Trabajo Colaborativo

Entrega Previa 2 - Escenario 5

Subgrupo 11

Integrantes:

Daniela Alexandra Chamorro Guerrero

Sara Rojo Lopera

Michel Tatiana Flórez Calderón

Andrés David Arias Combita

Damaris Lizeth Hernández Cortez

Carlos Augusto Méndez Sánchez

Eddy Santiago Paipilla Galindo

TUTOR:

David Seligmann

Institución Universitaria Politécnico Grancolombiano

Paradigmas de Programación [GRUPO B01]

Junio 2024

Javadoc

Es una utilidad que viene incluida en el JDK, que genera documentación en formato HTML, a partir del código fuente.

Se escriben comentarios en código que luego se transformaran en una documentación, existen etiquetas especiales para poder interpretarlas en la generación. Al estar el código y documentación en el mismo archivo del código fuente es más sencillo mantener la sincronización de ambos.

Etiquetas

Etiqueta	Descripción
@author	Autor de la clase. Solo para clases
@version	Versión de la clase (Solo para clases)
@see	Referencia a una clase o utilidad
@param	Descripción de un parámetro, una etiqueta por cada parámetro
@return	Descripción de lo que devuelve. Solo si no es void Podrá describir valores de retorno especiales

@throws	Documenta una posible excepción que puede propagar. Habrá una etiqueta throws por cada tipo de excepción
@deprecated	Marca el método como obsoleto
@since	Indica el n° de versión desde la que existe el método

Generacion de documentacion

Para generar Javadoc desde la terminal, se siguen estos pasos:

1. Validar que se tiene Java instalado:

Comando: java -version

2. Cambiar el directorio actual:

Comando: cd <<directorio_de_la_maquina>>/Adivina_el_codigo

3. Se ejecuta el comando Javadoc con los siguientes argumentos:

- -d JavaDoc especifica el directorio donde se generará la documentación.
- -sourcepath src especifica el directorio donde se encuentran los archivos fuente.
- -subpackages com.example especifica el paquete raíz y todos sus subpaquetes para los cuales deseas generar la documentación.

Comando: javadoc -d JavaDoc -sourcepath src -subpackages com.example

4. Verificar que el index de la documentacion haya sido efectivo abriendo el documento ‘index.html’ en el navegador.

Pruebas Unitarias en JUnit

Se consideraron las siguientes pruebas unitarias:

1. **Pruebas de inicialización:**

1.1 Verificar que la cuadrícula de intentos se inicialice con 10 círculos vacíos.

1.2 Verificar que la cuadrícula de retroalimentación se inicialice con 10 rectángulos vacíos.

1.3 Verificar que la etiqueta de intentos restantes se inicialice con "Intentos restantes: 10".

1.4 Verificar que el mensaje de estado inicial sea "Bienvenido! Empieza a adivinar el código."

2. **Pruebas de selección de color:**

2.1 Verificar que, al seleccionar un círculo de color, se actualice el color del círculo seleccionado correspondiente.

2.2 Verificar que, al seleccionar un cuarto color, se deshabilite la selección de más colores.

3. **Pruebas de envío de suposición:**

3.1 Verificar que, si no se seleccionan 4 colores, se muestre un mensaje de error.

3.2 Verificar que, al enviar una suposición, se añada la suposición a la cuadrícula de intentos con los colores correspondientes.

3.3 Verificar que, al enviar una suposición, se añada la retroalimentación a la cuadrícula de retroalimentación.

3.4 Verificar que, al enviar una suposición correcta, se muestre un mensaje de éxito y se finalice el juego.

3.5 Verificar que, si se agotan los intentos, se muestre un mensaje de derrota y se finalice el juego.

Verificar que después de enviar una suposición, se borre la selección actual y se limpie el mensaje de retroalimentación.

4. **Pruebas para la lógica del juego (GameLogic):**

4.1 Verificar que el código generado tenga una longitud de 4.

4.2 Verificar que cada color en el código sea uno de los colores válidos ("Red", "Green", "Blue", "Yellow").

4.3 Verificar que el código generado sea aleatorio y no se repita.

5. **Pruebas de verificación de suposición:**

5.1 Verificar que la posición correcta se incremente en 1 si un color en la suposición coincide con el mismo color y posición en el código.

5.2 Verificar que el color correcto se incremente en 1 si un color en la suposición coincide con un color en el código, pero en una posición diferente.

5.3 Verificar que el número de intentos se incremente en 1 después de cada verificación de suposición.

5.4 Verificar que la retroalimentación devuelta contenga el número correcto de posiciones correctas y colores correctos.

6. **Prueba de inicio de nuevo juego:**

6.1 Verificar que al hacer clic en el botón "Nuevo juego", se cargue la escena del juego.

6.2 Verificar que la escena del juego tenga el título correcto ("Adivina el código - Juego").

Como generarlas:

Generar pruebas unitarias implica la creación de pequeñas pruebas automatizadas que verifican el comportamiento de unidades individuales de código, como funciones o métodos. En el contexto de Java, esto se realiza comúnmente utilizando frameworks como JUnit. Para comenzar, es esencial asegurarse de que JUnit esté incluido en el proyecto, lo cual se puede lograr añadiendo la dependencia correspondiente en el archivo pom.xml si se utiliza Maven. A continuación, se debe crear una clase de prueba en el directorio de pruebas (por ejemplo, src/test/java). Dentro de esta clase, se escriben métodos de prueba anotados con @Test, donde se configuran los escenarios de prueba, se ejecuta el código a probar y se utilizan aserciones (assertEquals, assertTrue, etc.) para verificar que los resultados sean los esperados. Finalmente, las pruebas se ejecutan utilizando un entorno de desarrollo integrado (IDE) o una herramienta de

construcción como Maven (mvn test) para asegurar que todas las pruebas se ejecuten correctamente y los resultados sean los deseados.

Consideraciones

Independencia de las pruebas:

La independencia de la prueba significa que cada prueba unitaria debe ser autónoma y no depender del resultado o del orden de ejecución de otras pruebas, también nos ayuda a mantener y refactorizar nuestras pruebas más fácilmente.

Aislamiento:

Este significa que cada prueba unitaria debe ejecutarse en un entorno separado y controlado, sin verse afectado por otras pruebas o factores externos, esto nos facilita identificar la fuente de cualquier error o prueba fácilmente además de que nuestros resultados sean más consistentes y precisos.

Cobertura de código:

La cobertura de código es uno de los parámetros con los que podremos saber que parte de nuestra fuente se ha sometido a pruebas, este nos es muy útil para evaluar la calidad del conjunto de pruebas.

Dependencia en el POM:

La unidad básica de trabajo en Maven es el llamado Modelo de Objetos de Proyecto conocido simplemente como POM (de sus siglas en inglés: Project Object Model).

Este Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, etc.

Bibliografías:

Rodríguez, A. (s. f.). *Documentar proyectos Java con Javadoc. Comentarios, símbolos, tags (deprecated, param, etc.) (CU00680B)*. aprenderaprogramar.com. https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=646:documentar-proyectos-java-con-javadoc-comentarios-simbolos-tags-deprecated-param-etc-cu00680b&catid=68&Itemid=188

¿Cuáles son los beneficios y desafíos del aislamiento y la independencia de las pruebas? (2023, 14 marzo). www.linkedin.com. https://es.linkedin.com/advice/3/what-benefits-challenges-test-isolation-independence?lang=es

García, A. P. (2018, 2 noviembre). *Cobertura: Como comprobar cuanto código prueban nuestros test - Adictos al trabajo*. Adictos Al Trabajo. https://adictosaltrabajo.com/2008/10/18/maven-cobertura/