Documentación, refactorización y pruebas

Desarrollo de aplicaciones Multiplataforma

&

Desarrollo de aplicaciones Web

Documentación, refactorización y pruebas

Actividad

Realizar casos de pruebas unitarias en Java con JUnit de eclipse y a partir de un código realizar el grafo, la complejidad ciclomática y obtener los caminos independientes.

Aplicar la refactorización a un código y documentar un código en Java con JavaDoc en eclipse

**Objetivos**

Diseñar casos de prueba con Junit y Eclipse.

Dado un código, realizar el grafo, la complejidad ciclomática y obtener los caminos independientes.

* Aplicar patrones de refactorización para optimizar el código Java.
* Realizar la documentación de una clase y generar un JavaDoc con Eclipse.

|  |
| --- |
| **¿Cómo lo hago?** |
| 1. Rellena los datos que se piden en la tabla “Antes de empezar”. 2. Haz uso de fuentes comunes como Arial, Calibri, Times New Roman etc. 3. Utiliza el color negro para desarrollar tus respuestas y usa otros colores para destacar contenidos o palabras que creas necesario resaltar. 4. Recuerda entregar la actividad en formato PDF a no ser que el profesor o profesora indique lo contrario. 5. Recuerda nombrar el archivo siguiendo estas indicaciones:  * Ciclo\_Módulo o crédito\_Tema\_ACT\_número actividad\_Nombre y apellido   + Ejemplo: AF\_M01\_T01\_ACT\_01\_Maria Garcia |

|  |  |
| --- | --- |
| **Antes de empezar…** | |
| Nombre |  |
| Apellidos |  |
| Módulo/Crédito |  |
| UF (solo ciclos LOE) |  |
| Título de la actividad |  |

Actividad 1

***Patrones de refactorización***

En esta actividad aplicaremos algunos de los **patrones de refactorización** vistos en clase para optimizar el código.

*¿Qué has de hacer en esta actividad?*

**Para cada bloque de código** adjunto abajo:

1. **Crea una nueva clase** Java con el código (adjunta la evidencia)
2. Refactoriza el código
3. Adjunta el código una vez refactorizado (adjuntar evidencia)
4. **Explica** a continuación del código refactorizado **qué patrón has usado** y **comenta los cambios** que se han hecho**.**

Bloques de código a utilizar en la actividad:

***Bloque 1:***  Tenemos una clase que tiene un atributo que necesita información y funciones propias. Qué patrón de refactorización tenemos que usar para añadir otra información del cliente como edad, dni, etc.

public class Pedido {

    private int id;

    private String cliente;

    public Pedido(int id,String cliente) {

        this.id = id;

        this.cliente = cliente;

    }

}

***Bloque 2:***  Qué patrón tenemos que usar para refactorizar una clase que hace el trabajo que debería ser hecho por dos clases.

public class Cliente {

    private String nombre;

    private String telefonoTrabajo;

    private String prefijoTelefonoTrabajo;

    private String telefonoCasa;

    private String prefijoTelefonoCasa;

    public String getTelefonoTrabajo() {

        return telefonoTrabajo;

    }

    public String getTelefonoCasa() {

        return telefonoCasa;

    }

}

Actividad 2

***Documentación de una clase en Eclipse***

En esta actividad, realizaremos la **documentación de código Java**, usando las etiquetas (tags) del JavaDoc y la herramienta de generación de documentación automática JavaDoc de Eclipse.

*¿Qué has de hacer en esta actividad?*

Dado el código adjunto abajo, los pasos a realizar son los siguientes:

1. **Crear la clase Stack** (tienes el código abajo)
2. **Comprobar que no tenga errores**. En el caso que tenga, toca solucionarlos**. Comenta que has hecho para solucionarlos** y adjunta una evidencia.
3. **Documentar**:
   1. la clase
   2. cada uno de los campos y métodos
   3. las partes principales del código

Utiliza las etiquetas y comentarios apropiados. En este enlace podéis encontrar un resumen de los tipos de comentarios Java:

<http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html#comments>

Adjunta evidencia del código comentado.

Es necesario que se vea vuestro nombre como autor del código.

1. **Generad la documentación JavaDoc** con Eclipse, adjuntar evidencia.
2. **Adjuntar** en la entrega el **código Java** correctamente documentado, así como los **documentos JavaDoc generados**

Bloque de código a utilizar en la actividad:

**public** **class** Stack {

**private** **int** tamaño;

**private** Vector<Integer> elementos;

**public** Stack() {

elementos = **new** Vector<Integer>();

tamaño = 0;

}

**public** **boolean** stackVacio () {

**if** (tamaño==0) {

**return** **true**;

}

**return** **false**;

}

**public** **void** apilar ( Integer o ) {

elementos.add(tamaño, o);

tamaño++;

}

**public** Integer desapilar () {

**try** {

**if**(stackVacio())

**throw** **new** ErrorStackVacio();

**else** {

**return** elementos.get(--tamaño);

}

} **catch**(ErrorStackVacio error) {

System.***out***.println("ERROR: el stack está vacio");

**return** **null**;

}

}

**public** **int** getNumElements() {

**return** tamaño;

}

@SuppressWarnings("serial")

**class** ErrorStackVacio **extends** Exception {

**public** ErrorStackVacio() {

**super**();

}

}

}

Actividad 3

***Complejidad ciclomática***

En esta actividad, calcularemos la complejidad ciclomática de un programa.

*¿Qué has de hacer en esta actividad?*

1. Crea el grafo de flujo del código adjunto abajo (adjunta evidencia).
   1. Tenéis [herramientas online en internet para crear grafos de flujo](https://cacharrerosdelaweb.com/2015/12/cuatro-herramientas-online-para-crear-diagramas-flujo.html)
2. Calcula la complejidad ciclomática del código.
3. Identificar los caminos independientes del código.

Bloque de código a utilizar en la actividad:

ArrayList<Integer> vector = new ArrayList<Integer>();

int nr\_par = 0;

int nr\_impar = 0;

int i = 0;

while (i < vector.size()) {

if (vector.get(i) % 2 == 0) {

nr\_par++;

}

else

nr\_impar++;

i++;

}

System.out.println("nr\_par "+nr\_par);

System.out.println("nr\_impar "+nr\_par);

Actividad 4

***Implementación de pruebas***

Dado el código Java de la actividad 3, implementar las pruebas unitarias que permiten comprobar el correcto funcionamiento de la clase Stack. Para esto, usaremos Eclipse y la librería jUnit.

*¿Qué has de hacer en esta actividad?*

1. Crear una clase de prueba (New JUnit Test Case) llamada TestStack (un caso de prueba JUnit) sobre la clase Stack usando el código adjunto abajo.
2. Implementar los diferentes métodos de la clase TestStack.
   1. Añade al menos un comentario en cada método explicando brevemente que hace la prueba.
   2. Adjunta evidencia del código
3. Ejecutar el test
   1. comprobar que no hay errores en tus pruebas.
   2. Adjuntar una captura de pantalla de la ejecución de los tests implementados en jUnit.

Adjuntar el código de la clase TestStack con los tests implementados.

Bloque de código a utilizar en la actividad:

**public** **class** TestStack {

@SuppressWarnings("unused")

**private** Stack stackConElementos;

@SuppressWarnings("unused")

**private** Stack stackSinElementos;

@Before

**public** **void** setUp() **throws** Exception {

//configurar el test

}

@Test

**public** **void** testStackVacio() {

*fail*("Not yet implemented");

}

@Test

**public** **void** testApilar() {

*fail*("Not yet implemented");

}

@Test

**public** **void** testDesapilarStackSinElementos() {

*fail*("Not yet implemented");

}

@Test

**public** **void** testDesapilarStackConElementos() {

*fail*("Not yet implemented");

}

@Test

**public** **void** testGetNumElements() {

*fail*("Not yet implemented");

}

}