	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	108/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 08: Herencia (2da parte)




Elaborado por:

M.C. M. Angélica Nakayama C.
Ing. Jorge A. Solano Gálvez

Autorizado por:

M.C. Alejandro Velázquez Mena

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	109/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 08: Herencia (2da parte)

Objetivo:

Implementar programas que permitan crear objetos con diferentes valores iniciales a lo largo de una jerarquía de clases.

Actividades:

- Crear constructores sobrecargados.
- Implementar constructores en clases derivadas que utilicen los constructores definidos en la clase base.


Introducción

Un **constructor** es un método que tiene como propósito **inicializar** los atributos de un nuevo objeto. **Se ejecuta automáticamente cuando se crea un objeto o instancia de la clase.**

Los constructores son métodos especiales que sólo existen cuando se crea un objeto, por lo tanto, dentro de los métodos de la clase no se pueden invocar. Dependiendo del número y tipos de los argumentos proporcionados se ejecutará al constructor correspondiente.

Si no se ha escrito un constructor en la clase, el compilador proporciona un **constructor por defecto**, el cual no tiene parámetros e inicializa los atributos a su valor por defecto.

NOTA: En esta guía se tomará como caso de estudio el lenguaje de programación JAVA, sin embargo, queda a criterio del profesor el uso de éste u otro lenguaje orientado a objetos.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	110/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Constructores

Un método constructor permite reservar en memoria los atributos y métodos definidos en la clase. Las reglas para crear métodos constructores son:

- El constructor debe tener el mismo nombre que la clase.
- Puede tener cero o más parámetros.
- No devuelve ningún valor (ni si quiera void).
- Toda clase debe tener al menos un constructor.

Cuando se define un objeto se pasan los valores de los parámetros al constructor utilizando una sintaxis similar a una llamada normal a un método.

Ejemplo:


Para la clase Punto se declara un constructor que reciba los valores de las coordenadas (x, y) y dichos valores se le asignan a los atributos miembro correspondientes.

```
public class Punto {
    int x,y;

    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void imprimePunto() {
        System.out.println("Punto [x=" + x + ", y=" + y + "]");
    }
}
```

Cuando se definen constructores de manera explícita, en el ejemplo se crea el constructor que recibe dos parámetros, Java deja de agregar el **constructor por defecto** (el constructor sin parámetros) por lo cual si se intenta invocar al constructor por defecto, el compilador marcará error:

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	111/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
public class PruebaPunto {

    public static void main(String[] args) {
        Punto p = new Punto();
        p.x=5;
        p.y=8;
        p.imprimePunto();

        Punto x = new Punto();
        x.x=7;
        x.y=2;
        x.imprimePunto();
    }
}
```

En este caso se pueden hacer dos cosas:


- Cambiar la creación de las instancias para que ahora se envíen los valores (x, y) desde el momento de su creación.

```
public class PruebaPunto {

    public static void main(String[] args) {
        Punto p = new Punto(5, 8);
        p.imprimePunto();

        Punto x = new Punto(7, 2);
        x.imprimePunto();
    }
}
```

- Agregar un constructor por defecto (sin parámetros) el cual puede inicializar los valores a un valor por defecto (o al valor que se desee) o bien puede estar sin implementación (código).

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	112/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

public class Punto {

    int x,y;

    public Punto(){

    }

    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void imprimePunto() {
        System.out.println("Punto [x=" + x + ", y=" + y + "]");
    }

}

```

En este último caso, dado que se cuenta con 2 constructores distintos (diferenciados por el número y tipo de parámetros) se tendrían disponibles las dos formas de crear instancias, ya sea con los valores por defecto o pasándolos como parámetros, es decir, constructores sobrecargados.

```


public class PruebaPunto {

    public static void main(String[] args) {
        Punto p = new Punto(5, 8);
        p.imprimePunto();

        Punto x = new Punto();
        x.x=7;
        x.y=2;
        x.imprimePunto();
    }

}

```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	113/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Constructores en la herencia


Como ya se mencionó, un constructor es un método que tiene el mismo nombre que la clase, que no define un valor de retorno y cuyo propósito es inicializar los atributos de un nuevo objeto. Se ejecuta automáticamente cuando se crea un objeto o instancia de la clase.

Cuando se crea un objeto de una clase derivada, implícitamente se crea un objeto de la clase base, el objeto de la súper clase se inicializa con el constructor por defecto aunque se puede invocar a cualquier constructor.

Por tanto, cuando se crea un objeto de una clase base, sin importar el constructor utilizado, se produce una llamada implícita al constructor sin argumentos de la clase base. Sin embargo, si se quiere utilizar constructores sobrecargados es necesario invocarlos explícitamente.

Como se mencionó anteriormente, los constructores son métodos especiales que sólo existen cuando se crea un objeto y no pueden ser invocados dentro de los métodos de la clase. Sin embargo, durante la creación de los objetos, todos los constructores existen, de tal manera que entre constructores sí se pueden invocar (ya sea constructores de la misma clase, con la palabra reservada **this**, o de la clase base, con la palabra reservada **super**).

Así mismo, dentro de un constructor se puede acceder a los elementos (atributos o métodos) de la clase derivada a través de la palabra reservada **this**, o acceder a los elementos de la clase base a través de la palabra reservada **super**.


	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	114/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo

```
public class Empleado {
    private String nombre;
    private int numEmpleado, sueldo;

    public Empleado (String nombre, int sueldo, int numEmpleado) {
        this.nombre = nombre;
        this.sueldo = sueldo;
        this.numEmpleado = numEmpleado;
    }

    public void setNombre(String nombre){
        this.nombre = nombre;
    }
    public String getNombre(){
        return this.nombre;
    }
    public void setNumEmpleado(int numEmpleado){
        this.numEmpleado = numEmpleado;
    }
    public int getNumEmpleado(){
        return this.numEmpleado;
    }
    public void setSueldo(int sueldo){
        if (sueldo >= 0) {
            this.sueldo = sueldo;
        }
    }
    public int getSueldo(){
        return this.sueldo;
    }
    public void aumentarSueldo(int porcentaje) {
        sueldo += (int)(sueldo * porcentaje / 100);
    }
    @Override
    public String toString (){
        return "Nombre: " + this.nombre +
            "\nNúmero: " + this.numEmpleado +
            "\nSueldo: " + this.sueldo;
    }
}
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	115/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

public class Gerente extends Empleado {
    private int presupuesto;
    public Gerente (String nombre, int sueldo,
                    int numEmpleado, int presupuesto) {
        super(nombre, sueldo, numEmpleado);
        this.presupuesto = presupuesto;
    }
    public void setPresupuesto(int presupuesto){
        this.presupuesto = presupuesto;
    }
    public int getPresupuesto(){
        return this.presupuesto;
    }
    void asignarPresupuesto(int p) {
        presupuesto = p;
    }
    @Override
    public String toString(){
        return super.toString() +
            "\nPresupuesto: " + this.presupuesto;
    }
}

```

Como se puede observar en el ejemplo anterior, el constructor de la clase *Gerente* invoca directamente al constructor de la clase *Empleado* mediante la palabra reservada *super*, pasando como argumentos nombre, sueldo y numEmpleado.


La llamada a otros constructores debe ser **la primera sentencia** dentro del constructor, por lo tanto, solo es posible hacer una llamada a otros constructores dentro del método constructor, es decir, o se utiliza *super* (para acceder a un constructor de la clase base) o se utiliza *this* (para acceder a un constructor de la subclase).

```

public class PruebaEmpleado {
    public static void main (String [] args){
        Gerente gerente = new Gerente("Luis Aguilar", 16000, 8524, 50000);
        System.out.println(gerente);
    }
}

```

En el ejemplo de la clase *Gerente*, debido a que el método *toString* de la clase *Empleado* muestra los atributos deseados, se invoca explícitamente y se concatena el atributo presupuesto. Por eso, el objeto *gerente* creado en la clase *PruebaEmpleado* muestra toda la información del gerente.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	116/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Sobrecarga (overloading) vs Sobrescritura (overriding)

Una de las cosas que más confunden a los programadores novatos son las diferencias entre los conceptos de **sobrecarga** y **sobrescritura**.

La **sobrescritura** es un concepto que tiene sentido en la **herencia** y se refiere al hecho de **volver a definir** un método heredado.

La **sobrecarga** sólo tiene sentido en la clase misma y se refiere a la posibilidad de definir varios métodos con el **mismo nombre**, pero **con diferentes tipos y número de parámetros**.


Destrucción de objetos (liberación de memoria)

Como los objetos se asignan **dinámicamente**, cuando estos objetos se destruyen será necesarios verificar que la memoria ocupada por ellos ha quedado liberada para usos posteriores. El procedimiento es distinto según el tipo de lenguaje utilizado. Por ejemplo, en C++ los objetos asignados dinámicamente se deben liberar utilizando un operador *delete* y en Java y C# se hace de modo automático utilizando una técnica conocida como **recolección de basura** (garbage collection).

Cuando no existe ninguna referencia a un objeto se supone que ese objeto ya no se necesita y la memoria ocupada por ese objeto puede ser recuperada (liberada), entonces el sistema se ocupa automáticamente de liberar la memoria.

Sin embargo, no se sabe exactamente cuándo se va a activar el **garbage collector**, si no falta memoria es posible que no se llegue a activar en ningún momento. Por lo cual no es conveniente confiar en él para la realización de otras tareas más críticas.

Java no soporta destructores pero existe el método *finalize()* que es lo más aproximado a un destructor. Las tareas dentro de este método serán realizadas cuando el objeto se destruya y se active el **garbage collector**.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	117/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía

Sierra Katy, Bates Bert

SCJP Sun Certified Programmer for Java 6 Study Guide

Mc Graw Hill

Martín, Antonio

Programador Certificado Java 2.

Segunda Edición.

México

Alfaomega Grupo Editor, 2008

Joyanes, Luis

Fundamentos de programación. Algoritmos, estructuras de datos y objetos.

Cuarta Edición

México

Mc Graw Hill, 2008