



Escuela de Ingeniería Informática



DEPURACIÓN, PRUEBAS Y REFACTORIZACIÓN

Metodología de la programación
Curso 2023-2024

Contenidos

- Revisión Tarea Lab01 (Game2048)
- Depuración
- Pruebas
- Refactorización

Revisión de tarea. Errores

Errores más comunes

- No se ha cambiado el nombre del proyecto
- No se ha exportado bien el proyecto
- Se ha usado checkParam en lugar de proyecto útil
- No se han usado los submétodos especificados en game2048
- No se ha acabado el código o los test
- Hay errores de compilación
- Hay warnings (sobran imports)
- Hay errores en el algoritmo
- Las pruebas no funcionan

Revisión de tarea. A destacar

- Clases y test en el mismo paquete, diferente carpeta (src/test)
- Métodos con protección de paquete
 - Sólo se usan en los test (que estarán en el mismo paquete)
- Copia defensiva.
 - Cuando se devuelve o se recibe una colección.
 - Copia superficial/profunda. Objetos mutables/inmutables
 - `Arrays.copyOf(vector,longitud) // vector.clone`
 - caso especial. Vector de 2 dimensiones

Revisión de tarea. A destacar

¿Qué haremos nosotros?

- Siempre que se implemente un método `getColección()` haremos una copia superficial y devolvemos la copia de la colección.

Si es un array de 1 dimensión:

```
return arrayOriginal.clone()
```

```
o bien return Arrays.copyOf(arrayOriginal, longitudOriginal)
```

Si es un `ArrayList`:

```
return new ArrayList(ArrayListOriginal);
```

Si es una matriz haremos una copia de toda la estructura. Por tanto:

```
tipoMatriz[][] copy = new ...
```

```
for (int i=0; i < tipoMatriz.length; i++){
```

```
copy[i] = original[i].clone; // o bien copy[i]= Arrays.copyOf(original[i], long);
```

Siempre que recibamos en un constructor una colección. Haremos una copia superficial y se la asignaremos al atributo.

Revisión de tarea. A destacar

- Uso de StringBuilder
- Java code conventions

(120 caracteres)

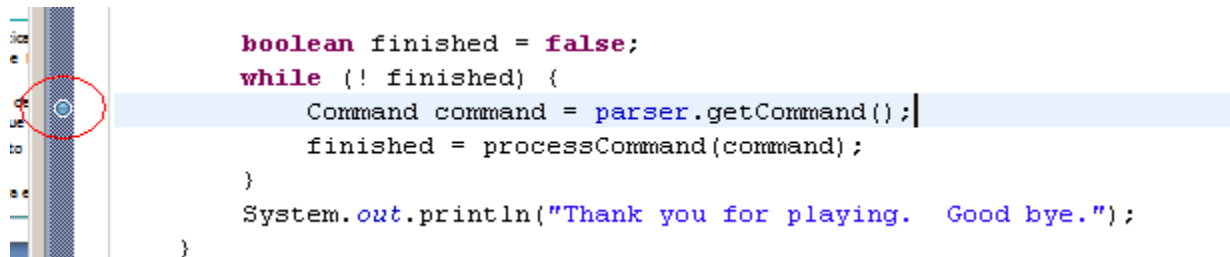
Window/preferences/general/editor/Text Editors/ show line numbers

Depuración

- **Depuración:** Consiste en encontrar y corregir errores (defectos) en los programa.
- **Depurador:** Programa que puede ejecutar tu programa a la vez y mostrarte paso a paso cómo evoluciona la ejecución y el valor de los datos en cualquier momento durante la ejecución.
- Piensa en el depurador como el VAR en el fútbol, que permite ver paso a paso lo que sucedió en una jugada determinada.
- El uso del depurador puede reducir en gran medida el tiempo que lleva perfeccionar un programa


Depuración

- Se trata de “encontrar y corregir errores o defectos del programa”
- Creamos puntos de parada “Breakpoints” sobre el margen izquierdo del editor.



- También podemos modificar sus propiedades.
 - Número de veces que el flujo de ejecución alcanza un punto dado.
 - Expresiones condicionales de parada.

Depuración

- Para **ejecutar un programa en modo debug** podemos hacerlo mediante dos opciones:
 - Con el botón de Debug 
 - O usando F11
- Cuando se ejecuta en modo depuración el IDE cambia a la **perspectiva Java Debug**
- Esta es su apariencia...

Perspectiva Debug

The screenshot shows the Eclipse IDE in the Debug perspective. The main editor displays the `Game.java` file with a breakpoint set at line 77. The Variables window on the right shows the state of the program, including the `this` object and its fields. The Outline window on the right shows the class structure. The Console at the bottom shows the output of the program.

Estado de ejecución

Variables Activas

Name	Value
this	Game (id=16)
currentRoom	Room (id=17)
description	"outside the main entrance of the university" (i...
exits	HashMap<K,V> (id=26)
parser	Parser (id=19)
commands	CommandWords (id=31)
reader	Scanner (id=33)
finished	false

Breakpoints en el código

```
boolean finished = false;
while (! finished) {
    Command command = parser.getCommand();
    finished = processCommand(command);
}
System.out.println("Thank you for playing. Good");
}

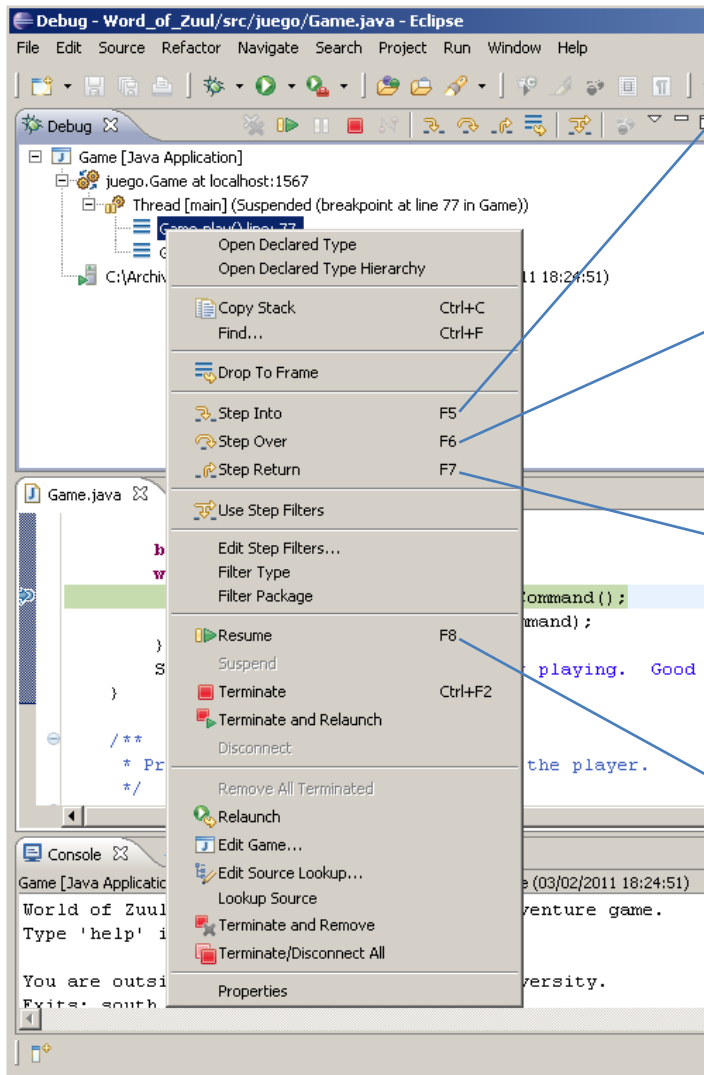
/**
 * Print out the opening message for the player.
 */
```

Consola Java

```
Game [Java Application] C:\Archivos de programa\Java\jre6\bin\javaw.exe (03/02/2011 18:24:51)
World of Zuul is a new, incredibly boring adventure game.
Type 'help' if you need help.

You are outside the main entrance of the university.
Exits: south east west
```

Comandos de Debug más comunes



- Salta a la siguiente instrucción incluso dentro de un método o función.
- Salta a la siguiente instrucción sin entrar en métodos o funciones.
- Regresa a la sentencia de llamada del actual método o función.
- Ejecuta hasta el siguiente breakpoint.

Ejercicio – Depuración

- Crear un nuevo workspace
- Importar el proyecto `student_lab02_bugs`
 - Cambiar el nombre
- Analizar el código
- Ejecutar la aplicación
- Encontrar los errores usando el depurador

Ejercicio – Depuración

- Ejecutar y ver funcionamiento de pila
- Analizar valores de variables
- Punto de ruptura en `firstRepeatedCharacter` **dentro** del bucle. Analizar propiedades
 - Parar después de 2 iteraciones (Hit count 2)
 - Parar si cumple condición
- Crear expresión para analizar algún dato
 - Ver cuánto vale `word.length()`
- Punto de ruptura en excepción
 - Analizar valores de variable justo antes de la excepción

Ejercicio – Depuración

- Punto de ruptura en firstRepeatedCharacter dentro del bucle

```
13= /**
14  * Constructor que crea un objeto analizador con la palabra recibida
15  *
16  * @param aWord La palabra a ser analizada
17  */
18= public WordAnalyzer(String aWord) {
19     word = aWord;
20 }
21
22= /**
23  * Obtiene el primer carácter repetido. Un carácter está <i>repetido</i>
24  * aparece al menos dos veces en posiciones adyacentes.
25  * Por ejemplo, 'l' está repetida en "pollo", pero 'o' no está.
26  *
27  * @return el primer caracter repetido, o 0 si no lo encuentra
28  */
29= public char firstRepeatedCharacter() {
30     for (int i = 0; i < word.length(); i++) {
31         char ch = word.charAt(i);
32         if (ch == word.charAt(i + 1))
33             return ch;
34     }
35     return 0;
36 }
37
38= /**
```

Variables

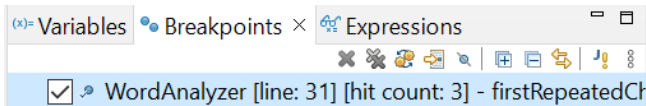
Name	Value
no method return value	
this	WordAnalyzer (id=22)
word	"abcd" (id=27)
i	0

Ejecución parada cuando i = 0.
Palabra abcd

Punto de ruptura en línea 31

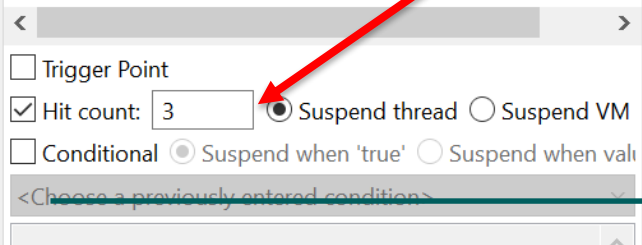
Ejercicio – Depuración

- Punto de ruptura en firstRepeatedCharacter **dentro del bucle**



Punto de ruptura en línea 31

Para cuando
pasen 3
iteraciones



Name	Value
no method return value	
this	WordAnalyzer (id=22)
word	"abcdefg" (id=24)
i	2

i vale 2 porque empieza en 0

Ejercicio – Depuración

- Punto de ruptura en firstRepeatedCharacter **dentro** del bucle

Calculo longitud de word

Name	Value
<code>word.length()</code>	7
<code>word.charAt(2)</code>	c
Add new expression	

Calculo carácter en posición 2

En parada se pueden ejecutar expresiones para averiguar valores

Pruebas - Testing

- Las **pruebas del software** son una manera de proporcionar información acerca de su **calidad**
 - Las pruebas **no demuestran la corrección de software**
 - La corrección de software sólo se puede demostrar por medio de **métodos formales**
- Hay diferentes tipos de pruebas de software
 - ***Unit testing**, Integration testing, System testing, Regression testing, Acceptance testing (alpha, beta)*
 - *Performance testing, Usability testing, Security testing, Internationalization testing, Stress testing*

Unit testing (pruebas unitarias)

- Nos centraremos en las pruebas unitarias: prueban la funcionalidad de una sección específica (unit) del código.
- JUnit es un “Framework” para **automatizar** las pruebas unitarias de programas Java.
 - Permite realizar las pruebas de regresión cuando se realizan cambios en el código. Automatizar las pruebas para poder pasarlas cada vez que se hace algún cambio en cualquier parte del código. Asegurar así que los casos de prueba que ya habían sido probados y fueron exitosos permanezcan así.
- Escrito por Erich Gamma y Kent Beck. Es Open Source y esta disponible en <http://www.junit.org/>
- En Eclipse viene integrado directamente.

Uso de Junit en Eclipse

- Crea un nuevo proyecto `apellido1_apellido2_nombre_lab02_analyzer`
- Crea un paquete llamado `uo.mp.lab02.analyzer.model`
- Copia el fichero `WordAnalyzer.java` (sin errores) en el nuevo paquete

Para los test crea

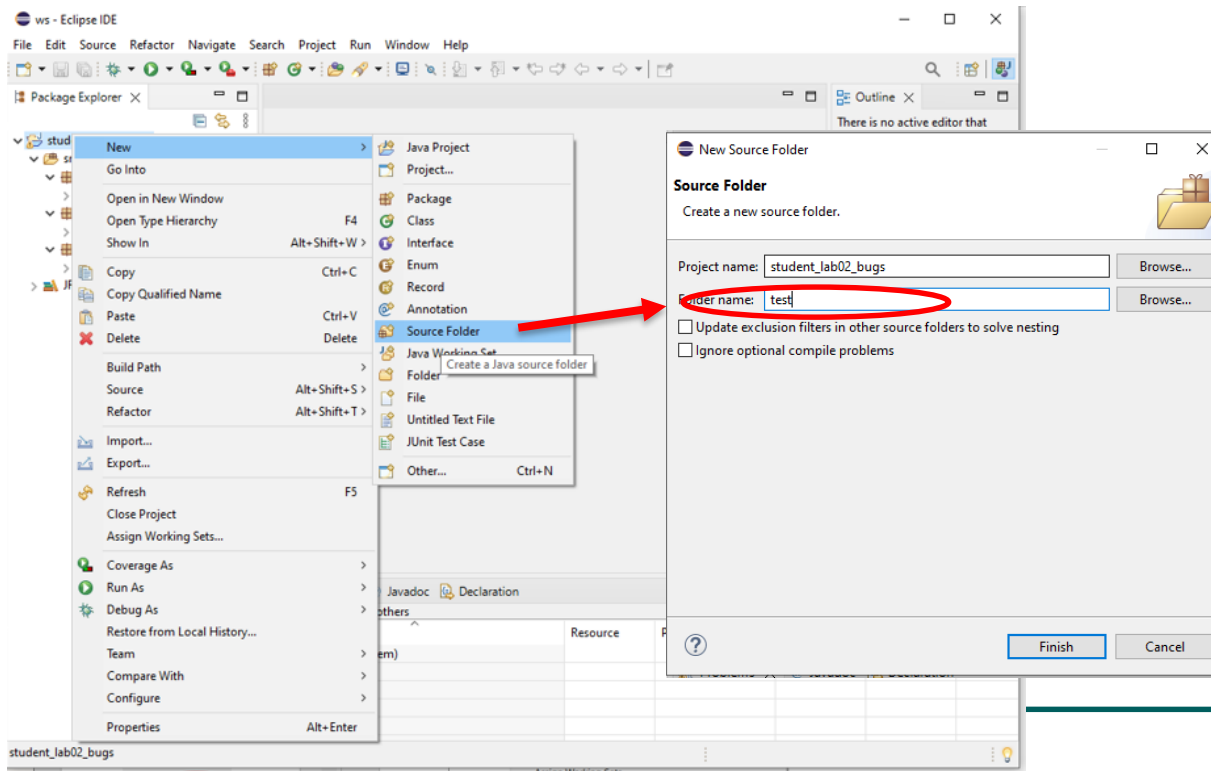
- Una **carpeta para los test**, llamada test
 - Desde **File/new/source folder**
- Un **paquete para cada paquete de clases** a probar
 - Llamado `uo.mp.lab02.analyzer.model`
- Una **clase de pruebas por cada método** a probar
 - Llamada `NombreClaseNombreMétodoTest`
 - Ejemplo: clase `WordAnalyzerFirstRepeatedCharacterTest`
 - Desde **File/ new / Junit Test Case**
- Un **método por cada caso** (o escenario) de prueba
 - Llamado como el caso o como el caso y resultado
 - Ejemplo método `emptyWord` o bien `emptyWordReturn0`

ORGANIZACIÓN

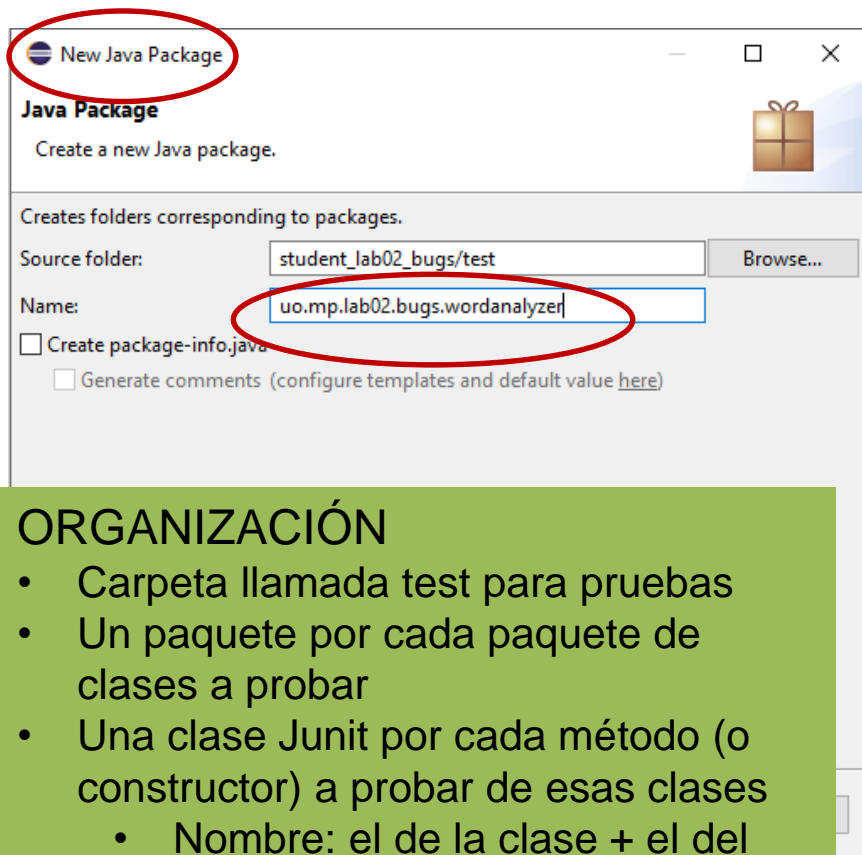
- Carpeta test para pruebas
- Un paquete por cada paquete de clases a probar
- Una clase JUnit por cada método de las clases a probar
- Un método por caso a probar

Uso de JUnit

- Se debe crear una carpeta (**source folder**) llamada test
- Dentro de esta carpeta se crea un **paquete con el mismo nombre del paquete donde está la clase a probar**

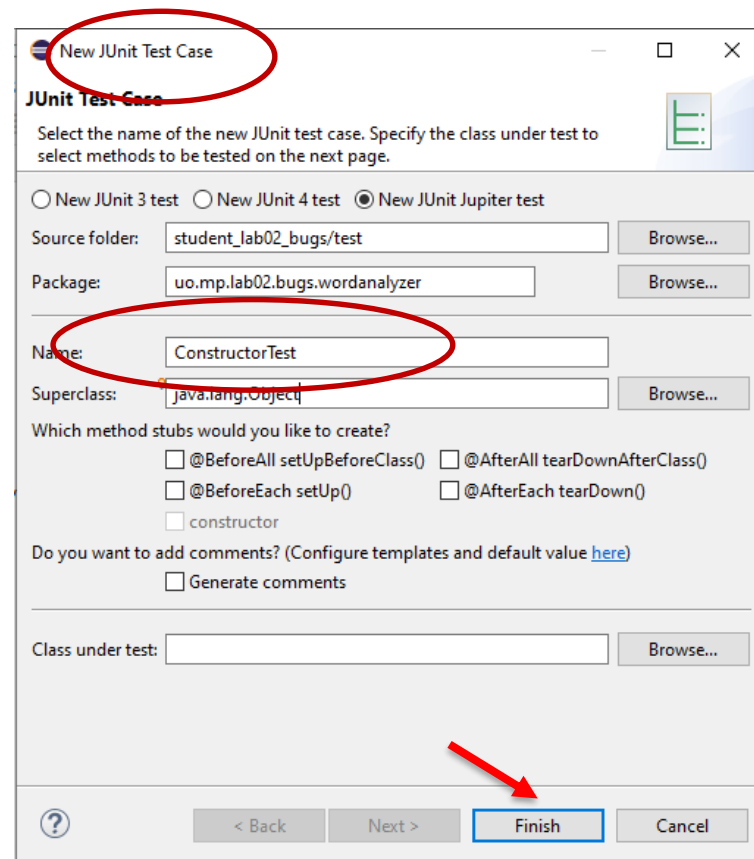


Usando JUnit



ORGANIZACIÓN

- Carpeta llamada test para pruebas
- Un paquete por cada paquete de clases a probar
- Una clase Junit por cada método (o constructor) a probar de esas clases
 - Nombre: el de la clase + el del método acabado en Test

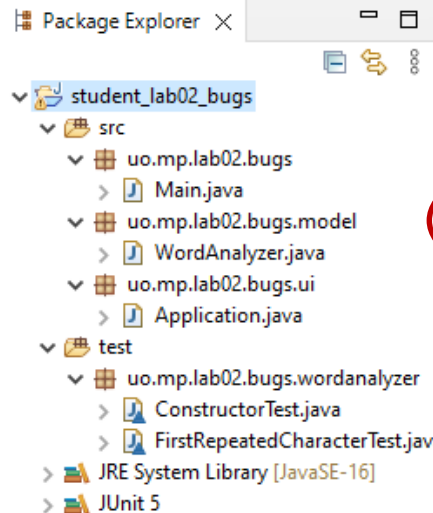


Se validan parámetros en todos los métodos públicos y constructores

Usando JUnit

ws - student_lab02_bugs/test/uo/mp/lab02/bugs/wordanalyzer/ConstructorTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help



```
1 package uo.mp.lab02.bugs.wordanalyzer;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 /*
6  * Casos:
7  * 1- Recibe null en lugar de palabra
8  * 2- Recibe una cadena vacía
9  * 3- Recibe una cadena con varios caracteres
10 */
11
12 import org.junit.jupiter.api.Test;
13
14 import uo.mp.lab02.bugs.model.WordAnalyzer;
15
16 class ConstructorTest {
17     // caso 1 Recibe null en lugar de palabra
18     @Test
19     void nullInsteadOfWord() {
20         String word = null;
21         try {
22             WordAnalyzer analyzer = new WordAnalyzer(word);
23             fail("Esperaba excepción");
24         } catch (IllegalArgumentException e) {
25             assertEquals("Null en lugar de palabra", e.getMessage());
26         }
27     }
28
29 }
```

```
class ConstructorTest {
    // caso 1 Recibe null en lugar de palabra
    @Test
    void nullInsteadOfWord() {
        String word = null;
        try {
            new WordAnalyzer(word);
            fail("Esperaba excepción");
        } catch (IllegalArgumentException e) {
            assertEquals("Null en lugar de palabra", e.getMessage());
        }
    }
}
```

Variable local no usada

El Código NUNCA debe tener warnings

Usando JUnit

ORGANIZACIÓN

- **Carpeta** llamada **test** para pruebas
 - Tipo* carpeta: Source Folder
 - Nombre* carpeta: test
- Un **paquete** para cada paquete que tiene las clases a probar
 - Nombre*: el del paquete que contiene las clases a probar
- Una **clase** Junit por cada método o constructor a probar de las clases del paquete
 - Nombre*: el de la clase + el del método o constructor acabado en test
- Un **método** por cada caso de uso
 - *Nombre*: indicativo del caso que trata

Comentarios para cada caso

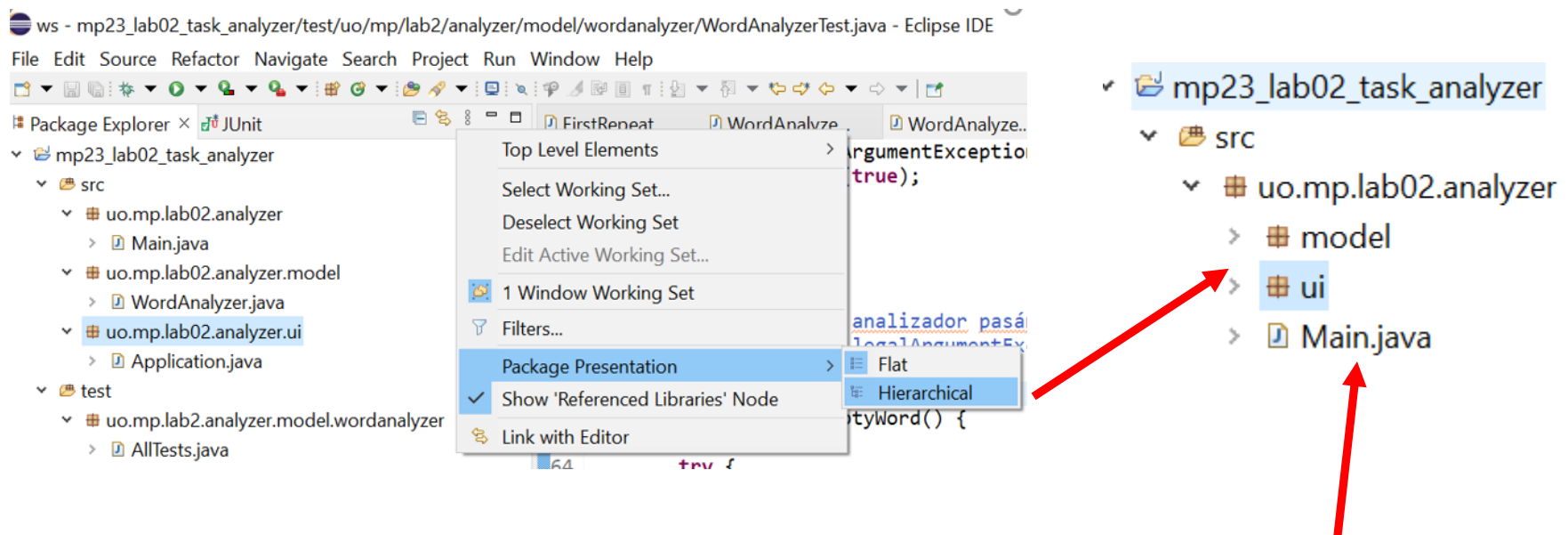
- En caso de constructor, la situación inicial no tiene aún objeto creado
 - GIVEN estado inicial
 - WHEN se ejecuta con ciertas condiciones
 - THEN estado final del objeto si cambia y resultado devuelto (si tiene)

- Ejemplo

```
public class WordAnalyzerTest {  
  
    /*  
     * Casos  
     * 1- Palabra cualquiera -> se crea el analizador  
     * 2- null como parámetro -> salta excepción  
     */  
    |  
    /**  
     * GIVEN palabra cualquiera  
     * WHEN Se intenta crear un analizador dicha palabra  
     * THEN se crea el analizador conteniendo esa palabra  
     */  
    |  
    @Test  
    public void only1CharacterWordIsOk() {  
        WordAnalyzer wordAnalyzer = new WordAnalyzer("abcd");  
        assertNotNull(wordAnalyzer);  
        assertEquals("abcd", wordAnalyzer.toString());  
    }  
}
```


Usando JUnit

- Se puede mostrar representación jerárquica de paquetes



El Main está en el paquete raíz analyzer

Refactorización

- *Técnica para la reestructuración de código, modificando su estructura interna pero sin cambiar su comportamiento*
 - Mejoran aspectos del software:
 - Mantenibilidad, lectura, rendimiento...
 - Existen varias técnicas de refactorización
 - Muchas son soportadas por el IDE de Eclipse
 - En la opción del menú Refactor
 - Las usaremos en las prácticas según las vayamos necesitando
-

Refactor → Rename

- Hay dos opciones:
 - Refactor | Rename
 - Alt + Shift + R
- Cambia el nombre de la clase `WordAnalyzer` en el proyecto `bugs`
- **¿Qué cambios se han producido en el código?**
- Si se hace de forma manual
 - Habría que cambiar el nombre de la clase, del constructor, el tipo de cada variable en la clase principal, ...
 - Se pueden cometer errores. Implica pérdida de tiempo.

Tarea a entregar

- **Proyecto game2048 completado** con las operaciones que se piden en el enunciado y con el siguiente nombre
apellido1_apellido2_nombre_lab01_task_game2048
Para renombrar (un proyecto, paquete, Sobre él pulsar botón derecho y Refactor/rename
- **Antes de entregar, revisa tu tarea de forma autónoma usando la checkList proporcionada** para asegurarte de que está bien..

Recuerda:

- Todas las tareas, deberán subirse comprimidas en un único fichero al campus.
- Las tareas **deben estar acabadas 24 horas antes** de la siguiente clase de laboratorio.
- Más de 2 tareas inválidas o no entregadas supone **la pérdida de la evaluación continua**