Centro Asociado Palma de Mallorca

Introducción Práctica de Programación Java

Antonio Rivero Cuesta

Sesión IV

Documentación del Código

Las ***buenas prácticas de programación*** son técnicas que le ayudarán a producir programas más claros, comprensibles y fáciles de mantener.

Ciertas organizaciones requieren que todos los programas comiencen con un comentario que explique su propósito, el autor, la fecha y la hora de la última modificación del mismo.

Es un complemento importante para obtener código de buena calidad.

Permite al programador comunicar sus intenciones a los lectores humanos en un lenguaje natural de alto nivel.

En lugar de forzarlos a leer código de nivel relativamente bajo.

Los programadores pueden usarla sin tener que conocer los detalles de su implementación.

Comentarios

En un programa Java hay tres tipos de comentarios.

- Comentario de línea.
- Comentario de multilínea.
- Comentario de documentación, Javadoc.

Comentario de Línea

Empieza con //

Comienza con estos caracteres y termina al final de la línea.

También lo llamamos **comentario al final de una línea.**

```
// Programa para imprimir texto.
```

Comentario de Multilínea

Empieza por /*.

Y termina por */.

```
/* Aquí empieza el bloque comentado
y aquí acaba */
```

El compilador ignora todo el texto contenido dentro del comentario.

Comentario de Documentación

Empieza por /**.

Y termina por */.

Java dispone de la herramienta Javadoc para documentar automáticamente los programas.

En un comentario de documentación normalmente se indica el autor y la versión del software.

En Java la documentación del código es fundamental.

Resulta evidente en cuanto se consulta la documentación de las clases que se proporciona directamente en los paquetes que distribuye Sun.

También podremos hacer lo mismo nosotros.

Java promueve que el programador documente el funcionamiento de las clases dentro del propio código.

Los javadoc generan un conjunto de páginas en HTML por las que se puede navegar utilizando cualquier navegador Web.

```java
/**
* Programa HolaMundo
* @author Antonio Rivero
* @version 1.00 2015/7/9
*/
public class HolaMundo{
  public static void main(String[] args){
    System.out.println("Hola mundo");
  } //Cierre del main
} //Cierre de la clase
```

Se debe poner un comentario:

- Al principio de cada archivo con la descripción y objetivo del mismo.
- Antes de cada método, explicando para qué sirve.
- Antes de cada algoritmo, explicando qué hace el algoritmo.
- Antes de cada definición de estructura de datos, indicando cuál es su objetivo.
- Antes de cada parte significativa del programa.

El uso de comentarios hace más claro y legible un programa.

En los comentarios se debe decir qué se hace, para qué y cuál es el fin de nuestro programa.

Conviene utilizar comentarios siempre que merezca la pena hacer una aclaración sobre el programa.

Líneas en Blanco y Espacios

Los Utilizamos para mejorar la legibilidad del programa.

Los comentarios deben ir precedidos por una línea en blanco para separarlos lógicamente de la parte precedente del programa.

Etiquetas

Java complementa la tarea de documentación utilizando ciertas etiquetas en los comentarios.

Orden de las Etiquetas

@author: En clases e interfaces. Se pueden poner varios. En este caso resulta apropiado hacerlo en orden cronológico.

@version: En clases y en interfaces.

@param: En métodos y constructores. Se ponen tantos como argumentos tenga el constructor o el método. Deberían aparecer en el mismo orden en que se declaran.

@return: En métodos.

@exception: En constructores y métodos. Deberían aparecer en el mismo orden en que se declaran o en orden alfabético.

@throws: Con Javadoc 1.2 es un sinónimo de @exception.

@see: Se pueden poner varios. Se recomienda empezar por los más generales e ir indicando después los más concretos.

@since. En clases. Versión desde la que está presente la clase.

@deprecated. En métodos. Indicación de que el método es obsoleto.

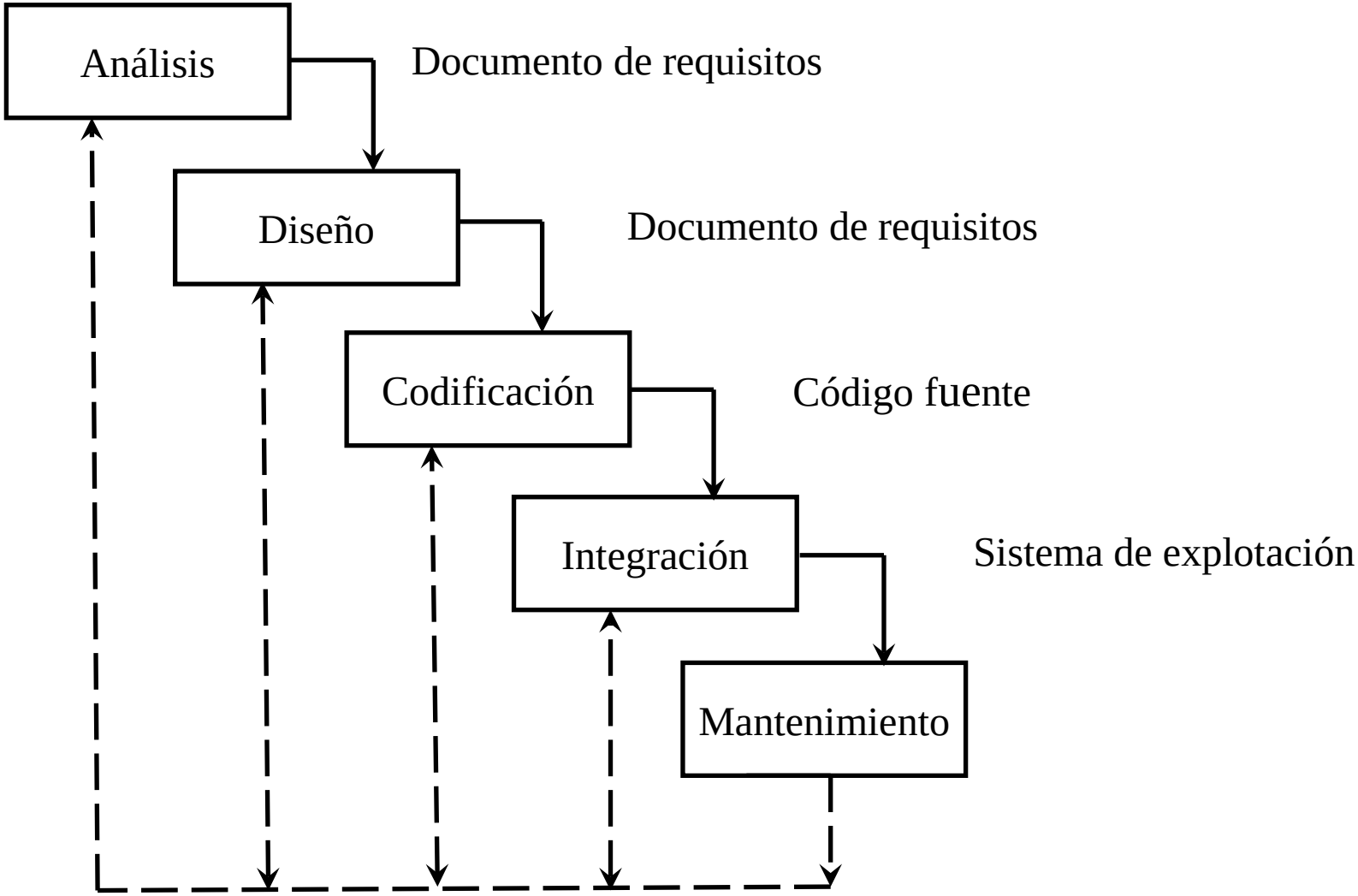
Convenios de Programación en Java

Tener unas reglas para programar es importante porque:

- La mayor parte del tiempo se dedica a leer programas, por lo que resulta importante que sean fáciles de leer y de entender.
- Seguir unas reglas de programación permite mejorar la legibilidad del código escrito y entender fácilmente el código que no es de uno mismo.

- Si se da código propio a terceras personas, o se intenta explicar lo que se ha hecho, será más fácil de entender si se ha seguido un estilo consistente y similar, o igual, al de otros programadores.
- Casi nunca ocurre que todo el código que utiliza para un programa sea escrito por un solo programador, sino que se habrá desarrollado por un grupo de programadores.

Ciclo de Vida en Cascada



Acoplamiento y Cohesión

Hay dos términos que son fundamentales cuando hablamos sobre la calidad de un diseño de clases:

- Acoplamiento.
- Cohesión.

Acoplamiento

Describe la interconexión de las clases.

Debemos lograr:

- Un acoplamiento débil o bajo.
- Que cada clase sea altamente independiente.
- Se comunique con otras clases mediante una interfaz compacta y bien definida.

El grado de acoplamiento determina el grado de dificultad de realizar modificaciones en una aplicación.

En una estructura de clases fuertemente acopladas, un cambio en una clase hace necesario también cambiar otras varias clases.

Este hecho es el que tratamos de evitar porque el efecto de hacer un pequeño cambio puede rápidamente propagarse a la aplicación completa.

Además, encontrar todos los lugares en que resulta necesario hacer los cambios y realmente llevar a cabo estos cambios puede ser dificultoso y consumir demasiado tiempo.

Por otro lado, en un ***sistema débilmente acoplado***, podemos con frecuencia modificar una clase sin tener que realizar cambios en ninguna otra y la aplicación continúa funcionando.

Cohesión

Describe cuánto se ajusta una ***unidad de código*** a una tarea lógica concreta.

En un sistema altamente cohesivo cada ***unidad de código*** es responsable de una tarea bien definida.

Un diseño de clases de buena calidad exhibe un alto grado de cohesión.

La razón principal que subyace al principio de cohesión es la ***reutilización***.

Si un método de una clase es responsable de una única cosa bien definida es más probable que pueda ser usado nuevamente en un contexto diferente.

Una ventaja complementaria es que cuando se requiere un cambio de un aspecto de una aplicación, probablemente encontremos todas las piezas de código relevantes ubicadas en la misma unidad.