase Padre o Super Clase
                    ↑
class Empleado extends Persona {...} //Definición de la clase Hija o Subclase
                    ↑
class Gerente extends Empleado {...} //Definición de otra clase Hija o Subclase

```

Nota: TODAS las clases que no especifican de manera explícita un `extends`, entonces heredan de la clase Object, la cual es la clase core de Java.

CURSO DE FUNDAMENTOS DE JAVA

www.globalmentoring.com.mx

Para definir la herencia en Java basta con utilizar la palabra `extends` en la definición de la clase Java e indicar el nombre de la clase Padre.

Ej. `class Empleado extends Persona`

Cabe aclarar que en Java la herencia es simple, esto quiere decir que sólo podemos heredar de una clase a la vez. Puede haber una jerarquía de clases hacia arriba, por ejemplo Clase Abuelo, Padre e Hija, pero una clase no puede heredar por decir de una clase Padre y una clase Madre, es decir, de dos clases al mismo tiempo, únicamente por jerarquía de clases.

Esto SI está permitido, definir una Jerarquía de clases (Clase Padre o SuperClase, Clase Hija o SubClase) :

```

class Persona

class Empleado extends Persona

class Gerente extends Empleado

```

Esto NO está permitido (NO se puede heredar de dos clases a la vez, en Java la herencia es simple):

```
class Gerente extends Empleado, Persona
```

Es importante indicar que TODAS las clases en Java heredan de la clase Object, de manera indirecta o directa. En el caso de la clase Persona, como no está indicando que hereda de alguna clase, es decir, no usa `extends` en la definición de su clase, se entiende que en automático hereda de la clase Object. Pero las clases Empleado o Gerente, debido a que extienden de otras clases, heredarán los métodos de la clase Object de manera indirecta, pero todas las clases en Java de una manera u otra heredan las características de la clase Object. Finalmente, la clase Object es la clase core o base de Java, y pertenece al paquete `java.lang`. Posteriormente hablaremos del tema de paquetes en Java.

USO DE SUPER Y THIS

```

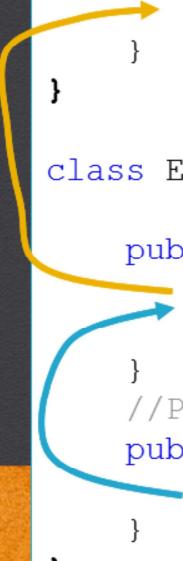
class Persona {//Definición de la clase Padre o Super Clase

    //Constructor 1 argumento
    public Persona(String nombre) {
        this.nombre = nombre;
    }
}

class Empleado extends Persona {//Definición de la clase Hija o Subclase

    public Empleado(String nombre, double sueldo) {
        super(nombre); //Super debe ser la primera linea
        this.sueldo = sueldo;
    }
    //Prueba creación del objeto Empleado
    public static void main(String[] args) {
        Empleado e1 = new Empleado("Juan", 25000);
    }
}

```



Si aplicamos el concepto de herencia la clase Hija o Subclase puede acceder a los atributos, métodos o constructores permitidos de la clase padre solo con utilizar la palabra super.

De esta manera podemos aprovechar código heredado y aprovecharlo por ejemplo para iniciar un objeto de la clase hija. Esto es muy común utilizarlo dentro del constructor para llamar desde el constructor de la clase hija al constructor de la clase Padre.

El constructor de la clase Hija sólo realizará el trabajo correspondiente a la clase Hija, como puede ser la asignación de atributos, pero no realizará el trabajo que ya realiza el constructor de la clase Padre si es posible utilizarlo. Para ello se utiliza la palabra super.

De hecho la palabra this se puede utilizar de manera idéntica pero para llamar a un constructor de la misma clase, y la palabra super para llamar a los constructores de la clase Padre. Ojo, super no solo permite llamar constructores de la clase padre, sino cualquier atributo, método o constructor heredable de la clase Padre.

En el ejemplo podemos observar que utilizamos la palabra super para mandar a llamar el constructor de la clase Padre desde la clase Hija.

toString Y SUPER

```
//Extractos de las clase Persona y Empleado
public class Persona {
    @Override
    public String toString() {
        return "Persona{" + "idPersona=" + idPersona
            + ", nombre=" + nombre
            + ", edad=" + edad + '}';
    }
}

public class Empleado extends Persona {
    @Override
    public String toString() {
        //Primero mandamos a llamar el método toString de Persona
        //para que podamos observar los valores de la clase Padre,
        //y despues imprimimos los valores de la clase hija
        return super.toString() + " Empleado{sueldo=" + sueldo + "}";
    }
}
```

En la lámina podemos observar un ejemplo para el uso del método `toString()`. El método `toString()` según hemos comentado, lo vamos a utilizar para mostrar el estado de un objeto, es decir, los valores de los atributos en cierto momento del tiempo de vida del objeto.

El método `toString` es un método heredado de la clase `Object`. La clase `Object` es la clase Padre de todas las clases Java, ya sea de manera explícita o implícita, según hemos comentado. Por ejemplo, en el caso de la clase `Empleado`, la clase `Object` sería la clase abuela de la clase `Empleado`, debido a que se ha definido que de manera explícita la clase `Persona` como su clase Padre, y debido a que la clase Padre no ha indicado que extiende de alguna clase, entonces se entiende que su clase padre es la clase `Object`. De esta manera la clase `Empleado` también recibe los métodos y atributos que llegase a heredar de la clase `Object`.

Para indicar que un método pertenece a una clase padre o de nivel superiores, como la clase `Object`, entonces debemos agregar la anotación `@Override`, esto quiere decir que estamos sobreescribiendo el comportamiento de un método de la clase padre o clases superiores. Sin embargo este tema de sobreescritura lo estudiaremos en el siguiente curso. Pero para el ejemplo de este curso sólo diremos que es necesario agregar esta anotación para que indiquemos al compilador Java que sabemos que es un método definido en una clase superior, en este caso es un método público de la clase `Object`.

Ahora, este método `toString()` nos sirve según hemos comentado, para convertir el estado de un objeto a una cadena. Pero para el caso que estamos mostrando, la clase `Persona` tiene su método `toString()` y lo ideal es que reutilizaramos este código para completar el método `toString` de la clase `Empleado`, ya que si combinamos el método `toString` de la clase `Persona` con el de la clase `Empleado`, entonces podremos mostrar el estado de todos los atributos de la clase `Empleado` y de su clase padre `Persona`.

¿Cómo hacemos esto? Como observamos en el código, lo que hacemos es utilizar la palabra `super`, y debido a que `toString` es un método público redefinido (sobreescrito) en la clase `Persona`, entonces podemos acceder a este método de la clase padre por medio de la palabra `super`, el operador punto y el nombre del método que deseamos acceder del parent. Esto nos va a regresar una cadena con el estado del objeto `Persona`, y con ello solo basta concatenar el valor que aún no hemos colocado de la clase `Empleado`, por ello concatenamos el valor de la variable `sueldo` y ahora si podemos regresar la cadena completa para que ahora el método `toString` de la clase `Empleado` pueda mostrar tanto su propio estado como el estado de cada atributo de su clase parent que es `Persona`.

Esto lo pondremos en práctica en el ejercicio siguiente.

EJERCICIO CURSO FUNDAMENTOS DE JAVA

- **ABRIR LOS ARCHIVOS DE EJERCICIOS EN PDF.**
- **EJERCICIO:** Ejercicio Herencia en Java

CURSO DE FUNDAMENTOS DE JAVA

www.globalmentoring.com.mx

CURSO ONLINE

FUNDAMENTOS DE JAVA

Por: Ing. Ubaldo Acosta



Experiencia y Conocimiento para tu vida

CURSO DE FUNDAMENTOS DE JAVA

www.globalmentoring.com.mx

En Global Mentoring promovemos la Pasión por la Tecnología Java. Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados, y así te conviertas en un experto programador Java.

Además agregamos nuevos cursos para que continúes con tu preparación como programador Java profesional. A continuación te presentamos nuestro listado de cursos:

- | | |
|--------------------------|-------------------------------------|
| ✓ Fundamentos de Java | ✓ Hibernate Framework |
| ✓ Fundamentos de Java | ✓ Spring Framework |
| ✓ Programación con Java | ✓ JavaServer Faces |
| ✓ Java con JDBC | ✓ Java EE (EJB, JPA y Web Services) |
| ✓ HTML, CSS y JavaScript | ✓ JBoss Administration |
| ✓ Servlets y JSP's | ✓ Android con Java |
| ✓ Struts Framework | ✓ HTML5 y CSS3 |

Datos de Contacto:

Sitio Web: www.globalmentoring.com.mx

Email: informes@globalmentoring.com.mx

