



Ministerio de Producción  
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



**PROGRAMACIÓN ORIENTADA  
A OBJETOS**



# Programación Orientada a Objetos

## Temas:

La palabra clave **static**

Variables y métodos de clase

Ocultamiento de variables y métodos de clase

La palabra clave **final**

Atributos constantes

Constantes locales

Métodos que no pueden sobreescribirse



# La Palabra Clave Static

- Usualmente, al crear una clase se describe cómo se conformarán los objetos y cómo se comportarán.
  - No se tiene un objeto hasta que no se crea explícitamente uno.
- Esto **NO** siempre es suficiente.
  - Se quiere tener un único valor para un atributo de una clase, independientemente de la cantidad de objetos creados (cero o más).
  - Se necesita un método que no se encuentre asociado a los objetos de la clase.
    - Un método que pueda ser invocado incluso cuando no existan objetos.



# La Palabra Clave Static

- Usualmente, al crear una clase se describe cómo se conformarán los objetos y cómo se comportarán.
  - No se tiene un objeto hasta que no se crea explícitamente uno.
- Esto **NO** siempre es suficiente.
  - Se quiere tener un único valor para un atributo de una clase, independientemente de la cantidad de objetos creados (cero o más).
  - Se necesita un método que no se encuentre asociado a los objetos de la clase.
    - Un método que pueda ser invocado incluso cuando no existan objetos.

¿Qué hacer?

# La Palabra Clave Static

- Usualmente, al crear una clase se describe cómo se conformarán los objetos y cómo se comportarán.
  - No se tiene un objeto hasta que no se crea explícitamente uno.
- Esto **NO** siempre es suficiente.
  - Se quiere tener un único valor para un atributo de una clase, independientemente de la cantidad de objetos creados (cero o más).
  - Se necesita un método que no se encuentre asociado a los objetos de la clase.
    - Un método que pueda ser invocado incluso cuando no existan objetos.

## ¿Qué hacer?

- Utilizar la palabra reservada **static**.
- Cuando un atributo o método son **static**, no se encuentra asociado a ninguna instancia particular de la clase.
- Se los llama atributos o métodos de clase
- Incluso cuando no haya objetos creados será posible acceder a los atributos y métodos **static**.



# Atributos de Clase

- Los atributos de clase son **atributos compartidos** por todas las instancias de una misma clase.
- Son variables asociadas con la clase y NO con instancias particulares.
- Son **GLOBALES** para la clase.
- Para declarar variables de clase utilizamos la palabra clave **static**.
  - Simplemente hay que colocar **static** antes de la definición.

# Atributos de Clase

- Los atributos de clase son **atributos compartidos** por todas las instancias de una misma clase.
- Son variables asociadas con la clase y NO con instancias particulares.
- Son **GLOBALES** para la clase.
- Para declarar variables de clase utilizamos la palabra clave **static**.
  - Simplemente hay que colocar **static** antes de la definición.
- Por ejemplo: se crea un atributo estático y se lo inicializa.

```
public class StaticTest {  
    protected static int i = 47;  
}
```

# La Palabra Clave Static

- Las variables locales **NO** pueden ser **static**.
- Solo los atributos pueden ser **static**.
- Si el atributo **static** es de un tipo primitivo y no es inicializado, toma el valor standard inicial para su tipo.
- Si el atributo **static** es la referencia a un objeto, toma el valor de null.
- Java permite que las inicializaciones estáticas sean agrupadas dentro de un bloque especial:

```
public class StaticTest {  
    protected static int i;  
    static {  
        i = 47;  
    }  
}
```

- Este código es ejecutado solo una vez.
- - La primera vez que se crea un objeto de esta clase.
  - La primera vez que se accede a un miembro **static** de la clase.





# Atributos de Clase

- Para acceder a un atributo de clase **NO** es necesario crear instancias de la clase. Pueden ser accedidas usando el nombre de la clase y sin haber creado instancias.

**Veamos un ejemplo:** supongamos que estamos desarrollando un sistema para un comercio, en el que tenemos **clientes**, **productos** y realizamos **la facturación** correspondiente a la **compra de productos que hace un cliente**.

# Atributos de Clase

```
package modelo;

public class Cliente {

    public static int siguienteNro = 0;
    private int nroCte;
    private String nombre;
    private String apellido;

    public Cliente(){
        siguienteNro++;
        nroCte=siguienteNro;
    }

    public int getSiguienteNro(){
        return siguienteNro;
    }

    public String toString(){
        return ("Cliente nro: "+nroCte+" Apellido: "+
            apellido+" Nombre: " +nombre);
    }

    // setters y getters
    public int getNroCte() {
        return nroCte;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Necesitamos que un cliente tenga un número de cliente único.  
¿Cómo lo podemos resolver?

Declaramos una variable de clase llamada **siguienteNro** que es accesible por todos y que además es compartida por todas las instancias.

Observemos como la declaramos:

**public static int siguienteNro;**

**siguienteNro** es compartida por todas las instancias de Cliente

Cada vez que creamos un cliente nuevo, incrementamos en 1 la variable compartida.

También podemos escribir **Cliente.siguienteNro++**;

Observemos, que para acceder a una variable de clase usamos el operador ".", al igual que lo hacemos con las variables de instancia, con la diferencia que no lo hacemos sobre una instancia, sino sobre el **nombre de la clase**.

# Atributos de Clase

## Clase de prueba

```
package modelo;

public class TestCliente {

    public static void main(String[] args) {
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
        Cliente c1 = new Cliente();
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
        c1.setApellido("Orlando");
        c1.setNombre("Federico");
        System.out.println(c1);
        System.out.println("valor de siguienteNro en c1: "
            + c1.getSiguienteNro());
        Cliente c2 = new Cliente();
        c2.setApellido("Craviotto");
        c2.setNombre("Martin");
        System.out.println(c2);
        System.out.println("valor de siguienteNro en c1: "
            + c1.getSiguienteNro());
        System.out.println("valor de siguienteNro en c2: "
            + c2.getSiguienteNro());
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
    }
}
```

```
Valor de siguienteNro: 0
Valor de siguienteNro: 1
Cliente nro: 1 Apellido: Orlando Nombre: Federico
valor de siguienteNro en c1: 1
Cliente nro: 2 Apellido: Craviotto Nombre: Martin
valor de siguienteNro en c1: 2
valor de siguienteNro en c2: 2
Valor de siguienteNro: 2
```

Observemos que cada instancia de **Cliente** tiene sus propios valores para las variables de instancia: **c1** tiene a **Orlando** como **apellido** mientras que **c2** tiene a **Craviotto** como apellido. Lo mismo sucede con el nombre y el número de cliente de cada uno de ellos. Ahora, observemos el valor de la variable **siguienteNro** cuando la accedemos a través de las instancias y a través del nombre de la clase: El valor es el mismo: 2. Estamos consultado a una **variable compartida, NO es una variable individual de cada instancia**

# Atributos de Clase

## Clase de prueba

```
package modelo;

public class TestCliente {

    public static void main(String[] args) {
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
        Cliente c1 = new Cliente();
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
        c1.setApellido("Orlando");
        c1.setNombre("Federico");
        System.out.println(c1);
        System.out.println("valor de siguienteNro en c1: "
            + c1.getSiguienteNro());
        Cliente c2 = new Cliente();
        c2.setApellido("Craviotto");
        c2.setNombre("Martin");
        System.out.println(c2);
        System.out.println("valor de siguienteNro en c1: "
            + c1.getSiguienteNro());
        System.out.println("valor de siguienteNro en c2: "
            + c2.getSiguienteNro());
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
    }
}
```

```
Valor de siguienteNro: 0
Valor de siguienteNro: 1
Cliente nro: 1 Apellido: Orlando Nombre: Federico
valor de siguienteNro en c1: 1
Cliente nro: 2 Apellido: Craviotto Nombre: Martin
valor de siguienteNro en c1: 2
valor de siguienteNro en c2: 2
Valor de siguienteNro: 2
```

Observemos que cada instancia de **Cliente** tiene sus propios valores para las variables de instancia: **c1** tiene a **Orlando** como **apellido** mientras que **c2** tiene a **Craviotto** como apellido. Lo mismo sucede con el nombre y el número de cliente de cada uno de ellos. Ahora, observemos el valor de la variable **siguienteNro** cuando la accedemos a través de las instancias y a través del nombre de la clase: El valor es el mismo: 2. Estamos consultado a una **variable compartida, NO es una variable individual de cada instancia**

¿Puedo acceder directamente a **siguienteNro** o estoy obligado a hacerlo a través de **getSiguienteNro()**? ¿Por qué?

Como **siguienteNro** es una variable pública, es lo mismo hacer **Cliente.siguienteNro** que **Cliente.getSiguienteNro()**

# Atributos de Clase

## Clase de prueba

```
package modelo;

public class TestCliente {

    public static void main(String[] args) {
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
        Cliente c1 = new Cliente();
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
        c1.setApellido("Orlando");
        c1.setNombre("Federico");
        System.out.println(c1);
        System.out.println("valor de siguienteNro en c1: "
            + c1.getSiguienteNro());
        Cliente c2 = new Cliente();
        c2.setApellido("Craviotto");
        c2.setNombre("Martin");
        System.out.println(c2);
        System.out.println("valor de siguienteNro en c1: "
            + c1.getSiguienteNro());
        System.out.println("valor de siguienteNro en c2: "
            + c2.getSiguienteNro());
        System.out.println("Valor de siguienteNro: "
            + Cliente.siguienteNro);
    }
}
```

```
Valor de siguienteNro: 0
Valor de siguienteNro: 1
Cliente nro: 1 Apellido: Orlando Nombre: Federico
valor de siguienteNro en c1: 1
Cliente nro: 2 Apellido: Craviotto Nombre: Martin
valor de siguienteNro en c1: 2
valor de siguienteNro en c2: 2
Valor de siguienteNro: 2
```

Observemos que cada instancia de **Cliente** tiene sus propios valores para las variables de instancia: **c1** tiene a **Orlando** como **apellido** mientras que **c2** tiene a **Craviotto** como apellido. Lo mismo sucede con el nombre y el número de cliente de cada uno de ellos. Ahora, observemos el valor de la variable **siguienteNro** cuando la accedemos a través de las instancias y a través del nombre de la clase: El valor es el mismo: 2. Estamos consultado a una **variable compartida, NO es una variable individual de cada instancia**

¿Puedo acceder directamente a **siguienteNro** o estoy obligado a hacerlo a través de **getSiguienteNro()**? ¿Por qué?

Como **siguienteNro** es una variable pública, es lo mismo hacer **Cliente.siguienteNro** que **Cliente.getSiguienteNro()**

¿Es lo mismo hacer **c2.getSiguienteNro()** o **c1.getSiguienteNro()** qué **Cliente.siguienteNro**? ¿Por qué?

**SI!!!** porque tanto **c1** como **c2** en el método **getSiguienteNro()** acceden a la variable de clase compartida **siguienteNro**.

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgrados(double rads){  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ?

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgradados(double rads){  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ?

Vamos a agregar una constante **PI** a **CirculoPlano**.

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgradados(double rads){  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ?

Vamos a agregar una constante **PI** a **CirculoPlano**.



# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgradados(double rads) {  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ?

Vamos a agregar una constante **PI** a **CirculoPlano**.

```
public class TestOcultamiento {
```

```
    public static void main(String args[]){  
        CirculoPlano cp=new CirculoPlano(10, 20, 10);  
        System.out.println("Area : " + cp.area());  
        System.out.println("Circunferencia: " + cp.circunferencia());  
    }  
}
```

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgrados(double rads) {  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public static final double PI = 3.14159265358979323846;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ?

Vamos a agregar una constante **PI** a **CirculoPlano**.

¿ A qué **PI** hacen referencia **area()** y **circunferencia()** ?

```
public class TestOcultamiento {
```

```
    public static void main(String args[]){  
        CirculoPlano cp=new CirculoPlano(10, 20, 10);  
        System.out.println("Area : " + cp.area());  
        System.out.println("Circunferencia: " + cp.circunferencia());  
    }  
}
```

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgradados(double rads) {  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public static final double PI = 3.14159265358979323846;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ?

Vamos a agregar una constante **PI** a **CirculoPlano**.

¿ A qué **PI** hacen referencia **area()** y **circunferencia()** ?

**A la definida en Circulo, PI= 3.14159**

```
public class TestOcultamiento {
```

```
    public static void main(String args[]){  
        CirculoPlano cp=new CirculoPlano(10, 20, 10);  
        System.out.println("Area : " + cp.area());  
        System.out.println("Circunferencia: " + cp.circunferencia());  
    }  
}
```

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgrados(double rads) {  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public static final double PI = 3.14159265358979323846;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ? **SI !!**

Vamos a agregar una constante **PI** a **CirculoPlano**.

¿ A qué **PI** hacen referencia **area()** y **circunferencia()** ?

**A la definida en Circulo, PI= 3.14159**

```
public class TestOcultamiento {
```

```
    public static void main(String args[]){  
        CirculoPlano cp=new CirculoPlano(10, 20, 10);  
        System.out.println("Area : " + cp.area());  
        System.out.println("Circunferencia: " + cp.circunferencia());  
    }  
}
```

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgrados(double rads) {  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public static final double PI = 3.14159265358979323846;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ? **SI !!**

Vamos a agregar una constante **PI** a **CirculoPlano**.

¿ A qué **PI** hacen referencia **area()** y **circunferencia()** ?

**A la definida en Circulo, PI= 3.14159**

```
public class TestOcultamiento {
```

```
    public static void main(String args[]){  
        CirculoPlano cp=new CirculoPlano(10, 20, 10);  
        System.out.println("Area : " + cp.area());  
        System.out.println("Circunferencia: " + cp.circunferencia());  
    }  
}
```

**Area : 314.159**  
**Circunferencia : 62.8318**

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgradados(double rads) {  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

PI y r de Circulo

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public static final double PI = 3.14159265358979323846;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ? **SI !!**

Vamos a agregar una constante **PI** a **CirculoPlano**.

¿ A qué **PI** hacen referencia **area()** y **circunferencia()** ?

**A la definida en Circulo, PI= 3.14159**

```
public class TestOcultamiento {
```

```
    public static void main(String args[]){  
        CirculoPlano cp=new CirculoPlano(10, 20, 10);  
        System.out.println("Area : " + cp.area());  
        System.out.println("Circunferencia: " + cp.circunferencia());  
    }  
}
```

**Area : 314.159**  
**Circunferencia : 62.8318**

# Ocultar Atributos de Clase

```
public class Circulo {  
  
    public static final double PI= 3.14159;  
    private double r;  
  
    public Circulo (double r) { this.r = r; }  
    public static double radianesAgradados(double rads){  
        return rads * 180 / PI;  
    }  
  
    public double area() {  
        return PI * r * r;  
    }  
  
    public double circunferencia() {  
        return 2 * PI * r;  
    }  
}
```

PI y r de Circulo

PI y r de Circulo

```
public class CirculoPlano extends Circulo {  
  
    private double r, cx, cy;  
  
    public static final double PI = 3.14159265358979323846;  
  
    public CirculoPlano(double r, double x, double y) {  
        super(r);  
        this.cx = x; this.cy = y;  
        this.r = Math.sqrt(cx*cx + cy*cy);  
    }  
  
    public boolean pertenece(double x, double y) {  
        double dx = x - cx, dy = y - cy;  
        double distancia = Math.sqrt(dx*dx + dy*dy);  
        return (distancia < r);  
    }  
}
```

¿ Es posible **ocultar variables de Clase** ? **SI !!**

Vamos a agregar una constante **PI** a **CirculoPlano**.

¿ A qué **PI** hacen referencia **area()** y **circunferencia()** ?

**A la definida en Circulo, PI= 3.14159**

```
public class TestOcultamiento {
```

```
    public static void main(String args[]){  
        CirculoPlano cp=new CirculoPlano(10, 20, 10);  
        System.out.println("Area : " + cp.area());  
        System.out.println("Circunferencia: " + cp.circunferencia());  
    }  
}
```

**Area : 314.159**  
**Circunferencia : 62.8318**

# Métodos de Clase

- Se utilizan cuando se necesita utilizar un método que no se encuentra asociado a ninguna instancia particular de la clase.
  - Se necesita invocar el método incluso cuando no hay objetos creados.
- **NO** se pueden invocar métodos no **static** dentro de un método **static**.
  - La reversa **si** es posible.
- Es posible invocar un método estático desde la propia clase que lo define, sin ningún objeto.



# Métodos de Clase

```
package modelo;

public class Cliente {

    private static int siguienteNro = 0;
    private int nroCte;
    private String nombre;
    private String apellido;

    public Cliente(){
        siguienteNro++;
        nroCte=siguienteNro;
    }
    // metodos de clase
    public static int getSiguitenNro(){
        return siguienteNro;
    }
    // metodos de instancia
    public String toString(){
        return ("Cliente nro: "+nroCte+" Apellido: "+
            apellido+" Nombre: " +nombre);
    }
}
```

El método **getSiguienteNro()** ¿devuelve un valor particular de una variable para cada objeto Cliente?

**NO!!!**

Siempre devuelve lo mismo, independientemente del objeto Cliente que recibe el mensaje, ya que retorna el valor de la variable de clase **siguienteNro** que es el mismo para todos los objetos **Cliente**.

# Métodos de Clase

```
package modelo;

public class Cliente {

    private static int siguienteNro = 0;
    private int nroCte;
    private String nombre;
    private String apellido;

    public Cliente(){
        siguienteNro++;
        nroCte=siguienteNro;
    }
    // metodos de clase
    public static int getSiguitenNro(){
        return siguienteNro;
    }
    // metodos de instancia
    public String toString(){
        return ("Cliente nro: "+nroCte+" Apellido: "+
            apellido+" Nombre: " +nombre);
    }
}
```

El método **getSiguienteNro()** ¿devuelve un valor particular de una variable para cada objeto Cliente?

**NO!!!**

Siempre devuelve lo mismo, independientemente del objeto Cliente que recibe el mensaje, ya que retorna el valor de la variable de clase **siguienteNro** que es el mismo para todos los objetos **Cliente**.

**getSiguienteNro()** es un método de clase pues accede a una variable de clase.  
Lo declaramos:

# Métodos de Clase

```
package modelo;

public class Cliente {

    private static int siguienteNro = 0;
    private int nroCte;
    private String nombre;
    private String apellido;

    public Cliente(){
        siguienteNro++;
        nroCte=siguienteNro;
    }
    // metodos de clase
    public static int getSiguienteNro(){
        return siguienteNro;
    }
    // metodos de instancia
    public String toString(){
        return ("Cliente nro: "+nroCte+" Apellido: "+
            apellido+" Nombre: " +nombre);
    }
}
```

El método **getSiguienteNro()** ¿devuelve un valor particular de una variable para cada objeto Cliente?

**NO!!!**

Siempre devuelve lo mismo, independientemente del objeto Cliente que recibe el mensaje, ya que retorna el valor de la variable de clase **siguienteNro** que es el mismo para todos los objetos **Cliente**.

**getSiguienteNro()** es un método de clase pues accede a una variable de clase.

Lo declaramos:

**public static int getSiguienteNro()**

# Métodos de Clase

```
package modelo;

public class Cliente {

    private static int siguienteNro = 0;
    private int nroCte;
    private String nombre;
    private String apellido;

    public Cliente(){
        siguienteNro++;
        nroCte=siguienteNro;
    }
    // metodos de clase
    public static int getSiguienteNro(){
        return siguienteNro;
    }
    // metodos de instancia
    public String toString(){
        return ("Cliente nro: "+nroCte+" Apellido: "+
            apellido+" Nombre: " +nombre);
    }
}
```

El método **getSiguienteNro()** ¿devuelve un valor particular de una variable para cada objeto Cliente?

**NO!!!**

Siempre devuelve lo mismo, independientemente del objeto Cliente que recibe el mensaje, ya que retorna el valor de la variable de clase **siguienteNro** que es el mismo para todos los objetos **Cliente**.

**getSiguienteNro()** es un método de clase pues accede a una variable de clase.

Lo declaramos:

**public static int getSiguienteNro()**

¿Para ejecutar **getSiguienteNro()** necesito crear un objeto **Cliente**?

**NO** necesito tener creado un objeto **Cliente**, puedo acceder a la variable **siguienteNro** usando el nombre de la clase:

**Cliente.getSiguienteNro();**

# Métodos de Clase

Ahora en **TestCliente** invocamos al método al método de clase **getSiguienteNro()**:

```
package modelo;

public class TestCliente {

    public static void main(String[] args) {
        System.out.println("Valor de siguienteNro: "
            + Cliente.getSiguienteNro());
        Cliente c1 = new Cliente();
        System.out.println("Valor de siguienteNro: "
            + Cliente.getSiguienteNro());
        c1.setApellido("Orlando");
        c1.setNombre("Federico");
        System.out.println(c1);
        Cliente c2 = new Cliente();
        c2.setApellido("Craviotto");
        c2.setNombre("Martin");
        System.out.println(c2);
        System.out.println("Valor de siguienteNro: "
            + Cliente.getSiguienteNro());
    }
}
```

Imprime lo siguiente:

```
Valor de siguienteNro: 0
Valor de siguienteNro: 1
Cliente nro: 1 Apellido: Orlando Nombre: Federico
Cliente nro: 2 Apellido: Craviotto Nombre: Martin
Valor de siguienteNro: 2
```

Podemos acceder al valor de la variable de clase **siguienteNro** invocando al método **getSiguienteNro()** sin haber creado una instancia de Cliente.

En los métodos de clase NO está disponible el objeto **this**, pues no tienen ningún objeto asociado, y tampoco están disponibles las variables de instancia.

Un **método de clase** solo tiene acceso a sus variables locales, parámetros y variables de clase. **NO** puede acceder a variables ni métodos de instancia.  
Un **método de instancia** si puede acceder a métodos y variables de clase.

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
  
    int i = 2;  
  
    int f() { return -i; }  
  
    static char g() { return 'B'; }  
}
```

**Sobreescribe**   **Oculto**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);  
  
        System.out.println(b.f());  
  
        System.out.println(b.g());  
  
        System.out.println(B.g());  
  
        A a = (A) b;  
        System.out.println(a.i);  
  
        System.out.println(a.f());  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
    int i = 2;  
    int f() { return -i; }  
    static char g() { return 'B'; }  
}
```

**Sobrescribe**   **Ocultar**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());  
  
        System.out.println(b.g());  
  
        System.out.println(B.g());  
  
        A a = (A) b;  
        System.out.println(a.i);  
  
        System.out.println(a.f());  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
    int i = 2;  
    int f() { return -i; }  
    static char g() { return 'B'; }  
}
```

**Sobreescribe**   **Oculto**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());  
  
        System.out.println(B.g());  
  
        A a = (A) b;  
        System.out.println(a.i);  
  
        System.out.println(a.f());  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```



# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
    int i = 2;  
    int f() { return -i; }  
    static char g() { return 'B'; }  
}
```

**Sobreescribe**   **Oculto**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());         // B.g(); imprime B  
  
        System.out.println(B.g());  
  
        A a = (A) b;  
        System.out.println(a.i);  
  
        System.out.println(a.f());  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
  
    int i = 2;  
  
    int f() { return -i; }  
  
    static char g() { return 'B'; }  
}
```

**Sobrescribe**   **Ocultar**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());         // B.g(); imprime B  
  
        System.out.println(B.g());         // Es la mejor manera de invocar a B.g()  
  
        A a = (A) b;  
        System.out.println(a.i);  
  
        System.out.println(a.f());  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
  
    int i = 2;  
  
    int f() { return -i; }  
  
    static char g() { return 'B'; }  
}
```

**Sobrescribe**   **Ocultar**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());         // B.g(); imprime B  
  
        System.out.println(B.g());         // Es la mejor manera de invocar a B.g()  
  
  
        A a = (A) b;                       // Castea b a una instancia de la clase A  
        System.out.println(a.i);  
  
        System.out.println(a.f());  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
  
    int i = 2;  
  
    int f() { return -i; }  
  
    static char g() { return 'B'; }  
}
```

**Sobreescribe**   **Oculto**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());         // B.g(); imprime B  
  
        System.out.println(B.g());         // Es la mejor manera de invocar a B.g()  
  
  
        A a = (A) b;                       // Castea b a una instancia de la clase A  
        System.out.println(a.i);           // A.i; imprime 1  
  
        System.out.println(a.f());  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
    int i = 2;  
    int f() { return -i; }  
    static char g() { return 'B'; }  
}
```

**Sobrescribe**   **Oculto**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());         // B.g(); imprime B  
  
        System.out.println(B.g());         // Es la mejor manera de invocar a B.g()  
  
  
        A a = (A) b;                       // Castea b a una instancia de la clase A  
        System.out.println(a.i);           // A.i; imprime 1  
  
        System.out.println(a.f());         // B.f(); imprime -2. Se ejecuta el f() de B!  
  
        System.out.println(a.g());  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
  
    int i = 2;  
  
    int f() { return -i; }  
  
    static char g() { return 'B'; }  
}
```

**Sobrescribe**   **Oculto**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());         // B.g(); imprime B  
  
        System.out.println(B.g());         // Es la mejor manera de invocar a B.g()  
  
  
        A a = (A) b;                       // Castea b a una instancia de la clase A  
        System.out.println(a.i);           // A.i; imprime 1  
  
        System.out.println(a.f());         // B.f(); imprime -2. Se ejecuta el f() de B!  
  
        System.out.println(a.g());         // A.g(); imprime A  
  
        System.out.println(A.g());  
    }  
}
```

# Ocultar Métodos de Clase

Los **métodos de clase** pueden **ocultarse** por una subclase, pero **NO se sobrescriben**.

```
class A {  
    int i = 1;  
    int f() { return i; }  
    static char g() { return 'A'; }  
}  
class B extends A {  
  
    int i = 2;  
  
    int f() { return -i; }  
  
    static char g() { return 'B'; }  
}
```

**Sobrescribe**   **Ocultar**

```
public class OverrideTest {  
    public static void main(String args[]) {  
        B b = new B();  
        System.out.println(b.i);           // B.i; imprime 2  
  
        System.out.println(b.f());         // B.f(); imprime - 2  
  
        System.out.println(b.g());         // B.g(); imprime B  
  
        System.out.println(B.g());         // Es la mejor manera de invocar a B.g()  
  
  
        A a = (A) b;                       // Castea b a una instancia de la clase A  
        System.out.println(a.i);           // A.i; imprime 1  
  
        System.out.println(a.f());         // B.f(); imprime -2. Se ejecuta el f() de B!  
  
        System.out.println(a.g());         // A.g(); imprime A  
  
        System.out.println(A.g());         // Es la mejor manera de invocar a A.g()  
    }  
}
```

# Constantes

Las constantes en JAVA se definen usando la palabra clave **final**.

Una vez que un atributo declarado **final** es inicializado, no es posible cambiar su valor.

Cuando estamos definiendo atributos **final** tenemos dos formas de inicializarlas:

- en la declaración
- en el constructor

## INICIALIZACIÓN EN EL CONSTRUCTOR

```
package modelo;

public class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    private double precio=0.0;
    private final int iva;

    public Producto(String marca,String nombre,double precio){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.iva=21;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```

## INICIALIZACIÓN EN LA DECLARACIÓN

```
package modelo;

public class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,double precio){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```



# Constantes

¿Qué sucede si modificamos el valor del atributo **final** **iva**?

```
package modelo;

public class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,double precio) {}
    public int getIva() {
        return iva;
    }
    public void setIva(int iva) {
        this.iva=iva;
    }
}
```

Si queremos modificar el valor del atributo **final** **iva**, se produce un error de compilación, ya que las constantes una vez inicializadas NO pueden cambiar su valor.

¿Qué sucede si el atributo **final** en lugar de ser de tipo primitivo es un objeto?  
Vamos a averiguarlo ...

# Constantes

```
package modelo;

public class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio, Empresa productor {
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }

    public String toString(){
        return (marca + " " + nombre + " " + precio +
            " " + productor);
    }
}
```

Agregamos a la clase **Producto** una nueva constante, llamada **productor** de tipo **Empresa**

¿Hay alguna diferencia con la declaración de la constante de tipo primitivo **iva**?

**NO** hay diferencia entre declarar constantes primitivas y constantes objetos.

```
package modelo;

public class Empresa {

    public String nombre;
    private String direccion;
    public String getDireccion() {
        return direccion;
    }
    public String toString(){
        return ("Empresa: " + nombre + " "
            + direccion);
    }
    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

# Constantes

Ahora, si quisiera modificar la empresa productora del producto por otra, ¿puedo hacerlo?  
**No!!!!.**

La variable **productor** almacena una referencia a memoria de la posición donde se encuentra el objeto Empresa con el que se inicializo y dicha referencia no puede ser modificada.

Se producirá un error en compilación si lo intentamos.

```
package modelo;

public class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,double precio) {

    }

    public void setProductor(Empresa productor) {
        this.productor = productor;
    }
}
```

Observemos que si intentamos **cambiar** el valor del atributo **productor** se produce un **error de compilación**, ya que una vez inicializado NO puede ser modificada la referencia al objeto.

# Constantes

¿Puedo modificar los datos del objeto **productor**? por ej. ¿puedo cambiarle el valor a la variable de instancia **nombre**? **Si!!!!**.

Los valores de las variables de instancia del objeto **productor** pueden cambiarse.

NO podemos cambiar el valor del objeto productor, que es una referencia a una posición de memoria.

```
package modelo;

public class TestProducto {

    public static void main(String[] args) {
        Empresa e1 = new Empresa();
        e1.setNombre("Rexona");
        e1.setDireccion("50 y 115");
        Empresa e2 = new Empresa();
        e1.setNombre("Lux");
        e1.setDireccion("1 y 47");
        Producto p1 = new Producto("Lux", "jabon tocador", 2.2, e1);
        System.out.println(p1);
        p1.productor=e2;
        p1.productor.nombre="Limol";
        System.out.println(p1);
    }
}
```

# Constantes

¿Puedo modificar los datos del objeto **productor**? por ej. ¿puedo cambiarle el valor a la variable de instancia **nombre**? **Si!!!!**.

Los valores de las variables de instancia del objeto **productor** pueden cambiarse.

NO podemos cambiar el valor del objeto productor, que es una referencia a una posición de memoria.

```
package modelo;

public class TestProducto {

    public static void main(String[] args) {
        Empresa e1 = new Empresa();
        e1.setNombre("Rexona");
        e1.setDireccion("50 y 115");
        Empresa e2 = new Empresa();
        e1.setNombre("Lux");
        e1.setDireccion("1 y 47");
        Producto p1 = new Producto("Lux", "jabon tocador", 2.2, e1);
        System.out.println(p1);
        p1.productor=e2;
        p1.productor.nombre="Limol";
        System.out.println(p1);
    }
}
```

Al intentar cambiar el valor de la referencia al objeto **productor** se produce un error en compilación

# Constantes

¿Puedo modificar los datos del objeto **productor**? por ej. ¿puedo cambiarle el valor a la variable de instancia **nombre**? **Si!!!!**.

Los valores de las variables de instancia del objeto **productor** pueden cambiarse.

NO podemos cambiar el valor del objeto productor, que es una referencia a una posición de memoria.

```
package modelo;

public class TestProducto {

    public static void main(String[] args) {
        Empresa e1 = new Empresa();
        e1.setNombre("Rexona");
        e1.setDireccion("50 y 115");
        Empresa e2 = new Empresa();
        e1.setNombre("Lux");
        e1.setDireccion("1 y 47");
        Producto p1 = new Producto("Lux", "jabon tocador", 2.2, e1);
        System.out.println(p1);
        p1.productor=e2;
        p1.productor.nombre="Limol";
        System.out.println(p1);
    }
}
```

Al intentar cambiar el valor de la referencia al objeto **productor** se produce un error en compilación

Podemos modificar los valores de los datos del objeto productor.

# Constantes

¿Puedo modificar los datos del objeto **productor**? por ej. ¿puedo cambiarle el valor a la variable de instancia **nombre**? **Si!!!!**.

Los valores de las variables de instancia del objeto **productor** pueden cambiarse.

NO podemos cambiar el valor del objeto productor, que es una referencia a una posición de memoria.

```
package modelo;

public class TestProducto {

    public static void main(String[] args) {
        Empresa e1 = new Empresa();
        e1.setNombre("Rexona");
        e1.setDireccion("50 y 115");
        Empresa e2 = new Empresa();
        e1.setNombre("Lux");
        e1.setDireccion("1 y 47");
        Producto p1 = new Producto("Lux", "jabon tocador", 2.2, e1);
        System.out.println(p1);
        p1.productor=e2;
        p1.productor.nombre="Limol";
        System.out.println(p1);
    }
}
```

Al intentar cambiar el valor de la referencia al objeto **productor** se produce un error en compilación

Podemos modificar los valores de los datos del objeto productor.

Si comentamos la línea del error ¿Qué imprime TestProducto?

# Constantes

¿Puedo modificar los datos del objeto **productor**? por ej. ¿puedo cambiarle el valor a la variable de instancia **nombre**? **Si!!!!**.

Los valores de las variables de instancia del objeto **productor** pueden cambiarse.

NO podemos cambiar el valor del objeto productor, que es una referencia a una posición de memoria.

```
package modelo;

public class TestProducto {

    public static void main(String[] args) {
        Empresa e1 = new Empresa();
        e1.setNombre("Rexona");
        e1.setDireccion("50 y 115");
        Empresa e2 = new Empresa();
        e2.setNombre("Lux");
        e2.setDireccion("1 y 47");
        Producto p1 = new Producto("Lux", "jabon tocador", 2.2, e1);
        System.out.println(p1);
        p1.productor=e2;
        p1.productor.nombre="Limol";
        System.out.println(p1);
    }
}
```

Al intentar cambiar el valor de la referencia al objeto **productor** se produce un error en compilación

Podemos modificar los valores de los datos del objeto productor.

Si comentamos la línea del error ¿Qué imprime TestProducto?

```
Lux jabon tocador 2.2 Empresa: Lux 1 y 47
Lux jabon tocador 2.2 Empresa: Limol 1 y 47
```





# Constantes

## Resumen

- Dos tipos útiles:
  - Una constante en “tiempo de compilación” que jamás cambiará.
  - Un valor inicializado en “tiempo de ejecución” que no se quiere cambiar.
- En Java se identifican con la palabra clave **final**.
- Un atributo que es static y final existe independientemente de los objetos de la clase a la que pertenece y tiene un valor que no puede ser cambiado.
- Con tipos primitivos, final hace que el valor sea constante.
- Con objetos, final hace que la referencia sea constante.
  - No puede apuntar nunca a otro objeto.
  - El objeto en sí mismo (su estado y atributos) si pueden ser modificados.

# Constantes Locales

```
package modelo;

public class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public Empresa productor;
    private double precio=0.0;
    private int iva = 21;

    public Producto(String marca,String nombre,[]

    public String toString(){[]

    public final double getPrecio(int ganancia) {
        final int prec=100+ganancia;
        prec = prec + 10;
        return (precio*prec*(iva+100))/10000.00;
    }
}
```

La variable local **prec** la declaramos **final**.  
**prec** es una constante local al método  
**getPrecio()**

Si intentamos modificar la constante **prec**  
se produce un **error en compilación**

También podemos declarar argumentos de métodos como constantes. Funcionan de la misma manera que las variables locales.

La posibilidad de definir **constantes locales** en los métodos es útil ya que en la codificación de algunos algoritmos son necesarias.

# Métodos que no pueden sobrecribirse

Supongamos que redefinimos el método **getPrecio()** de la clase **Producto**, porque ahora queremos tener una ganancia superior. Pero además no queremos que las subclases de **Producto** modifiquen el cálculo del precio ¿cómo podemos hacerlo?

Definiendo el método **getPrecio()** en la clase **Producto** como **final** conseguimos que las subclases de Producto NO lo sobreescriban.

```
package modelo;

public class ProductoTocador extends Producto {

    public ProductoTocador(String marca,String nombre,
        double precio,Empresa productor){
        super (marca,nombre,precio,productor);
    }

    public final double getPrecio() {
        return (super.getPrecio()*150*(getIva()+100)=/100000.00;
    }

    public String toString(){
        return ("Producto tocado" + super.toString());
    }
}
```

# Métodos que no pueden sobrecribirse

```
package modelo;

public class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio,Empresa productor){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }

    public String toString(){
        return (marca+" "+nombre+" "+getPrecio()+
            " "+ productor);
    }

    public final double getPrecio() {
        return (precio*130*(iva+100))/10000.00;
    }
}
```

Supongamos que redefinimos el método **getPrecio()** de la clase **Producto**, porque ahora queremos tener una ganancia superior. Pero además no queremos que las subclases de **Producto** modifiquen el cálculo del precio ¿cómo podemos hacerlo?  
Definiendo el método **getPrecio()** en la clase **Producto** como **final** conseguimos que las subclases de **Producto** NO lo sobreescriban.

```
package modelo;

public class ProductoTocador extends Producto {

    public ProductoTocador(String marca,String nombre,
        double precio,Empresa productor){
        super(marca,nombre,precio,productor);
    }

    public final double getPrecio() {
        return (super.getPrecio()*150*(getIva()+100)=/100000.00;
    }

    public String toString(){
        return ("Producto tocado" + super.toString());
    }
}
```

La subclase de **Producto**, **ProductoTocador** hereda el método **getPrecio()** pero **NO** puede sobreescribirlo.

# Clases que no pueden extenderse

¿Qué pasa si declaro **final** a una clase?

```
package modelo;

public final class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio,Empresa productor){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```

# Clases que no pueden extenderse

¿Qué pasa si declaro **final** a una clase?

- Una clase **final** es una clase que no puede ser extendida, es inmutable, no cambia. No tiene capacidad de cambio.

```
package modelo;

public final class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio,Empresa productor){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```

```
package modelo;

public class ProductoTocador extends Producto {
    }
}
```

# Clases que no pueden extenderse

¿Qué pasa si declaro **final** a una clase?

- Una clase **final** es una clase que no puede ser extendida, es inmutable, no cambia. No tiene capacidad de cambio.
- Una clase **final** determina que nadie podrá heredar de dicha clase.

```
package modelo;

public final class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio,Empresa productor){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```

```
package modelo;

public class ProductoTocador extends Producto {
}

}
```

# Clases que no pueden extenderse

¿Qué pasa si declaro **final** a una clase?

- Una clase **final** es una clase que no puede ser extendida, es inmutable, no cambia. No tiene capacidad de cambio.
- Una clase **final** determina que nadie podrá heredar de dicha clase.
- Si es posible crear instancias de una clase declarada **final**

```
package modelo;

public final class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio,Empresa productor){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```

```
package modelo;

public class ProductoTocador extends Producto {
}

}
```



# Clases que no pueden extenderse

¿Qué pasa si declaro **final** a una clase?

- Una clase **final** es una clase que no puede ser extendida, es **inmutable**, no cambia. No tiene capacidad de cambio.
- Una clase **final** determina que nadie podrá heredar de dicha clase.
- Si es posible crear instancias de una clase declarada **final**
- Por alguna razón, el diseño de la clase nunca necesitará cambios o por razones de seguridad no queremos que la clase sea extendida.

Todos los **métodos** en una clase **final** son **implícitamente final**, ya que no es posible sobreescribirlos. Es posible declarar a los métodos de una **clase final**, también **final**, aunque no le estamos agregando ningún comportamiento adicional.

```
package modelo;

public final class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio,Empresa productor){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```

```
package modelo;

public class ProductoTocador extends Producto {
}

// Error: Cannot inherit from final class Producto
```

# Clases que no pueden extenderse

¿Qué pasa si declaro **final** a una clase?

- Una clase **final** es una clase que no puede ser extendida, es **inmutable**, no cambia. No tiene capacidad de cambio.
- Una clase **final** determina que nadie podrá heredar de dicha clase.
- Si es posible crear instancias de una clase declarada **final**
- Por alguna razón, el diseño de la clase nunca necesitará cambios o por razones de seguridad no queremos que la clase sea extendida.

```
package modelo;

public final class Producto {
    private static int siguienteCodigo=0;
    private int codigo;
    private String marca="Sin marca";
    private String nombre="Sin nombre";
    public final Empresa productor;
    private double precio=0.0;
    private final int iva = 21;

    public Producto(String marca,String nombre,
        double precio,Empresa productor){
        this.marca=marca;
        this.precio=precio;
        this.nombre=nombre;
        this.productor=productor;
        siguienteCodigo++;
        this.codigo=siguienteCodigo;
    }
}
```

```
package modelo;

public class ProductoTocador extends Producto {
}

}
```

# Clases que no pueden extenderse de la API Java

Supongamos que deseamos crear mi propia clase String extendiendo la clase String de la API JAVA:

```
package ej3;  
  
public class MiString extends String {  
  
}
```

Se produce un error en compilación. Esto se debe a que String es una clase final y por lo tanto no puede ser extendida.

java.lang  
**Class String**

[java.lang.Object](#)  
**java.lang.String**

All Implemented Interfaces:

[Serializable](#), [CharSequence](#), [Comparable<String>](#)

```
public final class String  
extends Object  
implements Serializable, Comparable<String>, CharSequence
```

The String class represents character strings. All string literals in Java programs

Strings are constant; their values cannot be changed after they are created. Strings are shared. For example:

Otras clases **final** de la API JAVA son: Boolean, Byte, Character, Double, Float, Integer, Long, Short, StringBuffer, System, etc.



# Resumiendo...

## Los métodos static...

- Deben ser invocados utilizando el nombre de la clase en lugar de un objeto.
- Pueden ser invocados sin que existan instancias de dicha clase.
- Son utilizados cuando la funcionalidad no depende (ni dependerá) del valor particular de una instancia.
- No se encuentran asociados a una instancia particular de modo que no puede acceder a los atributos de instancia.
- No pueden acceder a métodos no static dado que los métodos no static se encuentran usualmente asociados con el estado de atributos de instancia.
- Pueden acceder a los atributos estáticos.



# Resumiendo...

## La palabra clave final...

- Para crear una constante en Java, la variable debe ser declarada como `static final`.
- El valor de una variable `final` no puede ser cambiado una vez que es asignado.
- El valor a una variable `final` debe ser asignado al momento de la declaración o en el constructor.
- Un método `final` no puede ser sobre-escrito.
- Una clase `final` no puede ser extendida.



# Referencias

- **Fundamentos de Java (clases y métodos finales)**
  - <http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/herencia/final.htm>
- **Pensando en Java por Bruce Eckel, 4ra Edición**
  - Capítulo 6, Reutilizando clases – La palabra clave final
- **Pensando en Java por Bruce Eckel, 4ra Edición**
  - Capítulo 2, Todo es un objeto – Armandando un programa Java – La palabra clave static



Ministerio de Producción  
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



**PROGRAMACIÓN ORIENTADA  
A OBJETOS**