

ENTORNOS DE DESARROLLO

IES Santiago Hernández
Curso 2017-2018
Ignacio Agudo Sancho

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

Documentación

- ⦿ La documentación es el texto escrito que acompaña a los proyectos de cualquier tipo.
- ⦿ Es un requisito importante en un proyecto comercial: el cliente siempre va a solicitar que se documenten las distintas fases del proyecto.

Documentación

⦿ Tipos de documentación:

- De las especificaciones
- Del diseño
- Del código fuente
- De usuario final

Documentación. Tipos

- ⦿ Documentación de las especificaciones.
 - Sirve para asegurar que tanto el desarrollador como el cliente tienen la misma idea sobre las funcionalidades del sistema.
 - Tiene que quedar claro, o el desarrollo podrá no ser aceptable.
 - La norma IEEE 830 recoge recomendaciones para la documentación de requerimientos de software, e indica que las especificaciones deben describir la siguiente documentación:

Documentación. Tipos

⦿ Documentación de las especificaciones.

- IEE 830 - documentación:

- Introducción: define los fines y objetivos del sw.
- Descripción de la información: detalle del problema, incluyendo hardware y software necesario.
- Descripción funcional: cada función requerida en el sistema, incluyendo diagramas.
- Descripción del comportamiento: cómo se comportará el sw ante problemas externos y controles internos.
- Criterios de validación: límites de rendimiento, clases de pruebas, respuesta esperada del sw y consideraciones especiales.

Documentación. Tipos

⦿ Documentación del diseño

- En la fase de diseño se decide la estructura de datos a utilizar, la forma en la que se van a implementar las distintas estructuras, el contenido de las clases, sus métodos y sus atributos, los objetos a utilizar, las funciones, sus datos de entrada y salida, qué tareas realizan...
- Todo eso debe quedar reflejado en la documentación del diseño.

Documentación. Tipos

- ⦿ Documentación del código fuente
 - Durante la implementación.
 - Comentar cada una de las partes que tiene un programa.
 - Los comentarios se incluyen en el código fuente para clarificar y explicar cada elemento del programa.
 - Se deben comentar las clases, los métodos, las variables y todo aquel elemento que se considere importante.

Documentación. Tipos

⦿ Documentación de usuario final

- Se entrega al usuario tanto especializado como no especializado.
- En ella se describirá cómo utilizar las aplicaciones del proyecto.
- Puede ser redactado por cualquier persona, no necesariamente implicada en el desarrollo del proyecto.

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

Documentación del Código Fuente

- ⦿ Documentar los programas es necesario para explicar claramente lo que hace el programa.
- ⦿ Todo el equipo sabrá qué se está haciendo y por qué.
- ⦿ Mucho más fácil de mantener (reparar errores o añadir funcionalidades)

Documentación del Código Fuente

- ⦿ Dos reglas que no se deben olvidar nunca:
 - Todos los programas tienen errores y descubrirlos es solo cuestión de tiempo y de que el programa tenga éxito y se utilice frecuentemente.
 - Todos los programas sufren modificaciones a lo largo de su vida, al menos todos aquellos que tienen éxito.

Documentación del Código Fuente

- ⦿ Las modificaciones no siempre serán realizadas por el autor del programa, por eso es necesario tener bien documentado el código.
- ⦿ Interesa que se explique qué hace una clase o un método y por qué y para qué se hace, cuál es el uso esperado de una variable o esos métodos.
- ⦿ Incluso qué se debería revisar y modificar si hubiese tiempo para ello.

Documentación del Código Fuente

- ⦿ Para documentar proyectos existen muchas herramientas, normalmente cada lenguaje dispone de su propia herramienta, como:
 - PHPDoc
 - phpDocumentor
 - JavaDoc
 - JSDoc...
- ⦿ Nos centraremos en JavaDoc en Eclipse.

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

JavaDoc

- ⦿ Es la utilidad de Java para extraer y generar documentación directamente del código en formato HTML.
- ⦿ Para que la documentación sea útil debemos escribir los comentarios del código de acuerdo a las recomendaciones de Javadoc.
- ⦿ La documentación y el código se van a incluir dentro del mismo fichero.

JavaDoc

- ⦿ Recomendaciones sobre los comentarios y la documentación del código fuente:
 - Comentarios de línea: comienzan con “//” y terminan con la línea.
 - Comentarios de bloque (o tipo C): comienzan con “/*” y terminan con “*/”. Pueden agrupar varias líneas.

JavaDoc

- Recomendaciones sobre los comentarios y la documentación del código fuente:
 - Comentarios de documentación Javadoc:
 - Se colocan entre los delimitadores `/** ... */`
 - Agrupan varias líneas que irán precedidas de `“*”`
 - Deben colocarse ANTES de la declaración de una clase, un campo, un método o un constructor.
 - Dentro de los delimitadores se podrá escribir etiquetas HTML.
 - Los comentarios Javadoc están formados por dos partes: una descripción y un bloque de etiquetas.

JavaDoc

● Ejemplo de Javadoc:

- /**
- * <h2> Esto es un ejemplo de documentación </h2>
- * Puedo escribir etiquetas HTML, para mejorar
- * la visualización.
- * @author NachoAgudo 2017
- * @version v1.0
- */

JavaDoc

- Se pueden usar etiquetas o tags para documentar ciertos aspectos concretos como la versión de la clase, el autor, los parámetros utilizados o los valores devueltos.
- Las etiquetas Javadoc van precedidas por @
- A continuación veremos las etiquetas más utilizadas:

JavaDoc

- ◉ Etiquetas Javadoc más utilizadas:
 - `@author`: autor de la clase. Solo para clases.
 - `@version`: versión de la clase. Solo para clases.
 - `@see`: Referencia a otra clase, ya sea del API, del mismo proyecto o de otro. (`@see` cadena)
 - `@param`: descripción de parámetro. Una etiqueta por parámetro.
 - `@return`: descripción de lo que devuelve. Solo si no es void.
 - `@throws`: descripción de la excepción que pueda lanzar. Una etiqueta por excepción.
 - `@deprecated`: marca el método como obsoleto. Solo se mantiene por compatibilidad.
 - `@since`: num de versión desde la que existe el módulo.

JavaDoc

- ⦿ La mayor parte de los entornos de desarrollo incluyen un botón o un enlace para configurar y ejecutar el Javadoc.
- ⦿ En Eclipse: Menú Project → Generate Javadoc

JavaDoc

- ⦿ Al ejecutar el generador de Javadoc aparecerá una ventana donde nos pedirá:
 - Javadoc command: dónde se encuentra el javadoc.exe (dentro de la carpeta bin del JDK)
 - Proyecto y clases a documentar.
 - Visibilidad de los elementos que se van a documentar (con private se documentarán públicos, privados y protegidos).
 - Carpeta destino donde se almacenará el HTML generado.



JavaDoc

Generate Javadoc







Javadoc Generation

Select types for Javadoc generation.

Javadoc command:

/Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/Contents/Home/bin/javadoc  

Select types for which Javadoc will be generated:


<input checked="" type="checkbox"/>		CalculadoraTest
<input type="checkbox"/>		HiloJoin
<input type="checkbox"/>		HolaMundo
<input type="checkbox"/>		NumerosLed
<input type="checkbox"/>		PruebaGUI
<input type="checkbox"/>		PruebaWB

Create Javadoc for members with visibility:

☐ Private ☐ Package ☐ Protected ☒ Public

Public: Generate Javadoc for public classes and members.


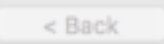
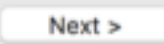
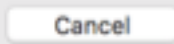
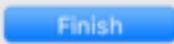
☒ Use standard doclet

Destination: 

☐ Use custom doclet

Doclet name:

Doclet class path:

JavaDoc

- ⦿ Si pulsamos en Next, tendremos que indicar:
 - Título del HTML que se va a generar
 - Opciones para la generación de páginas HTML
 - Como mínimo se selecciona la barra de navegación y el índice.

JavaDoc

☒ Document title:








Basic Options

- ☒ Generate use page
- ☒ Generate hierarchy tree
- ☒ Generate navigator bar
- ☒ Generate index
 - ☒ Separate index per letter

Document these tags

- ☒ @author
- ☒ @version
- ☒ @deprecated
 - ☒ deprecated list

Select referenced archives and projects to which links should be generated:

- ☐  CalculadoraTest - not configured
- ☐  charsets.jar - <https://docs.oracle.com/javase/8/docs/api/>
- ☐  cldrdata.jar - not configured
- ☐  dnsns.jar - not configured
- ☐  jaccess.jar - not configured
- ☐  jce.jar - <https://docs.oracle.com/javase/8/docs/api/>
- ☐  jfr.jar - <https://docs.oracle.com/javase/8/docs/api/>

Select All

Clear All

Browse...

JavaDoc

● Actividad:

- Observar los ficheros en la carpeta PruebaDoc y generar el Javadoc.
- Realizar pruebas de documentación en los proyectos de las actividades realizadas en el módulo de programación:
 - Añadir autor y versión a las clases.
 - Añadir descripciones a las clases, los métodos y sus parámetros utilizando etiquetas HTML.
 - Generar la documentación cambiando las distintas opciones.

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

Refactorización

- ⦿ Es una técnica que permite la optimización de un código por medio de cambios en su estructura interna sin que esto suponga alteraciones en su comportamiento externo.
- ⦿ No arregla errores ni añade funcionalidades, sino que mejora la calidad del código para facilitar nuevos desarrollos, corrección de errores o adición de funcionalidades.

Refactorización

- ⦿ El objetivo es limpiar el código para que sea más fácil de entender qué se está haciendo y modificar lo que se necesite.
- ⦿ ¿Qué hace la refactorización?
 - Limpia el código, mejorando la consistencia y claridad.
 - Mantiene el código (no corrige errores ni añade funcionalidades)
 - Va a permitir facilitar la realización de cambios en el código.
 - Se obtiene un código limpio y altamente modularizado

Refactorización

- ⦿ ¿Cuándo refactorizar? – Bad smells
- ⦿ Se debe refactorizar mientras se va realizando el desarrollo de la aplicación.
- ⦿ A continuación se describen los síntomas de una necesidad de refactorización (malos olores)

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

Refactorización – Bad smells (1/6)

- ⦿ Código duplicado: la principal razón. Hay que extraerlo y unificarlo.
- ⦿ Métodos muy largos: cuanto más largo, más difícil de comprender. Si es muy largo, posiblemente esté haciendo tareas que deberían ser responsabilidad de otros. Hay que descomponer el método en otros más pequeños. En POO cuanto más corto es un método, más fácil es reutilizarlo.

Refactorización – Bad smells (2/6)

- Clases muy grandes: si una clase intenta resolver muchos problemas, será muy grande, contendrá muchos métodos, atributos e instancias. La clase está asumiendo muchas responsabilidades. Hay que hacer clases más pequeñas y delimitar conjuntos de responsabilidades.

Refactorización – Bad smells (3/6)

- Lista de parámetros extensa: en POO no se suelen pasar muchos parámetros a los métodos, sino solamente los necesarios para que el objeto involucrado consiga lo necesario. Demasiados parámetros puede significar un problema de encapsulación de datos o la necesidad de crear una clase de objetos a partir de esos parámetros, y pasar el objeto como argumento en vez de todos parámetros. Especialmente si están relacionados y suelen ir siempre juntos.

Refactorización – Bad smells (4/6)

- Cambio divergente: una clase que se modifica con frecuencia por motivos que no están relacionados entre sí, lo mejor es eliminar la clase.
- Cirugía a tiro de pistola: se presenta cuando después de un cambio en una clase, se deben realizar varias modificaciones adicionales en diversos lugares para compatibilizar ese cambio. Hay que mover los comportamientos similares a una sola clase, así cada una tendrá su propia responsabilidad.

Refactorización – Bad smells (5/6)

- ⦿ Envidia de funcionalidad: un método utiliza más elementos de otra clase que de la suya propia. Hay que pasar el método a la clase cuyos elementos utiliza más.
- ⦿ Clase de solo datos: clases que solo tienen atributos y métodos de acceso a ellos (“get” y “set”). Hay que cuestionarse si realmente las necesitamos o no ya que no suelen tener comportamiento alguno.

Refactorización – Bad smells (6/6)

- Legado rechazado: subclases que utilizan solo unas pocas características de sus superclases. Suele indicar que no es correcta la jerarquía de clases tal y como fue pensada. Hay que extraer una superclase para obtener las variables y el comportamiento que necesitamos.

Refactorización – Ventajas

- ⦿ Mantenimiento del diseño del sistema
- ⦿ Incremento de facilidad de lectura
- ⦿ Comprensión del código fuente
- ⦿ Detección temprana de fallos
- ⦿ Aumento de la velocidad de programación

Refactorización – Inconvenientes

- ⦿ Las bases de datos
- ⦿ Los cambios de interfaces

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

Refactorización en Eclipse

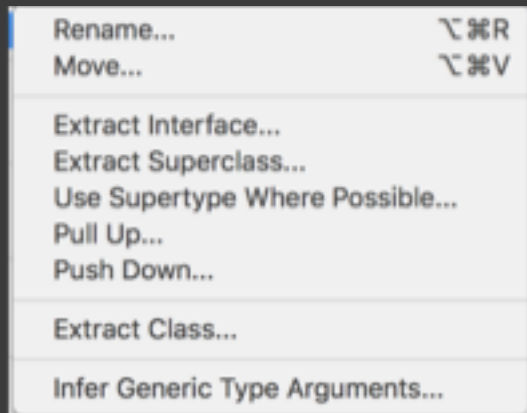
- ⦿ Podemos invocar la refactorización en:
 - Una clase
 - Un método
 - Un atributo
- ⦿ Según dónde la invoquemos, tendremos unas opciones u otras.

Refactorización en Eclipse

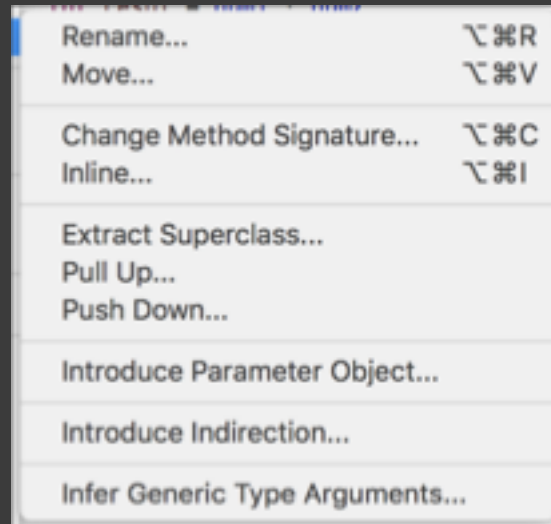
- ⦿ Podemos invocar la refactorización en:
 - Una clase
 - Un método
 - Un atributo
- ⦿ Según dónde la invoquemos, tendremos unas opciones u otras.

Refactorización en Eclipse

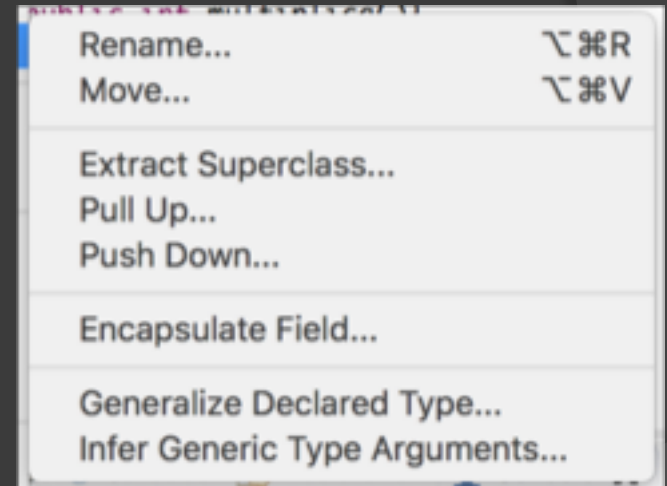
CLASE



MÉTODO



ATRIBUTO



Refactorización en Eclipse

- Los métodos de refactorización son las prácticas para refactorizar el código.
- Podremos plantear casos para refactorizar y se mostrarán las posibles soluciones en las que podremos ver el antes y el después.
- Para refactorizar, se selecciona el elemento, se pulsa el botón derecho del ratón, se selecciona Refactor y a continuación el método de refactorización.
- A continuación vemos los métodos más comunes.

Refactorización en Eclipse - Métodos

- Rename: cambiar el nombre del elemento. Se modifican las referencias a ese elemento.
- Move: mueve una clase de un paquete a otro y cambia las referencias. También se puede hacer arrastrando la clase a otro paquete.
- Extract constant: convierte un número o cadena en una constante. Mostrará dónde se van a hacer los cambios. Todos los usos del literal se sustituirán por la constante. Sirve para modificar el valor en un único lugar.

Refactorización en Eclipse - Métodos

- Extract Local Variable: asignar una expresión a variable local. Cualquier referencia a la expresión se sustituye por la variable. La misma expresión en otro método NO se modifica. Mostrará dónde se van a realizar los cambios, el antes y el después.
- Convert Local Variable to Field: convierte una variable local en un atributo privado de la clase. Todos los usos de la variable local se sustituyen por ese atributo.

Refactorización en Eclipse - Métodos

- Extract Method: convierte un bloque de código en un método. El bloque no debe dejar llaves abiertas. Eclipse ajustará automáticamente los parámetros y el retorno de la función. Hay que indicar si el método será público, protegido o privado.
- Change Method Signature: cambiar el nombre de un método y los parámetros que tiene. Se actualizarán todas las dependencias y llamadas al método dentro del proyecto.

Refactorización en Eclipse - Métodos

- Inline: permite ajustar una referencia a una variable o método con la línea en la que se utiliza y conseguir así una única línea de código.

- Antes:

```
File fichero = new File("nombres.dat");  
RandomAccessFile file;  
file = new RandomAccessFile(fichero, "rw");
```

- Después:

```
RandomAccessFile file;  
file = new RandomAccessFile(new File("nombres.dat"), "rw");
```

Refactorización en Eclipse - Métodos

- Member Type to Top Level: convierte una clase anidada en una clase de nivel superior con su propio archivo de java. Si la clase es estática, la refactorización es inmediata. Si no es estática, no pide un nombre para declarar el nombre de la clase que mantendrá la referencia con la clase inicial.

Refactorización en Eclipse - Métodos

- Extract Interface: permite escoger los métodos de una clase para crear una Interface.
 - NOTA: Interfaz es una especie de plantilla que define los métodos acerca de lo que puede o no hacer una clase, pero no los desarrolla. Son las clases que implementan esa interfaz quienes desarrollarán los métodos.
 - Por ejemplo: Interfaz animal con los métodos vacíos comer y respirar. Cada animal come y respira diferente, por lo que serán las clases de animal quienes implementen cómo comen y respiran.

Refactorización en Eclipse - Métodos

- Ejemplo interfaz:
- Clase Interface Clase que implementa la interface

```
interface Animal{  
    void comer();  
    void respirar();  
}
```

```
class Perro implements Animal{  
  
    public void comer(){  
        //se define cómo come el perro  
    }  
  
    public void respirar(){  
        //se define cómo respira el perro  
    }  
  
    public void ladrar(){  
        //se define cómo ladra el perro  
    }  
  
}
```

Refactorización en Eclipse - Métodos

- Extract Superclass: permite extraer una superclase. Si la clase ya usaba una superclase, la recién creada pasará a ser su superclase. Se pueden seleccionar los métodos y atributos que formarán parte de la superclase. Puede haber fallos de compilación si en los métodos que pasemos a la superclase hay referencias a campos de la clase original.

5. Documentación y optimización

⦿ Documentación.

- Documentación del código fuente
- JavaDoc

⦿ Refactorización.

- Bad smells
- Refactorizar en Eclipse

Refactorización en Eclipse - Actividades

- 3.1: En la clase `VentanaDepart.java` del proyecto `FicheroAleatorioVentana` que se encuentra en la carpeta de recursos, realiza los siguientes cambios:
 - Extrae una variable local llamada `existedepart` de la cadena "DEPARTAMENTO EXISTE."
 - Extrae una constante local llamada `NOEXISTEDEPART` de la cadena "DEPARTAMENTO NO EXISTE."
 - Convierte la variable local creada en un atributo de la clase.
 - Extrae una variable local llamada `depar_error` de la cadena "DEPARTAMENTO ERRÓNEO" y conviértela en un atributo de la clase.

Refactorización en Eclipse - Actividades

- 3.2: Extraer métodos dentro de la clase `VentanaDepart.java`
 - Dentro del método `public void actionPerformed(ActionEvent e)`, extrae varios métodos, uno para cada una de estas operaciones:
 - Insertar departamento
 - Consultar
 - Borrar
 - Modificar
 - Extraer los métodos de las instrucciones que van dentro de los distintos `if(e.getSource())`, que preguntan por `balta`, `consu`, `borra` y `modif`, llamarlos `altadepart`, `consuldepart`, `borradepart` y `modifdepart`.

Refactorización en Eclipse - Actividades

◎ 3.3:

- Cambia el nombre y los parámetros de los métodos creados anteriormente.
- Añadir un parámetro de tipo String, valor por defecto "PRUEBA"
- Cambiar el tipo de dato devuelto en los métodos para que devuelvan un entero.
- Comprobar y corregir los errores de retorno.

Refactorización en Eclipse - Actividades

● 3.4:

- Crea la interfaz `InterfaceVentanaDepart` que contenga los métodos creados en la actividad 3.2. Al crear la interfaz solamente se pueden añadir los métodos públicos, así que haz los cambios que se necesiten para crearla.

Refactorización en Eclipse - Actividades

● 3.5:

- Crea la superclase SuperclaseDepart a partir de la clase VentanaDepart, que incluya solo los métodos de grabar y visualizar.
- Observa que la declaración de la clase ventana ahora es extends de SuperclaseDepart.
- Soluciona los errores generados (referencias a campos de la clase original y definición de objetos de la clase anidada con el parámetro this).