



4. Manejo de métodos

En la programación orientada a objetos, uno de los fines primordiales es minimizar el tiempo y maximizar el desarrollo y la funcionalidad de las aplicaciones, esto se logra mediante el uso de métodos implementados, los cuales definen el comportamiento de los objetos de una clase. Por convención, Java establece diferentes tipos de métodos constructores y de comportamiento, los cuales se pueden sobrecargar con el propósito de manejar aplicaciones con gran versatilidad. A continuación se describen cada uno de los conceptos.

Tema 1. Métodos

Son utilizados para determinar un grupo de instrucciones que son separadas y que definen un comportamiento que requiere en algunos casos de valores para su proceso.

Un método es algo muy utilizado en la programación de Java y en POO, ya que representa las conductas de los objetos que se traten.

El método ayuda a definir un cambio en algún objeto y es importante que se entienda cómo funciona y utiliza al hacer alguna aplicación o un applet, pues es clave y pieza principal de la programación orientada a objetos.

Pueden ser de un tipo específico como int, double, string, cuenta, entre otros, o simplemente no tener algún tipo como void.

Cuando un método es void, no se espera que regrese algún valor a la aplicación que utilizó algún objeto como parte de la definición de la clase.

Si un método es definido como int, entonces debe utilizar la instrucción return y el valor int calculado.

Algunos ejemplos son:

```
public int obtenValor() {  
    return valor; // donde valor es de tipo entero  
}  
public double obtenSaldo() {  
    return saldo; // donde saldo es de tipo double  
}  
public void cambiaSaldo(saldo) {
```





```
this.saldo = saldo; //igualando la variable de la clase al parámetro  
}
```

Paso de parámetros

La manera en la que el método utiliza el parámetro que le es pasado, es la que define qué tipo de paso de parámetro es.

Por valor

Cuando a un método se le pasa un parámetro de algún tipo de dato primitivo (char, int, double, entre otros), lo que hace éste internamente en memoria es sacar una copia del valor que se está pasando para no afectar a la variable que ha sido enviada a través del método, por ejemplo:

Si se tiene la variable int x en un método llamado uno y este llama al método dos y este también tiene una variable x, el método dos saca una copia de x, y aunque es bautizada con el mismo nombre, se respeta el valor de x en el método uno.
Ejemplo:

```
public void uno() {  
    int x = 10;  
    dos(x);  
    System.out.println(" x en uno = " + x);  
}  
public void dos (int x) {  
    x = x + 2;  
    System.out.println(" x en dos = " + x);  
}
```

Al ejecutar estos métodos en alguna aplicación, se observa cómo se despliega:

```
x en dos = 12  
x en uno = 10
```

Mientras que la x que se mandó a dos se copió con el valor 10 y se le sumaron 2, dando 12, la x que está en uno sigue valiendo 10 porque se saca una copia del valor para enviarse a dos.





Paso de parámetros por referencia

Definición de la clase punto

Esta clase ayuda a visualizar el comportamiento de cualquier objeto punto con sus diferentes tipos de métodos:

```
public class Punto {  
    private int x; // variable para la coordenada en x  
    private int y; // variable para la coordenada en y  
    public Punto() { // método para construir un objeto sin parámetros  
        x = 0;  
        y = 0;  
    }  
    // método para construir un objeto con parámetros  
    public Punto(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public int obtenX() { // método que da el valor de la coordenada x  
        return x;  
    }  
    public int obtenY() { // método que da el valor de la coordenada y  
        return y;  
    }  
    // método que sirve para cambiar el valor //de la coordenada x  
    public void cambiaX(int x) {  
        this.x = x; // this se usa porque se está utilizando (x)  
        // como parámetro y como  
        // variable de instancia y esto es para que no se confunda  
    }  
    // método que sirve para cambiar el valor de la coordenada y  
    public void cambiaY(int y) {  
        this.y = y; // this se usa porque se está utilizando (y)  
        // como parámetro y como  
        // variable de instancia y esto es para que no se confunda Java  
    }  
}
```





Variables de instancia

Se utiliza primero que nada las variables de clase llamadas variables de instancia, ya que con cada objeto que se cree de la clase, se concibe una instancia de estas para cada objeto diferente; estas variables deben definirse como privadas.

Métodos constructores

Posteriormente se revisa que haya dos métodos que se llamen igual a la clase, pero que no manejen algún tipo, es decir int, double, string, entre otros; solamente se dice que son públicos. A estos métodos se les llama constructores y son los que se emplean cuando se crea un objeto de la clase. Es muy importante que un método constructor tenga definido internamente la asignación para cada una de las variables de instancia.

Métodos de acceso

Posteriormente se definen los métodos `obtenVariable()`, llamados métodos de acceso, es decir se define un procedimiento por cada variable de instancia (de acuerdo al tipo), esto es algo que permite obtener el valor de cada variable, ya que han sido definidas como privadas para que alguien que le dé otra clase, no las pueda modificar directamente.

Métodos modificadores

Luego se definen los métodos modificadores `cambiaVariable()`, de tipo void (ya que no regresan valor), éstos deben cambiar el valor de cada variable y sólo a través de ellos es que se deben modificar los valores de las variables de instancia, ya que ellas son privadas.

Por convención de Java, inclusive dentro de la misma clase, si se desea cambiar el valor de una variable, deberán utilizarse los métodos modificadores y de acceso, es decir, que si se quiere incrementar la x en 1, se puede hacer:

```
x++;  
O  
x = x + 1;
```

Para respetar las normas de convención de código Java se deben emplear los métodos modificadores y de acceso, de manera que quede así:





```
cambiaX ( obtenX() + 1 );
```

Aunque se ve mucho más largo o mucho más difícil de entender tiene su porqué, debido a que si algo se debe de hacer cuando se modifique el valor de x o cuando se obtenga su valor, por ejemplo, se hace en el método modificar o de acceso, según corresponda; de esta manera no se pierde el control del comportamiento de las variables de instancia del objeto y es así como se mantiene la integridad. Este es un muy buen hábito de programación.

Sólo los métodos constructores y los de acceso o de modificación pueden adherirse directamente a las variables de instancia de la clase.

En esta clase se desea un método que sirva para dibujar un punto en la coordenada (x,y), entonces se crea un nuevo método llamado `dibujaPunto()`, el cual requerirá un elemento gráfico como el que hasta ahora se ha utilizado con un applet.

```
public void paint(Graphics g)
```

Si el método `dibujaPunto()` de la clase `punto` no recibe como parámetro el objeto gráfico de la clase `graphics`, no habrá manera que pueda dibujar el punto.

Observe como quedaría la clase `punto` modificada:

```
import java.awt.*;
```

```
public class Punto {  
    private int x; // variable para la coordenada en x  
    private int y; // variable para la coordenada en y  
    public Punto() { // método para construir un objeto sin parámetros  
        x = 0;  
        y = 0;  
    }  
    // método para construir un objeto con parámetros  
    public Punto(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public int obtenX() { // método que da el valor de la coordenada x  
        return x;  
    }  
    public int obtenY() { // método que da el valor de la coordenada y
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



```
return y;
}
// método que sirve para cambiar el valor //de la coordenada x
public void cambiaX(int x) {
    this.x = x; // this se usa porque se está utilizando (x)
// como parámetro y como
// variable de instancia y esto es para que no se confunda
}
// método que sirve para cambiar el valor de la coordenada y
public void cambiaY(int y) {
    this.y = y; // this se usa porque se está utilizando (y)
//como parámetro y como
// variable de instancia y esto es para que no se confunda Java
}

public void dibujaPunto(Graphics g) {
    g.fillOval(obtenX() , obtenY(), 5, 5);
}
}
```

Es importante aquí hacer notar que para dibujar el punto se utiliza el método `obtenX()` y el método `obtenY()` y que cuando se pueda utilizar “x” y “y” directamente, se usan los métodos, ya que esto representa una norma de la programación orientada a objetos.

Esta clase se utiliza creando un objeto dentro de un applet y haciendo uso de sus métodos:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
// <applet width="200" height="200" code="AppletMetodos1"></applet>
public class AppletMetodos1 extends Applet implements ActionListener {
    Label l1, l2, l3, l4;
    TextField t1, t2, t3, t4;
    Button b;
    Punto p;

    public AppletMetodos1() {
        p = new Punto();
        l1 = new Label("X");
        t1 = new TextField(4);
    }
}
```



```
import javax.swing.*;
import java.awt.*;
import java.awt.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {
    private JLabel widthLabel, areaLabel;
    private JTextField lengthText, widthText, areaText;
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java

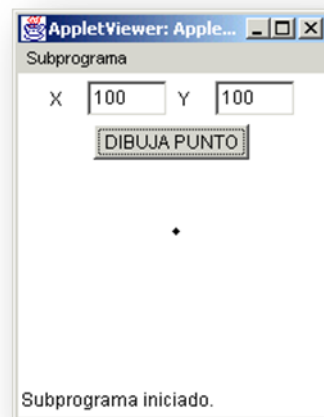


```
l2 = new Label("Y");
t2 = new TextField(4);
b = new Button("DIBUJA PUNTO");
add(l1);
add(t1);
add(l2);
add(t2);
add(b);
b.addActionListener(this);
}

public void paint(Graphics g) {
    p.dibujaPunto(g);
}

public void actionPerformed(ActionEvent ae) {
    int x = Integer.parseInt(t1.getText());
    int y = Integer.parseInt(t2.getText());
    p = new Punto(x, y);
    repaint();
}
}
```

El applet ejecutado se visualiza así:



Fuente: SENA

Es posible visualizar un punto grande en la coordenada 100, 100, este se hizo con el método `fillOval` y dándole un diámetro de más de un píxel para que se note la diferencia.



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {
    private JLabel widthLabel, areaLabel;
    private JTextField lengthText, widthText, areaText;
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



Al utilizar un método o al llamarlo (invocar) se pueden pasar valores y a su vez, este puede utilizar aquellos que han sido definidos en forma global, por ejemplo, en la clase punto, dentro del método paint(), se puede utilizar x o y, ya que ambas son variables de instancia definidas arriba de todos los métodos, por lo tanto todos estos pueden utilizar el valor de x o y.

Por otro lado la clase punto no es un applet y no tiene la capacidad de dibujar un elemento gráfico por sí solo, por eso desde el método paint() del applet se pasa al método paint de la clase punto al parámetro gráfico g, y de esa manera dentro del método paint() de la clase punto es como se puede dibujar.

Es importante notar que en la clase punto al introducir el método paint(), se hizo en import java.awt.*; ya que ahora que se dibuja se requiere utilizar el método fillRect() el cual pertenece a los elementos de la clase graphics que está dentro de ese paquete.

Un método es declarado como void cuando no necesita regresar un valor a quien lo llamó. En caso de requerir retornar un valor este puede ser de los mismos tipos que las variables definidas al principio:

Tipo de método	Descripción formato/tamaño
Byte	Entero de tamaño byte 8-bits a complementos a dos.
Short	Entero, corto 16-bits a complementos a dos.
Int	Entero 32-bits a complementos a dos.
Long	Entero, grande 64-bits a complementos a dos.
Float	Punto flotante de precisión sencilla 32-bits IEEE 754.
Double	Punto flotante de doble precisión 64-bits IEEE 754.
Char	Un solo carácter de 16-bits de tipo Unicode.
Boolean	Valor booleano, true or false.
NombreClase	Alguna clase existente, representará un objeto de alguna clase conocida.

Las variables y objetos definidos y creados dentro de un procedimiento se consideran locales al método y sólo pueden ser utilizados dentro de él. Si al llamar a otro método se quiere que éste utilice cualquier variable o algún objeto, dicho método debe de tomarlo a través de un parámetro para tal efecto.

Es muy importante que los métodos se llamen con el número exacto de variables y objetos definidos como parámetros, así como con el tipo correspondiente de cada uno de ellos.



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortaminAreaCalculator extends JFrame  
{  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;  
}
```





Tema 2. Posibles problemas con la creación de un objeto

Es muy común que cuando se ejecute el método `paint()` en el applet anterior, este no haya originado aún el objeto punto `p`, de manera que no existiría un objeto creado para dibujar una modificación a este applet; para que funcione sin problemas se debe revisar que el objeto `p` haya sido creado para poder graficarlo, entonces el método recomendado para `paint()` sería:

```
public void paint(Graphics g) {  
    if (p != null) {  
        p.dibujaPunto(g);  
    }  
}
```

Así entonces se revisa primero si el objeto no es `null` (o sea que ya tiene algo adentro) y entonces se dibuja el punto `p` utilizando para ello el método `dibujaPunto` de la clase punto.

Identificadores de acceso de un método

De acuerdo a la manera de adherirse a un método, este puede ser declarado como:

- **Public:** cualquier objeto de otra clase puede utilizar el método sobre un objeto de esta clase.

Ejemplos:

```
g.drawString(" Hola ", 100, 100);  
t.setText("");  
p.paint(g);
```

- **Private:** sólo un elemento de la clase puede emplear al método, ningún objeto de otra clase puede usar este método.
- **Protected:** solamente elementos de la clase o de los descendientes (por herencia) pueden usar este método.
- **Static:** la mejor manera de acceder al método `static` es utilizando la clase, éste no se liga como comportamiento de algún objeto de la clase, sino con el comportamiento de la clase en sí.





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



Un ejemplo del uso de este método es el `Integer.parseInt(t.getText())`, usado anteriormente. El método `parseInt` es declarado como `static` dentro de la clase `Integer`, pues su servicio no está supeditado a un objeto; esta cláusula `static` puede ir combinada con `public`, `private` o `protected`.

A continuación se utiliza la clase `Cuenta`, la cual representa una cuenta bancaria; dicha clase manejará una sola variable de instancia que se referirá al saldo, saliendo lo siguiente:

```
public class Cuenta {  
    private double saldo; // variable para manejar el saldo  
    public Cuenta() { // método para construir una cuenta vacía  
        saldo = 0.0;  
    }  
  
    // método para construir una cuenta con un saldo inicial  
    public Cuenta(double saldo) {  
        this.saldo = saldo;  
    }  
  
    // método que da el saldo de una cuenta  
    public double obtenSaldo() {  
        return saldo;  
    }  
  
    // método que sirve para cambiar el valor del saldo  
    public void cambiaSaldo(double saldo) {  
        // this se usa porque se está utilizando (saldo) como parámetro y como  
        this.saldo = saldo;  
        // variable de instancia y esto es para que no se confunda  
    }  
}
```

En esta clase se usan todos los métodos como públicos, ya que lo importante es que desde cualquier applet se puedan crear objetos de la clase `Cuenta`, simplemente definiendo algún objeto y creándolo con alguno de los constructores, por ejemplo:

```
Cuenta Juan = new Cuenta();
```

```
Cuenta Pedro = new Cuenta(1500.00);
```



import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

SERVICIO NACIONAL DE APRENDIZAJE

public class PortafolioAreaCalculator extends JFrame
{
 private JLabel widthLabel, areaLabel;
 private JTextField lengthText, widthText, areaText;
}





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



Ambas instrucciones crean objetos de la clase cuenta, el de Juan tiene un saldo en ceros, pero el de Pedro tiene un saldo de 1500. Ahora se hará uso de un método privado que en realidad no tiene sentido, pero que ejemplifica la siguiente definición de clase:

```
public class Cuenta {  
    private double saldo; // variable para manejar el saldo  
    public Cuenta() { // método para construir una cuenta vacía  
        saldo = 0.0;  
    }  
  
    // método para construir una cuenta con un saldo inicial  
    public Cuenta(double saldo) {  
        this.saldo = saldo;  
    }  
  
    // método que da el saldo de una cuenta  
    public double obtenSaldo() {  
        return saldo;  
    }  
  
    // método privado para poner a cero el saldo  
    private void conCero() {  
        saldo = 0.0;  
    }  
  
    // método que sirve para cambiar el valor del saldo  
    public void cambiaSaldo(double saldo) {  
        // this se usa porque se está utilizando (saldo) como parámetro y como  
        this.saldo = saldo;  
        // variable de instancia y esto es para que no se confunda  
    }  
}
```

El método privado conCero() pone en cero el saldo que se tiene, pero en realidad no se necesita, pues el método cambiaSaldo() que es quien lo llama está cambiando el valor del saldo, pero aquí lo que se quiere es hacer ver el uso de este, ejemplo:

```
Cuenta Juan = new Cuenta(700.00);
```

```
Juan.conCero();
```



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



Esto daría un error, pues el método `conCero()` es privado y no hay manera de utilizarlo, sino solamente desde la clase, tal y como se ha estado haciendo.

Un ejemplo de un método estático sería hacer uso de este a través del nombre de la clase y no del objeto.

Se efectúa un método que los regrese, es decir un string (cadena de caracteres), cuando sea utilizado el método, por ejemplo:

```
public class Cuenta {  
    private double saldo; // variable para manejar el saldo  
    public Cuenta() { // método para construir una cuenta vacía  
        saldo = 0.0;  
    }  
  
    // método para construir una cuenta con un saldo inicial  
    public Cuenta(double saldo) {  
        this.saldo = saldo;  
    }  
  
    // método que da el saldo de una cuenta  
    public double obtenSaldo() {  
        return saldo;  
    }  
  
    // método privado para poner a cero el saldo  
    private void conCero() {  
        saldo = 0.0;  
    }  
  
    // sólo para enseñar el uso del método estático  
    public static String mensaje() {  
        return "Uso del método estático";  
    }  
  
    // método que sirve para cambiar el valor del saldo  
    public void cambiaSaldo(double saldo) {  
        // this se utiliza porque se está usando (saldo) como parámetro y como  
        this.saldo = saldo;  
        // variable de instancia y esto es para que no se confunda  
    }  
}
```



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



En este caso el método del mensaje es público y estático, de hecho puede ser utilizado desde fuera de la clase y no es necesario que sea referenciado por algún objeto, sino con el nombre de la clase, por ejemplo:

```
Cuenta Pedro = new Cuenta(1500.00);
```

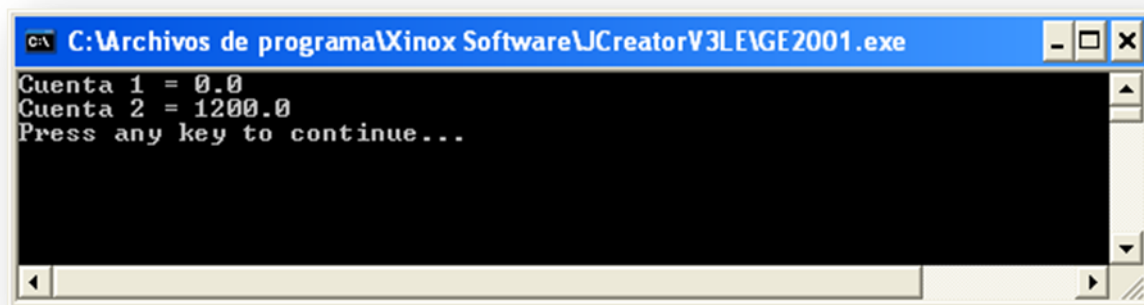
```
t.setText("El mensaje estático es = " + Cuenta.mensaje() );
```

De esta manera al hacer uso del método mensaje, regresará el mensaje definido en la clase.

Observe el siguiente ejemplo de la aplicación:

```
public class PruebaCuenta {  
    public static void main(String[] args) {  
        Cuenta cuenta1, cuenta2;  
        cuenta1 = new Cuenta();  
        cuenta2 = new Cuenta(1200.00);  
        System.out.println("Cuenta 1 = " + cuenta1.obtenSaldo());  
        System.out.println("Cuenta 2 = " + cuenta2.obtenSaldo());  
    }  
}
```

En este ejemplo se crean dos objetos de la clase cuenta y se despliega la ejecución de esta aplicación, así como se ve en la siguiente imagen:



Fuente: SENA

Ahora se utiliza el método conCero() con la siguiente aplicación:

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



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame  
{  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



```
Cuenta cuenta1, cuenta2;  
cuenta1 = new Cuenta();  
cuenta2 = new Cuenta(1200.00);  
System.out.println("Cuenta 1 = " + cuenta1.obtenSaldo());  
System.out.println("Cuenta 2 = " + cuenta2.obtenSaldo());  
cuenta2.conCero();  
System.out.println("Cuenta 1 = " + cuenta1.obtenSaldo());  
System.out.println("Cuenta 2 = " + cuenta2.obtenSaldo());  
}  
}
```

Esta aplicación no compila, ya que el método `conCero()` dirá que tiene acceso privado en la clase, por lo tanto sólo puede ser utilizado en la clase, ejemplo:

La clase cuenta quedaría:

```
public class Cuenta {  
    private double saldo; // variable para manejar el saldo  
    public Cuenta() { // método para construir una cuenta vacía  
        saldo = 0.0;  
    }  
  
    // método para construir una cuenta con un saldo inicial  
    public Cuenta(double saldo) {  
        this.saldo = saldo;  
    }  
  
    // método que da el saldo de una cuenta  
    public double obtenSaldo() {  
        return saldo;  
    }  
  
    // método privado para poner a cero el saldo  
    private void conCero() {  
        saldo = 0.0;  
    }  
  
    // solo para enseñar el uso del método estático  
    public static String mensaje() {  
        return "Uso del método estático";  
    }  
}
```



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame  
{  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



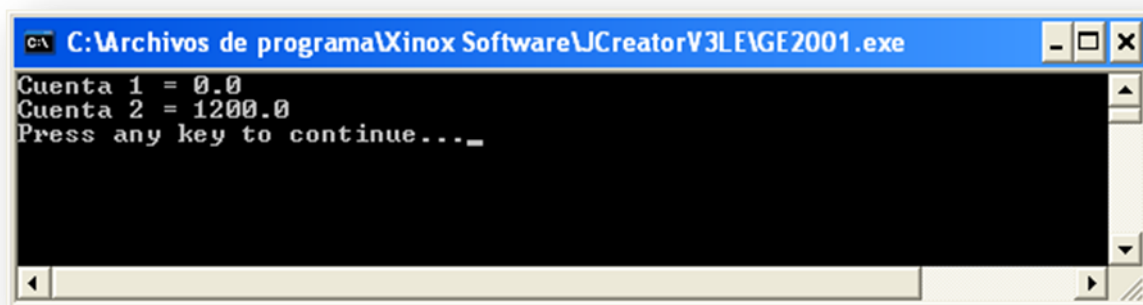
```
// borra el saldo
public void borra() {
    conCero();
}

// método que sirve para cambiar el valor del saldo
public void cambiaSaldo(double saldo) {
    // this se usa porque se está utilizando (saldo) como parámetro y como
    this.saldo = saldo;
}
// variable de instancia y esto es para que no se confunda
}
```

Y la aplicación sería:

```
public class PruebaCuenta {
    public static void main(String[] args) {
        Cuenta cuenta1, cuenta2;
        cuenta1 = new Cuenta();
        cuenta2 = new Cuenta(1200.00);
        System.out.println("Cuenta 1 = " + cuenta1.obtenSaldo());
        System.out.println("Cuenta 2 = " + cuenta2.obtenSaldo());
        cuenta2.borra();
        System.out.println("Cuenta 1 = " + cuenta1.obtenSaldo());
        System.out.println("Cuenta 2 = " + cuenta2.obtenSaldo());
    }
}
```

La ejecución de esta aplicación se visualizaría así:



Fuente: SENA



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {
    private JLabel widthLabel, areaLabel;
    private JTextField lengthText, widthText, areaText;
```





Si recuerda lo que se mencionó anteriormente sobre la convención de Java, una de ellas no ha sido respetada en la clase cuenta, la de borrar el saldo, es decir para hacer el saldo cero.

La clase cuenta quedaría como:

```
public class Cuenta {  
    private double saldo; // variable para manejar el saldo  
    public Cuenta() { // método para construir una cuenta vacía  
        saldo = 0.0;  
    }  
  
    // método para construir una cuenta con un saldo inicial  
    public Cuenta(double saldo) {  
        this.saldo = saldo;  
    }  
  
    // método que da el saldo de una cuenta  
    public double obtenSaldo() {  
        return saldo;  
    }  
  
    // metodo privado para poner a cero el saldo  
    private void conCero() {  
        cambiaSaldo(0.0);  
    }  
  
    // solo para enseñar el uso del método estático  
    public static String mensaje() {  
        return "Uso del método estático";  
    }  
  
    // borra el saldo  
    public void borra() {  
        conCero();  
    }  
  
    // método que sirve para cambiar el valor del saldo  
    public void cambiaSaldo(double saldo) {  
        // this se usa porque se está utilizando (saldo) como parámetro y como  
        this.saldo = saldo;  
    }  
}
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



```
// variable de instancia y esto es para que no se confunda  
}  
}
```

La aplicación sería la misma:

```
C:\Archivos de programa\Xinox Software\JCreatorV3\LEGE2001.exe  
Cuenta 1 = 0.0  
Cuenta 2 = 1200.0  
Cuenta 1 = 0.0  
Cuenta 2 = 0.0  
Press any key to continue...
```

Fuente: SENA

Utilizando ahora el método estático, se observa que la aplicación cambia a la siguiente:

```
public class PruebaCuenta {  
    public static void main(String[] args) {  
        Cuenta cuenta1, cuenta2;  
        cuenta1 = new Cuenta();  
        cuenta2 = new Cuenta(1200.00);  
        System.out.println("Cuenta 1 = " + cuenta1.obtenSaldo());  
        System.out.println("Cuenta 2 = " + cuenta2.obtenSaldo());  
        cuenta2.borra();  
        System.out.println("Cuenta 1 = " + cuenta1.obtenSaldo());  
        System.out.println("Cuenta 2 = " + cuenta2.obtenSaldo());  
        System.out.println("Mensaje = " + Cuenta.mensaje());  
        System.out.println("Mensaje = " + cuenta1.mensaje());  
    }  
}
```

En esta aplicación se observa el uso del mensaje de dos maneras:

```
System.out.println("Mensaje = " + Cuenta.mensaje());
```



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;
```





```
System.out.println("Mensaje = " + cuenta1.mensaje());
```

En la primera se está utilizando la clase cuenta como se ha hecho anteriormente con otros métodos estáticos como `Math.pow()` o `Integer.parseInt()`. Se visualiza también que el mismo método se está usando con el objeto `cuenta1`, lo cual es posible ya que un objeto de la clase cuenta puede tener acceso al método estático.

Tema 3. Sobrecarga de métodos

El encabezado o firma de un método (la primera línea en el) dice que el modificador tiene el método (público, protegido o privado), su tipo de retorno (void o alguno de los tipos previamente vistos) y los parámetros que requiere para trabajar.

Un método puede definirse varias veces (sobrecargarse) en la misma clase, pero cambiando su encabezado de manera que sean otros los parámetros a utilizar. En C++ se pueden sobrecargar operadores como el +, el -, entre otros, pero en Java no, en éste el operador + se puede utilizar para sumar números o para concatenar strings solamente; la sobrecarga se puede realizar en métodos constructores o no constructores.

Para ejemplificar la sobrecarga de métodos, se supone que se tiene alguna clase en la que se lleva el control del inventario de un artículo y puede ser que la cantidad que se reciba sea un entero, pero también puede que sea un número de doble precisión o un objeto tipo string, entonces se puede tener el mismo método definido tres veces:

```
public void cambia( int numero) {  
    inventario = numero;  
}
```

```
public void cambia( double numero) {  
    // esto es la conversión de tipos usado también en C++  
    inventario = (int) numero;  
}
```

```
public void cambia( String numero) {  
    // asumiendo que inventario es entero  
    inventario = Integer.parseInt(numero);  
}
```





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



Cuando el método sea llamado, Java revisará cual es el método a utilizar dependiendo de la firma y del parámetro pasado.

Esto es muy utilizado también en los constructores de los objetos, por ejemplo en la clase punto, en la que se puede construir un objeto punto a partir de dos valores enteros, pero al suponer que también quisieran ser construidos a través de dos valores de precisión doble (double) o que se quisiera construir un objeto a partir de otro objeto punto. Quedaría así:

```
public class Punto {  
    private int x; // variable para la coordenada en x  
    private int y; // variable para la coordenada en y  
  
    // método para construir un objeto sin parámetros  
    public Punto() {  
        x = 0;  
        y = 0;  
    }  
  
    // método para construir un objeto con parámetros enteros  
    public Punto(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    // método para construir un objeto con parámetros double  
    public Punto(double x, double y) {  
        this.x = (int) x;  
        this.y = (int) y;  
    }  
  
    // método para construir un objeto con parámetro objeto Punto  
    public Punto(Punto obj) {  
        this.x = obj.x;  
        this.y = obj.y;  
    }  
  
    // método que da el valor de la coordenada x  
    public int obtenX() {  
        return x;  
    }  
  
    // método que da el valor de la coordenada y
```



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;
```





```
public int obtenY() {  
    return y;  
}
```

// método que sirve para cambiar el valor de la coordenada x

```
public void cambiaX(int x) {  
    // this se usa porque se está utilizando (x) como parámetro y como  
    // variable de instancia y esto es para que no se confunda  
    this.x = x;  
}
```

// método que sirve para cambiar el valor de la coordenada y

```
public void cambiaY(int y) {  
    // this se usa porque se está utilizando (y) como parámetro y como  
    // variable de instancia y esto es para que no se confunda  
    this.y = y;  
}  
}
```

Ahora se manejan diferentes constructores para la clase, esto ya se había hecho antes, y es entonces por lo que también los constructores se pueden definir de varias maneras; de igual manera se puede redefinir algún o algunos métodos.

Al suponer que los valores de x o de y sean tomados no solamente de enteros, sino de valores tipo double, entonces se tendrá una clase como la siguiente:

```
public class Punto {  
    private int x; // variable para la coordenada en x  
    private int y; // variable para la coordenada en y
```

// método para construir un objeto sin parámetros

```
public Punto() {  
    x = 0;  
    y = 0;  
}
```

// método para construir un objeto con parámetros enteros

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





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



// método para construir un objeto con parámetros double

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

// método para construir un objeto con parámetro objeto Punto

```
public Punto(Punto obj) {  
    this.x = obj.x;  
    this.y = obj.y;  
}
```

// método que da el valor de la coordenada x

```
public int obtenX() {  
    return x;  
}
```

// método que da el valor de la coordenada y

```
public int obtenY() {  
    return y;  
}
```

// método que sirve para cambiar el valor de la coordenada x

```
public void cambiaX(int x) {  
    // this se usa porque se está utilizando (x) como parámetro y como  
    // variable de instancia y esto es para que no se confunda  
    this.x = x;  
}
```

// método que toma un double para cambiar a x

```
public void cambiaX(double x) {  
    // this se usa porque se está utilizando (x) como parámetro y como  
    // variable de instancia y esto es para que no se confunda  
    this.x = (int) x;  
}
```

// método que sirve para cambiar el valor de la coordenada y

```
public void cambiaY(int y) {  
    // this se usa porque se está utilizando (y) como parámetro y como  
    // variable de instancia y esto es para que no se confunda  
    this.y = y;  
}
```



```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {  
    private JLabel widthLabel, areaLabel;  
    private JTextField lengthText, widthText, areaText;
```





```
// método que toma un double para cambiar a y
public void cambiaY(double y) {
    // this se usa porque se está utilizando (y) como parámetro y como
    // variable de instancia y esto es para que no se confunda
    this.y = (int) y;
}

// método para obtener el objeto en formato String
public String toString() {
    return "(" + obtenX() + "," + obtenY() + ")";
}
}
```

Una aplicación no gráfica que use esta clase para entender su manejo puede ser:

```
public class AplicacionPunto {
    private static Punto a, b, c, d, e;

    public static void main(String[] args) {
        a = new Punto();
        b = new Punto(1, 2);
        c = new Punto(3.0, 4.0);
        d = new Punto(b);
        e = new Punto(c);
        System.out.println(" Punto a = " + a.toString());
        System.out.println(" Punto b = " + b.toString());
        System.out.println(" Punto c = " + c.toString());
        System.out.println(" Punto d = " + d.toString());
        System.out.println(" Punto e = " + e.toString());
        a.cambiaX(5.0);
        c.cambiaY(-3.0);
        System.out.println("Valores cambiados");
        System.out.println(" Punto a = " + a.toString());
        System.out.println(" Punto c = " + c.toString());
    }
}
```

Ambas clases deben estar en la misma carpeta para que se pueda utilizar la aplicación.





Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java



La forma en la que se puede visualizar esto es compilando como una aplicación no gráfica y ejecutándola tal y como se ve a continuación:

```
C:\Archivos de programa\Xinox Software\JCreatorV3\JL\IGE2001.exe
Punto a = <0,0>
Punto b = <1,2>
Punto c = <3,4>
Punto d = <1,2>
Punto e = <3,4>
Valores cambiados
Punto a = <5,0>
Punto c = <3,-3>
Press any key to continue...
```

Fuente: SENA

De esta manera se observa la forma de crear diferentes objetos de la clase punto, y cómo es posible cambiar el valor de alguna de las variables de instancia de esos objetos utilizando un número de punto flotante en vez de un número entero. Se ha hecho uso de una aplicación no gráfica para que se enfatice el uso de este concepto solamente.

Note que en la clase punto se utiliza el método toString(), muy usado en todas las clases singulares, ya que es la manera de representar cualquier objeto de la clase en formato string.



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

SERVICIO NACIONAL DE APRENDIZAJE

```
public class PortafolioAreaCalculator extends JFrame {
    private JLabel widthLabel, areaLabel;
    private JTextField lengthText, widthText, areaText;
```





Referencias

- Sánchez, M. (s.f.). *Gestión de eventos*. Consultado el 24 de abril 2014, en <http://odelys2003.files.wordpress.com/2013/01/1-1-javagestioneventos-100310084758-phpapp01.pdf>
- Servicio Nacional de Aprendizaje, SENA. (2010). *Desarrollo de aplicaciones con interfaz gráfica, manejo de eventos, clases y objetos: Java*. Colombia: Autor.

Control del documento

	Nombre	Cargo	Dependencia	Fecha
Autor	Jorge Hernán Moreno Tenjo	Instructor	Centro Metalmecánico. Regional Distrito Capital	Mayo de 2013
Adaptación	Ana María Mora Jaramillo	Guionista - Línea de producción	Centro Agroindustrial. Regional Quindío	Mayo de 2014
	Rachman Bustillo Martínez	Guionista - Línea de Producción	Centro Agroindustrial. Regional Quindío	Mayo de 2014

