SECCIÓN 21

SOBRECARGA DE CONSTRUCTORES

//Sobrecarga de Constructores en Java

public class Persona { // clase

//Construtores sin argumentos

Public Persona () {

}

//Constructor sobrecargado

public Persona ( String nombre int edad) {

this.nombre = nombre; // inicializador de variables clases

this.edad = edad; //inicializador de variables clases

}

}

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). 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.

USO DE super SOBRECARGA DE CONTRUCYORES

//Uso de super en la sobrecarga de Constructores en Java

public class Empleado extends Persona { //clases

private double sueldo; // variable clase

public Empleado ( String nombre int edad double sueldo) { //Contructor

//Susper debe ser la primera línea

super(nombre, edad);

this.suemdo =sueldo;

}

}

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.

ORDEN DE LA LLAMADA DE CONSTRUCTORES

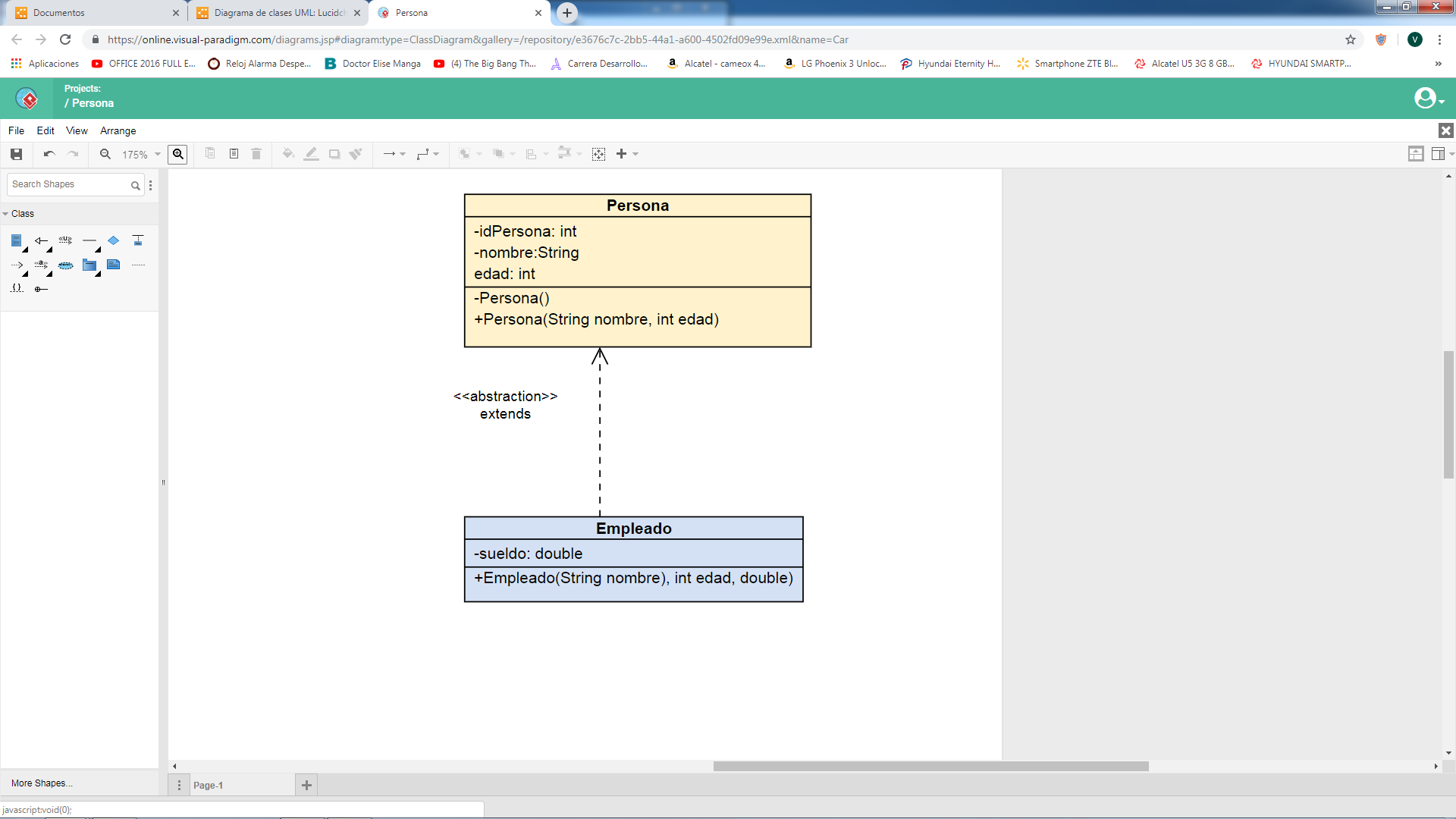


En primer lugar se manda a llamar el constructor del objeto que estamos creando. Al comenzar a ejecutar este constructor, se hace una llamada implícita a super( ), es decir, a la clase Padre.

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.

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 ( );

DIAGRAMA UML



//Código completo de la clase Persona

public class Persona {

private int idPersona ;

private String nombre;

private int edad;

private static int contadorPersonas;

private Persona (){

this.idPersona = ++contadorPersonas;

}

//Sobre caarga de Constructor

public Persona(String nombre, int edad){

this();

this.nombre = nombre;

this.edad = edad;

}

public int getIdPersona() {

return idPersona;

}

public void setIdPersona(int idPersona) {

this.idPersona = idPersona;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public int getEdad() {

return edad;

}

public void setEdad(int edad) {

this.edad = edad;

}

@Override

public String toString() {

return "Persona{" +

"idPersona=" + idPersona +

", nombre='" + nombre + '\'' +

", edad=" + edad +

'}';

}

}

//Codigo de la clase de Empleado

public class Empleado extends Persona {

private double sueldo;

public Empleado (String nombre, int edad , double sueldo){

super(nombre,edad);

this.sueldo = sueldo;

}

public double getSueldo() {

return sueldo;

}

public void setSueldo(double sueldo) {

this.sueldo = sueldo;

}

@Override

public String toString() {

return super.toString() + " Empleado{" +

"sueldo=" + sueldo +

'}';

}

}

//Codigo de la clase de SobreCargaConstructor

public class SobreCargaConstructores {

public static void main(String[] args) {

Persona persona1 = new Persona("Juan", 23);

System.out.println("persona1 = " + persona1);

Empleado empledo1 = new Empleado("Luis",35, 18888);

System.out.println("empledo1 = " + empledo1);

}

}