

# Uso de JavaDoc

Los comentarios, anotaciones en el código que el compilador ignora pero son útiles para los programadores, existen en los lenguajes de programación desde el principio. Desde hace mucho tiempo se observó que en realidad los comentarios se usaban para dos propósitos diferentes:

- 1.-Para explicar el propósito de sentencias o grupos de sentencias. Estos comentarios son útiles para el propio autor del código, y para otros que quieran modificar ese código.
- 2.-Comentarios explicando qué hace una "pieza" cerrada de código. Estos comentarios son útiles para quien quiere utilizar esta "pieza" en su propio programa, y que por tanto está necesita saber lo qué hace, no cómo se las ha arreglado el programador para conseguir este resultado.

# Diseñando en Java

## -Posibilidades

- comentarios "internos" se usan los caracteres // seguidos del comentario, o /\* .... \*/

Ejemplo

```
-int suma = 0; // al principio la suma vale 0
// vamos sumando cada uno de los cuadrados entre 1 y 100 y
//acumulando el resultado
for (int i=1; i<=99; i+=2) suma += i;
// finalmente mostramos el resultado por pantalla
System.out.println(suma);
```

# Posibilidades

## -Segundo tipo

- Comentarios JavaDoc
- Se escriben comenzando por `/**` y terminando con `*/`, pudiendo ocupar varias lineas.
- Siguen un formato prefijado.

# Formato de los comentarios JavaDoc

- Describir, principalmente, clases y métodos.
- Formato común
- Indicadores especiales, que comienzan siempre por '@'
- Por ejemplo.
- `package figuras;`
- `/**`
- `* Una clase para representar círculos`
- `* Cada círculo queda determinado por su radio junto con las`
- `* coordenadas de su centro.`
- `* @version 1.3, 27/13/03`
- `* @author Andrés Ulloa */`

```
public class Círculo {  
    protected double x,y; // coordenadas del centro  
    protected double r; // radio del círculo  
    /**  
     *Crea un círculo a partir de su origen su radio.  
     *@param x La coordenada x del centro del círculo.  
     *@param y La coordenada y del centro del círculo.  
     *@param r El radio del círculo. Debe ser mayor o igual a  
     *o. */
```

```
public Circulo(double x, double y, double r) { this.x=x;  
    this.y = y; this.r = r;  
}
```

```
/**
```

- Cálculo del área de este círculo.
- \* @return El área (mayor o igual que 0) del círculo.
- \*/
- ```
public double área() {  
    return Math.PI*r*r;  
}
```

```
/**
```

Indica si un punto está dentro del círculo.

@param px componente x del punto \*

@param py componente y del punto \*

@return true si el punto está dentro del círculo o false en otro caso.

```
*/
```

```
public boolean contiene(double px, double py) {
```

```
    /* Calculamos la distancia de (px,py) al centro del círculo  
    (x,y), que se obtiene como raíz cuadrada de  $(px-x)^2 + (py-y)^2$  */
```

```
    double d = Math.sqrt((px-x)*(px-x)+(py-y)*(py-y));
```

```
// el círculo contiene el punto si d es menor o igual al radio
```

```
return d <= r;
```

```
}
```

```
}
```



# Recalcar

- la descripción de la clase o del método no va precedida de ningún indicador
- Se usan indicadores para el número de versión (@version), el autor (@author)(no son obligatorios), por ejemplo en un método sin parámetros no se incluye obviamente el indicador @param(pueden existir mas de un indicador en un mismo comentario).

- **@author nombreDelAutor**

Indica quién escribió el código al que se refiere el comentario. Si son varias personas se escriben los nombres separados por comas o se repite el indicador, según se prefiera. *Es normal incluir este indicador en el comentario de la clase y no repetirlo para cada método*, a no ser que algún método haya sido escrito por otra persona.

**@version númeroVersión**

Si se quiere indicar la versión. *Normalmente se usa para clases, pero en ocasiones también para métodos.*

**@param nombreParámetro**

Para describir un parámetro de un método.

- **@return**

Describe el valor de salida de un método.

- **@see *nombre***

Cuando el trozo de código comentado se encuentra relacionada con otra clase o método, cuyo nombre se indica en *nombre*.

- **@throws *nombreClaseExcepción***

Cuando un método puede lanzar una excepción ("romperse" si se da alguna circunstancia) se indica así.

- **@deprecated**

Indica que el método (es más raro encontrarlos para una clase) ya no se usa y se ha sustituido por otro.


- **Observación** : JavaDoc es mucho más completo de lo que hemos descrito en este primer vistazo. Por ejemplo, además de para clases y métodos, también existen comentarios JavaDoc para paquetes.

# Javadoc parte de JDSK

- Pueden utilizarse para generar la documentación de los programas
- El formato más sencillo de esta herramienta, cuando se emplea desde línea de comandos es:
- `javadoc fichero.java`

## Ejemplo

`Javadoc Circulo.java`

- 
- Lo que hace esta herramienta es extraer los comentarios JavaDoc contenidos en el programa Java indicado y construir con ellos ficheros .html que puede servir como documentación de la clase.

figuras

## Class Círculo

```
java.lang.Object
└─ figuras.Círculo
```

```
public class Círculo
extends java.lang.Object
```

Una clase para representar círculos situados sobre el plano. Cada círculo queda determinado por su radio junto con las coordenadas de su centro.

### Constructor Summary

[Círculo](#)(double x, double y, double r)  
Crea un círculo a partir de su origen su radio.

### Method Summary

|         |                                                                                                |
|---------|------------------------------------------------------------------------------------------------|
| double  | <a href="#">área</a> ()<br>Cálculo del área de este círculo.                                   |
| boolean | <a href="#">contiene</a> (double px, double py)<br>Indica si un punto está dentro del círculo. |

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



## Constructor Detail

### Círculo

```
public Círculo(double x,  
              double y,  
              double r)
```

Crea un círculo a partir de su origen su radio.

**Parameters:**

- x - La coordenada x del centro del círculo.
- y - La coordenada y del centro del círculo.
- r - El radio del círculo. Debe ser mayor o igual a 0.

## Method Detail

### área

```
public double área()
```

Cálculo del área de este círculo.

**Returns:**

El área (mayor o igual que 0) del círculo.

### contiene

```
public boolean contiene(double px,  
                       double py)
```

Indica si un punto está dentro del círculo.

**Parameters:**

- px - componente x del punto
- py - componente y del punto

**Returns:**

true si el punto está dentro del círculo o false en otro caso.



- Como se ve, es un fichero muy completo, y con un aspecto agradable, generado sin apenas esfuerzo. En caso de que nuestra aplicación contenga varias clases la herramienta se encarga de generar ficheros índice y de establecer los enlaces entre todas ellas. La herramienta es, por tanto fundamental en el desarrollo en Java.
- **Aviso:** *Un programa mal documentado es, básicamente, un programa inútil.*
- Esto es así porque los programas no son nunca productos cerrados, acabados; siempre se encuentran errores que corregir, posible ampliaciones.

- Y el código que nos parece evidente un día se convierte en críptico e inmanejable una semana después. La herramienta javadoc ayuda a mantener la documentación del código en un formato legible y práctico.
- *Si tenemos instalado el compilador Eclipse la herramienta javadoc está directamente accesible.*