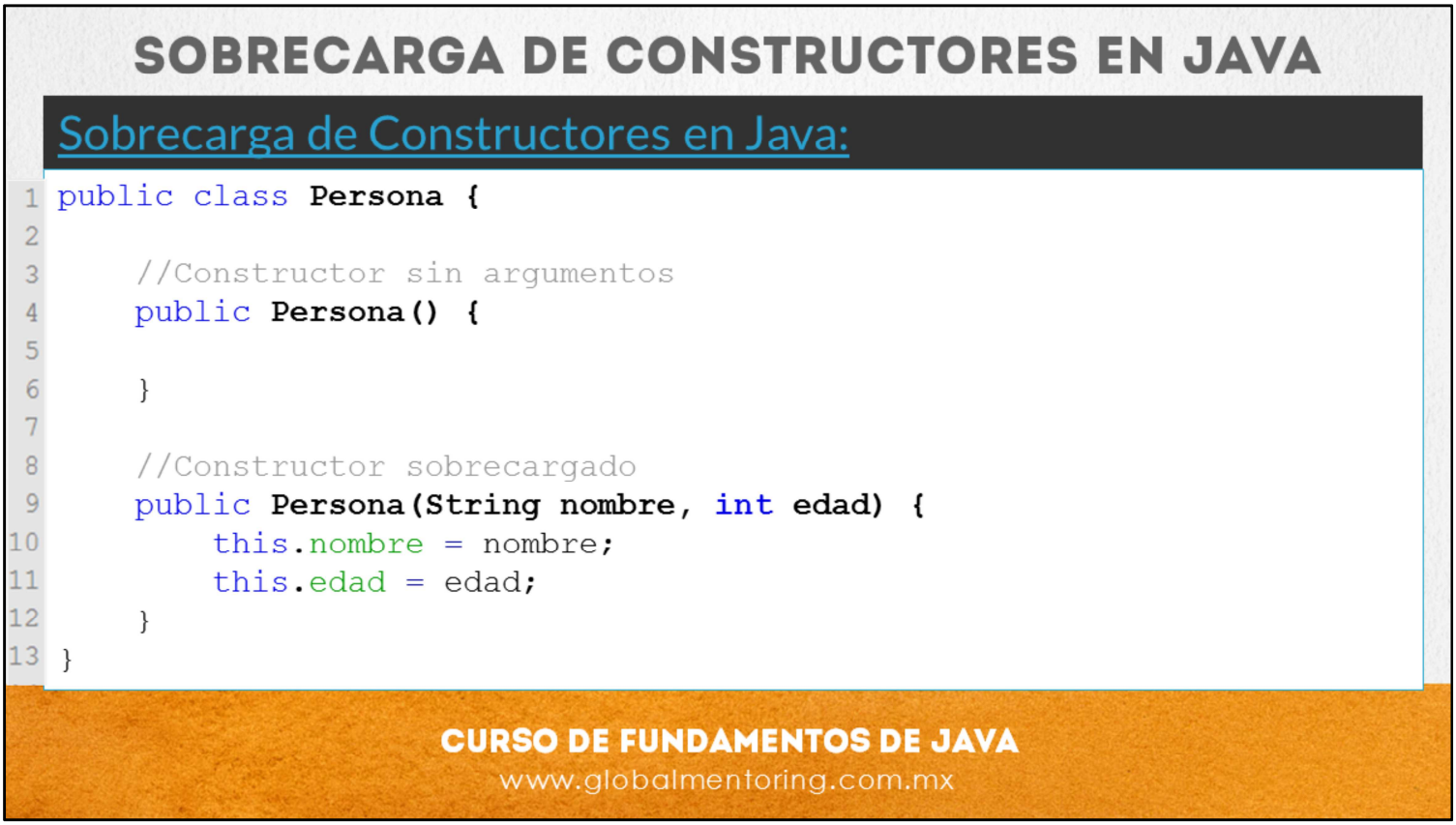

Hola, te saluda Ubaldo Acosta. Bienvenida o bienvenido nuevamente. Espero que estés listo para comenzar con esta lección.

Vamos a estudiar el tema de Sobrecarga de Constructores en Java.

¿Estás listo? Ok, ¡Vamos!

© Todos los Derechos Reservados Global Mentoring




En esta lección para a tratar el tema de sobrecarga de Constructores.

En primer lugar, la sobrecarga de Constructores tiene que ver con ofrecer distintas opciones para crear un objeto de una clase en particular, esto lo lograremos creando distintos Constructores con distintos argumentos, ya sea en número o en tipo de dato. Recordemos que el nombre del constructor es igual al nombre de la clase, respetando que Java es sensible a mayúsculas y minúsculas. Por lo tanto siempre que encontremos un método que sea igual al nombre de la clase y que no regrese ningún tipo de dato, entonces será un Constructor de la clase, pudiendo existir varios en la misma clase.

Podemos decir entonces, que la sobrecarga de un Constructor es ofrecer más opciones para poder construir un objeto de una clase. En el código mostrado en la lámina podemos observar que en primer lugar se ha creado el constructor vacío Persona (línea 4), es decir, sin argumentos. Sin embargo muchas veces queremos ofrecer más opciones de construir un objeto de tipo Persona y que desde el momento de la creación obligue a proporcionar los datos de nombre y edad.

Por lo tanto, agregamos un constructor sobrecargado, con 2 argumentos (línea 9), por lo que tenemos Persona( String, int). Lo importante al momento de sobrecargar un Constructor es observar los tipos de los argumentos, y no sus nombres, por lo que no debe existir otro constructor con los mismos tipos ya definidos en el mismo orden en que se define. Sin embargo, si agregamos un Constructor llamado Persona con los argumentos invertidos, es decir, Persona(int, String) entonces éste se considera otra sobrecarga de los constructores ya definidos. Es decir que el orden y el tipo de los argumentos sí importa.

En resumen, la sobrecarga de Constructores, es definir un Constructor con el mismo nombre de la clase, pero con distintos argumentos, considerando el tipo y orden de los argumentos. Esto se hace con el objetivo de brindar varias opciones para la creación de nuestros objetos según la clase que estemos codificando.

© Todos los Derechos Reservados Global Mentoring

# USO DE this SOBRECARGA DE CONSTRUCTORES

## Uso de this en la Sobrecarga de Constructores en Java:

```
1  public class Persona {
2      //Constructor sin argumentos
3      //Asigna el idPersona
4      private Persona() {
5          this.idPersona = ++contadorPersonas;
6      }
7
8      //Constructor sobrecargado
9      public Persona(String nombre, int edad) {
10         //Se manda a llamar el constructor vacío
11         //para que se asigne el idPersona
12         this();
13         this.nombre = nombre;
14         this.edad = edad;
15     }
16 }
```

En ocasiones podemos tener llamadas entre constructores de la misma clase. Existen varias razones para realizar esta tarea, pero supongamos que el constructor vacío hace la tarea de asignar un valor al atributo idPersona, entonces este código podemos evitar repetirlo en otros constructores únicamente haciendo una llamada al constructor vacío desde cualquier otro constructor sobrecargado.

Para hacer este llamada entre constructores es que utilizamos la palabra this y posteriormente especificamos los argumentos que enviamos según el constructor que deseemos utilizar. En nuestro ejemplo estamos llamando al constructor vacío (línea 12) con el objetivo de reutilizar el código que ya realiza el constructor vacío. De hecho podemos observar como una característica extra que el constructor vacío es privado, es decir que no podemos crear objetos de tipo Persona sin argumentos, sino que forzosamente debemos utilizar algún constructor sobrecargado con argumentos, en este caso debemos proporcionar el nombre y la edad de la persona de manera obligatoria, y de manera interna la funcionalidad de asignar el valor de idPersona lo realiza internamente la clase Persona por medio del constructor vacío, por ello se manda a llamar el constructor vacío por medio de this( ), esto quiere decir que se manda a llamar al constructor sin argumentos. Si tuviéramos un constructor con solo un argumento de tipo String y quisiéramos mandarlo llamar desde otro constructor, la sintaxis sería: this( “Cadena” ); es decir que deberíamos proporcionar un argumento de tipo string para llamar este argumento.

La única condición para mandar a llamar a otro constructor desde un constructor sobrecargado es que el uso de this sea la primera línea del constructor, por ello la línea 12 lo primero que se usa en el constructor es this( );

También es posible utilizar esta característica con clases Hijas, es decir, cuando estamos manejando herencia, veamos cómo se realiza esta tarea.

© Todos los Derechos Reservados Global Mentoring

# USO DE super SOBRECARGA DE CONSTRUCTORES

## Uso de super en la Sobrecarga de Constructores en Java:

```
1  public class Empleado extends Persona {
2
3      private double sueldo;
4
5      public Empleado(String nombre, int edad, double sueldo) {
6          //Super debe ser la primera línea
7          super(nombre, edad);
8          this.sueldo = sueldo;
9      }
10
11 }
```

CURSO DE FUNDAMENTOS DE JAVA
www.globalmentoring.com.mx

Partiendo de la clase Persona que tenemos en la lámina anterior, definiremos la clase Empleado, la cual extiende de la clase Persona. Por lo tanto hereda todas las características que son heredables, es decir, que no son de tipo private.

Esto quiere decir que la clase Empleado tiene acceso al constructor público Persona de dos argumentos (String e int), y por lo tanto podemos apoyarnos de el para inicializar los atributos de la clase Persona. De hecho por la forma en que está construida la clase Persona, es la única forma de inicializar los atributos de dicha clase.

Por ejemplo el atributo idPersona de la clase Persona, no existe forma de accederlo directamente, ni siquiera a través de algún método o un constructor. La única manera de accederlo e inicializarlo es mandando a llamar el constructor vacío, sin embargo este constructor vacío de la clase Persona es privado, por lo tanto no podemos usarlo desde la clase Empleado.

La única forma que tenemos de inicializar el objeto Persona es a través de la llamada al constructor público Persona(String nombre, int edad). Ahora, ¿la pregunta es cómo podemos acceder a este constructor?

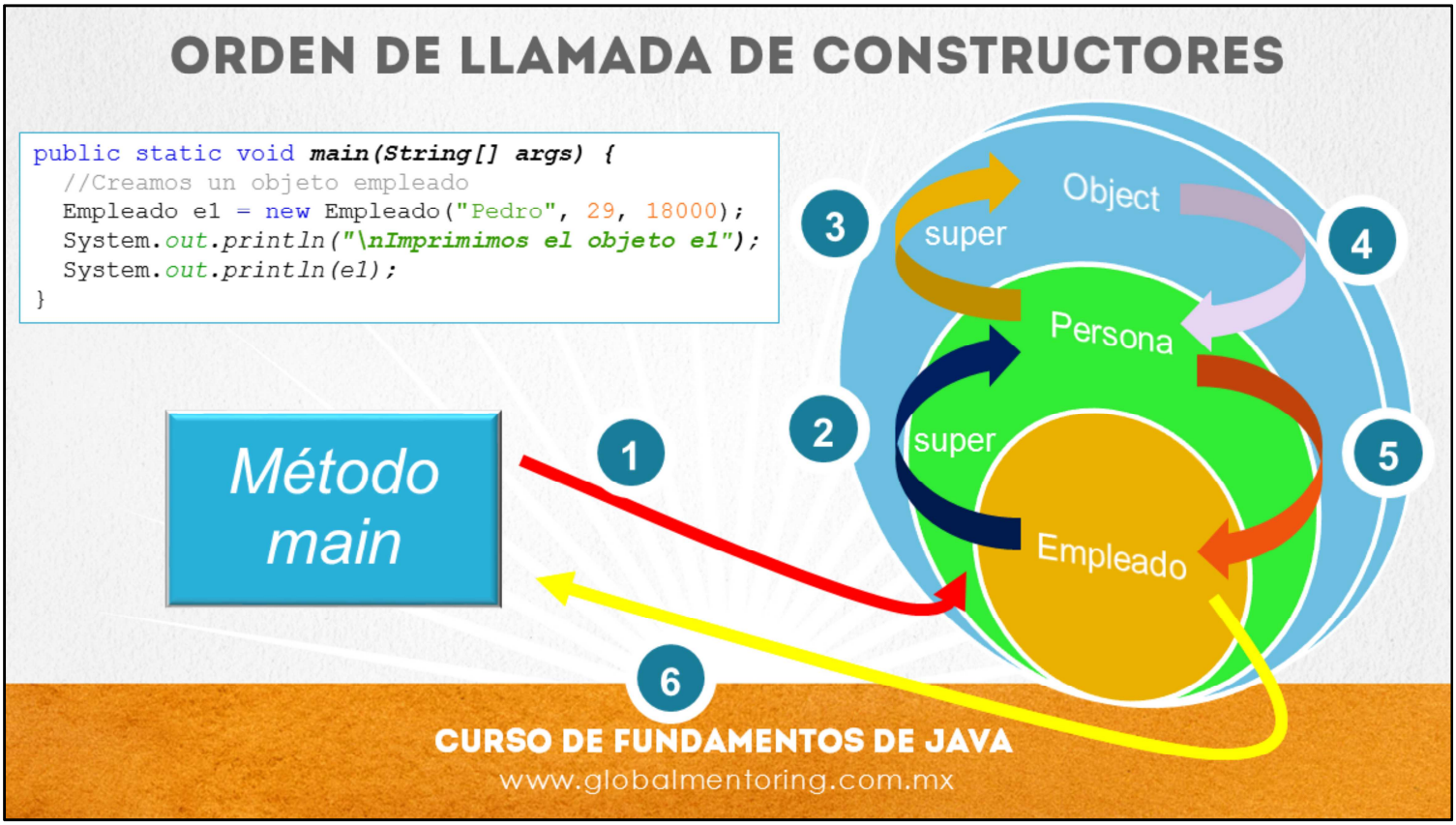
La respuesta es utilizando la palabra super. Esta palabra nos permite acceder a los constructores, métodos o atributos de la clase padre siempre y cuando sean accesibles para nuestra clase. Cuando veamos a detalle el tema de modificadores de acceso veremos que existen otros modificadores además de public que nos permitirán acceder a Constructores, métodos o atributos de la clase padre.

Sin embargo para el ejemplo que estamos mostrando, la palabra super la estamos usando para inicializar los atributos de la clase Persona, de la cual extiende Empleado, si no lo hiciéramos de esta forma, estaríamos dejando sin asignar los valores respectivos a los atributos de la clase Persona, ya que todos sus atributos son privados, y se podrían asignar valores si existieran métodos de tipo mutator (set) para los atributos respectivos, pero si no existe este método mutator entonces se quedarían sin inicializar.

Así que este es un ejemplo de cómo podemos empezar a configurar nuestras clases para que al momento de crear nuestros objetos empecemos poner restricciones sobre cómo se deben utilizar e inicializar sus atributos.

Finalmente, podemos observar que cada clase realizar su labor, es decir, la clase Persona realiza las tareas respectivas al momento de crear un objeto de tipo Persona, o la asignación de las variables para las clases hijas, y la clases hijas, en este caso la clase Empleado realiza la inicialización de los atributos que su clase define, pero no duplica código o realiza tareas que no debe realizar, cualquier tarea la delega a la clase Padre y/o los constructores correspondientes. De esta manera evitamos la duplicación de código y comenzamos a crear diseños más elaborados y funcionales en la creación y codificación de nuestras clases.

© Todos los Derechos Reservados Global Mentoring




Con lo que hemos visto hasta ahora, podemos darnos cuenta no solamente de cómo se configuran nuestras clases Java, sino también el orden en que se crean los objetos Java.

En primer lugar se manda a llamar el constructor del objeto que estamos creando. Al comenzar a ejecutar este constructor, se hace una llamada implícita a super( ), es decir, a la clase Padre.

Todas las clases en Java heredan de la clase Object según hemos comentado, por lo tanto todos los constructores en un momento u otro hacen una llamada a super( ), y eventualmente termina llamando al constructor vacío de la clase Object. Este constructor es el encargado de reservar memoria entre varias tareas más, sin embargo para esta lección basta con saber que es el constructor que se manda a llamar en todos los casos que se crea un objeto.

Una vez que termina de ejecutarse el constructor de la clase padre, se regresa el control al constructor que había iniciado la llamada, o que es a su vez el constructor padre, y así hasta que la cadena de constructores se concluya. En pocas palabras los constructores hijos son los que inician la llamada, pero estos a su vez en su primera línea de código, de manera directa o indirecta llaman al constructor de la clase padre. Si no es especifica nada, entonces la llamada es al constructor vacío de la clase padre, es decir, super ( );

Una vez que el constructor vacío de la clase Padre termina de ejecutarse, se regresa el control al constructor de la clase que inicio la llamada y continua con la ejecución de su código. Si ya se concluyó con la llamada de constructores entonces se regresa el controla al método que hizo la llamada a la construcción del objeto respectivo.

Por ejemplo, en el código de nuestro ejemplo, el método main crea un objeto de la clase Empleado, pero esta a su vez tiene como clase padre a la clase Persona, y la clase persona al ya no extender de ninguna otra clase de manera explícita, entonces extiende de manera implícita de la clase Object. Todas las clases en Java si no especifican un extends en la definición de su clase, entonces podemos decir que su clase padre es la clase Object según hemos comentado.

Por lo tanto, la primer llamada que se realiza es al constructor de la clase Empleado, con 3 argumentos, es decir, Empleado(String, int, int). Este código vamos a suponer que es el mismo de las láminas anteriores. Entonces, antes de hacer la asignación del atributo sueldo, lo primero que hace es hacer una llamada al constructor de su clase Padre, es decir, al constructor de la clase Persona. Debido a que sí se está haciendo una llamada explícita por medio de super, entonces se manda a llamar al constructor de la clase Persona con dos argumentos.

Este constructor de la clase Persona(String, int) supondremos que es el mismo código de las láminas anteriores, por lo tanto, se hace una llamada al constructor sin argumentos con el uso de this( ). Y es constructor sin argumentos debido a que no hace una llamada explícita a super, entonces de manera implícita se llama super( ), es decir al constructor de la clase Object. Una vez que termina este constructor de la clase object de ejecutar su código, regresa el control al constructor vacío de la clase Persona( ) y continua una línea después de donde se hizo la llamada implícita a super( ). Y una vez que terminan de ejecutarse los códigos de los constructores de la clase Persona, regresa el control al constructor de la clase Empleado, para finalmente concluir con la creación del objeto Empleado, el cual ya ha inicializado todos los atributos tanto de la clase Empleado y de la clase Persona, según el proceso descrito anteriormente.

Esto es importante tenerlo en cuenta ya que cuando estamos programando nuestros constructores y considerando la inicialización de variables es importante saber el orden en el que nuestros constructores serán ejecutados, así como la creación de las clases, sobre todo cuando estamos aplicando el concepto de herencia.

© Todos los Derechos Reservados Global Mentoring

# EJERCICIOS CURSO FUNDAMENTOS DE JAVA

- **ABRIR LOS ARCHIVOS DE EJERCICIOS EN PDF.**
- **EJERCICIO**: Ejercicio Sobrecarga de Constructores en Java.

**CURSO DE FUNDAMENTOS DE JAVA**
www.globalmentoring.com.mx

© Todos los Derechos Reservados Global Mentoring


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:

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

**Datos de Contacto:**

Sitio Web: www.globalmentoring.com.mx
Email: informes@globalmentoring.com.mx

© Todos los Derechos Reservados Global Mentoring