



Escuela de Ingeniería Informática



DEPURACIÓN, PRUEBAS Y REFACTORIZACIÓN

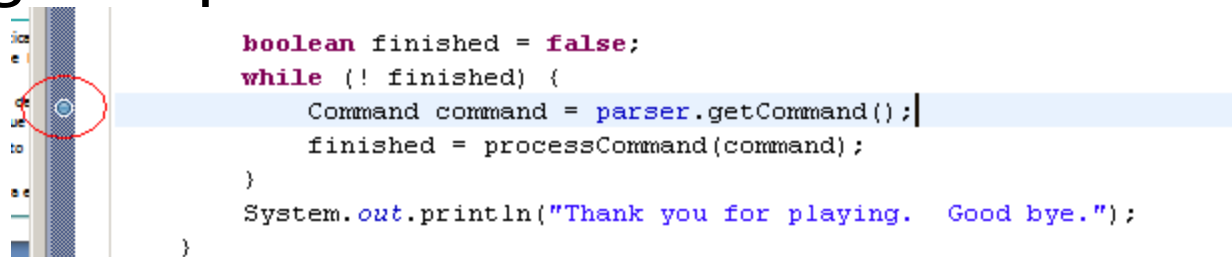
Metodología de la programación
Curso 2020-2021

Ejercicio – Depuración

- Crear un nuevo workspace
- Importar el proyecto `student_session02_bugs`
– Cambiar el nombre
- Analizar el código
- Ejecutar la aplicación
- Encontrar los errores usando el depurador


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 pasa antes de detenerse.
 - Condiciones para la suspensión.

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` window at the bottom shows the output of the program.

Estado de ejecución

Variables Activas

Breakpoints en el código

Consola Java

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

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

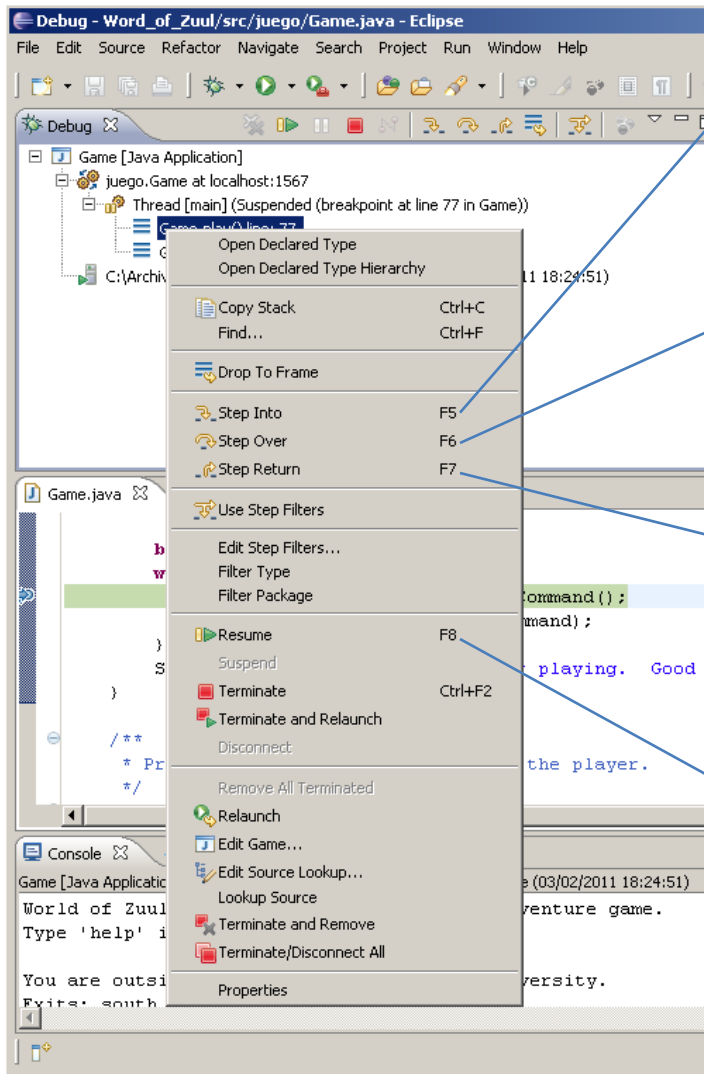
/**
 * Print out the opening message for the player.
 */
```

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.

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)

- JUnit es un “Framework” para **automatizar** las pruebas unitarias de programas Java.
- Permite realizar las pruebas de regresión cuando estamos realizando cambios.
- Escrito por Erich Gamma y Kent Beck.
- Es Open Source y esta disponible en <http://www.junit.org/>
- En Eclipse viene integrado directamente.

Usando JUnit

- Crea un nuevo proyecto `nombre_apellido1_apellido2_session02_analyzer`
- Crea un paquete llamado `uo.mp.s2.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 clase** a probar
 - Llamado `uo.mp.s2.analyzer.model.wordanalyzer` (**minúsculas**)
- Una **clase de pruebas por cada método** a probar
 - Llamada `NombreMétodoTest`
 - Ejemplo: clase `FirstRepeatedCharacterTest`
 - 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`

Usando JUnit

New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

☐ New JUnit 3 test ☒ New JUnit 4 test ☐ New JUnit Jupiter test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

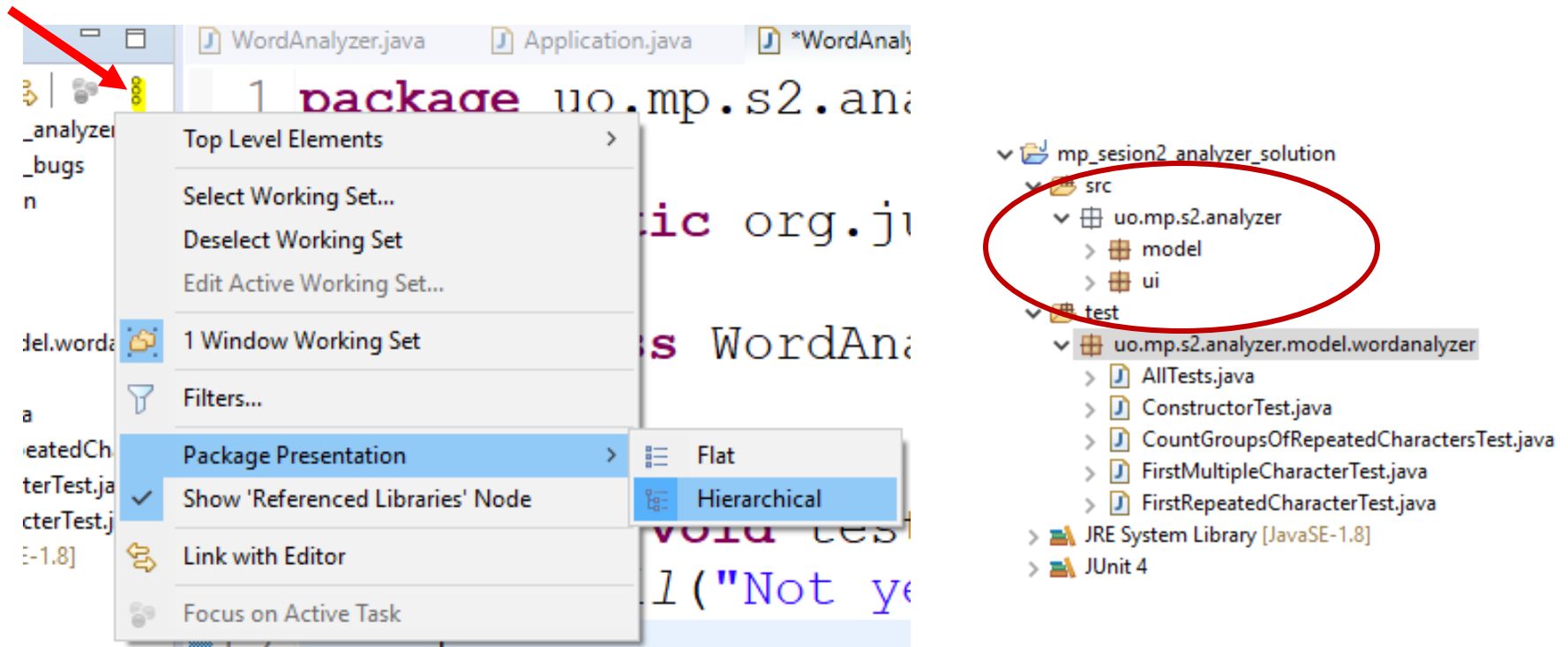
Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Class under test:

Usando JUnit

- Se puede mostrar representación jerárquica de paquetes

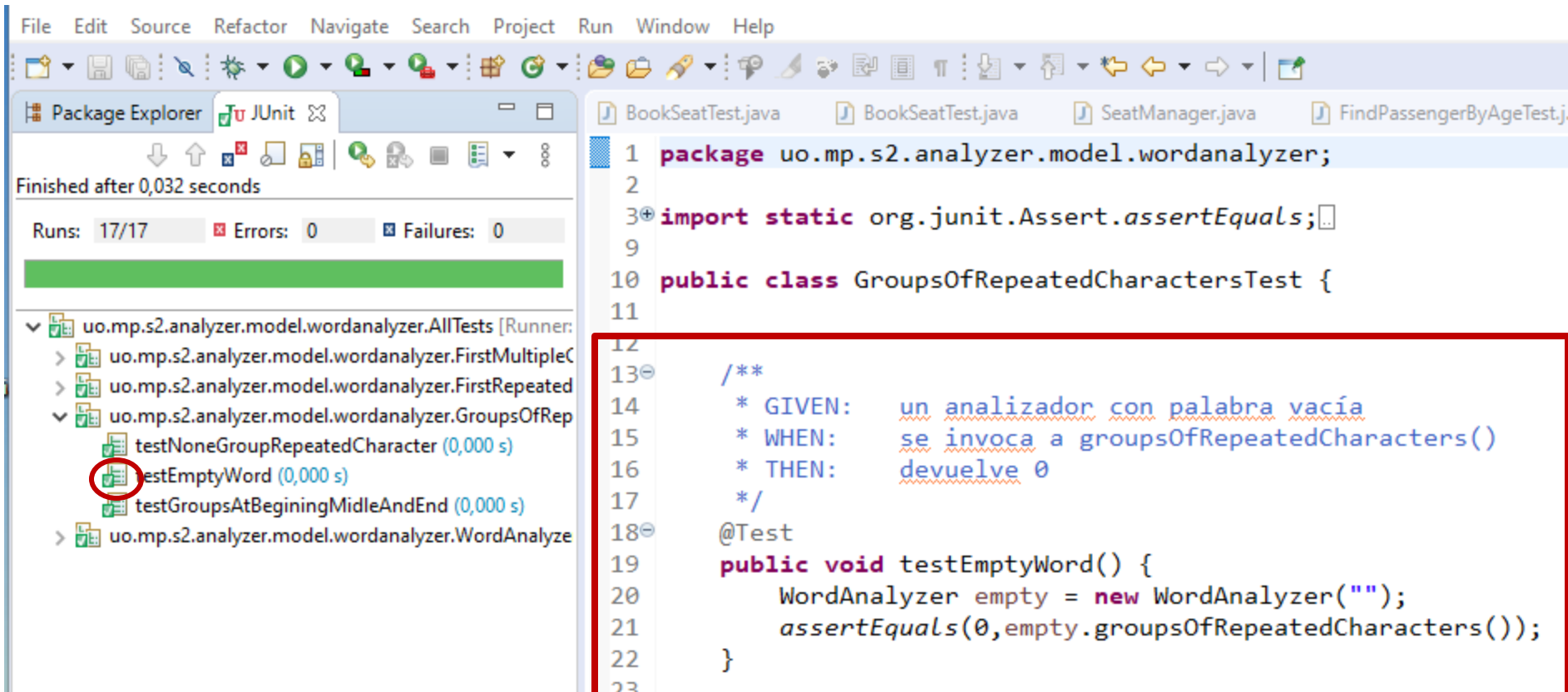


Usando JUnit

- Comenta cada caso con la estructura
 - GIVEN estado inicial
 - WHEN se ejecuta con ciertas condiciones (valores de parámetros)
 - THEN estado final del objeto si cambia y resultado devuelto (si tiene)
- Ejemplo

```
/**
 *
 * GIVEN analizador para cadena vacía
 * WHEN ejecuta firstRepeatedCharacter
 * THEN devuelve 0
 */
@Test
public void testEmptyWord() {
    WordAnalyzer emptyWordAnalyzer = new WordAnalyzer("");
    assertEquals(0, emptyWordAnalyzer.firstRepeatedCharacter());
}
```

Implementando Tests



The screenshot displays an IDE interface with two main panels. The left panel, titled 'JUnit', shows the results of a test run. It indicates 'Finished after 0,032 seconds' and 'Runs: 17/17', 'Errors: 0', and 'Failures: 0'. A tree view under 'uo.mp.s2.analyzer.model.wordanalyzer.AllTests [Runner:]' lists several test methods. The method 'testEmptyWord (0,000 s)' is highlighted with a red circle. The right panel shows the source code of 'BookSeatTest.java'. The code defines a package, imports JUnit's Assert, and defines a class 'GroupsOfRepeatedCharactersTest'. A Javadoc comment and a test method 'testEmptyWord()' are highlighted with a red box. The Javadoc comment describes the test scenario in Spanish: 'un analizador con palabra vacía', 'se invoca a groupsOfRepeatedCharacters()', and 'devuelve 0'. The test method 'testEmptyWord()' creates a 'WordAnalyzer' instance with an empty string and asserts that 'groupsOfRepeatedCharacters()' returns 0.

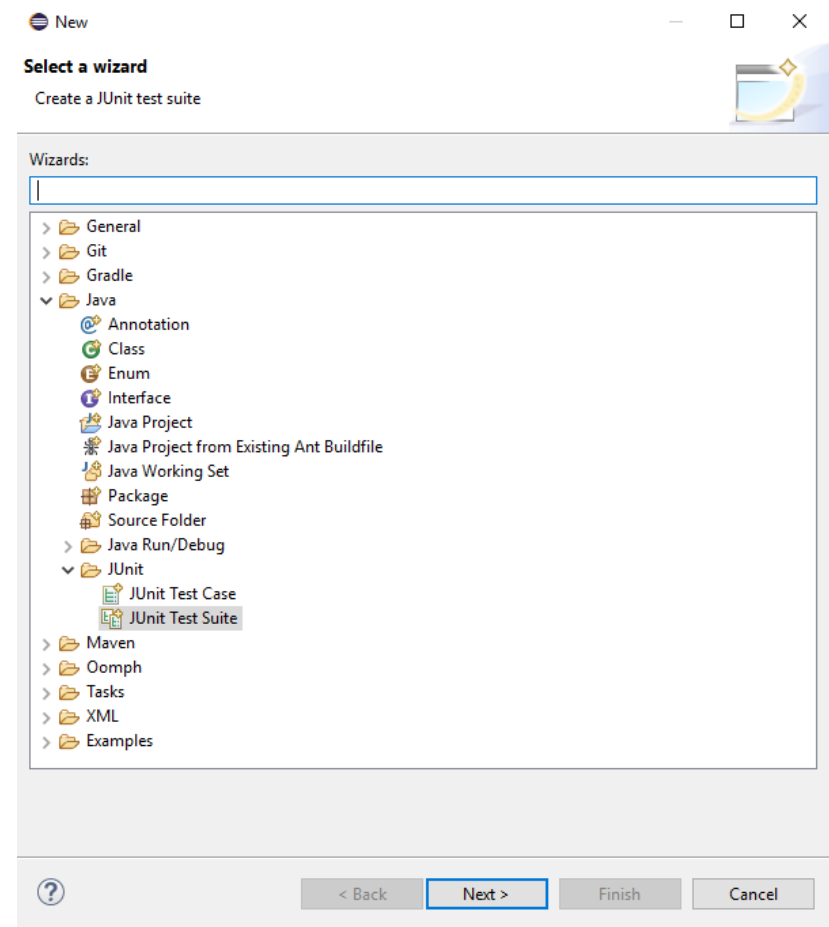
```
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit
Finished after 0,032 seconds
Runs: 17/17 Errors: 0 Failures: 0
uo.mp.s2.analyzer.model.wordanalyzer.AllTests [Runner:]
  > uo.mp.s2.analyzer.model.wordanalyzer.FirstMultipleC
  > uo.mp.s2.analyzer.model.wordanalyzer.FirstRepeated
  > uo.mp.s2.analyzer.model.wordanalyzer.GroupsOfRep
    testNoneGroupRepeatedCharacter (0,000 s)
    testEmptyWord (0,000 s)
    testGroupsAtBeginingMidleAndEnd (0,000 s)
  > uo.mp.s2.analyzer.model.wordanalyzer.WordAnalyze

BookSeatTest.java BookSeatTest.java SeatManager.java FindPassengerByAgeTest.j
1 package uo.mp.s2.analyzer.model.wordanalyzer;
2
3 import static org.junit.Assert.assertEquals;
9
10 public class GroupsOfRepeatedCharactersTest {
11
12
13 /**
14  * GIVEN: un analizador con palabra vacía
15  * WHEN: se invoca a groupsOfRepeatedCharacters()
16  * THEN: devuelve 0
17  */
18 @Test
19 public void testEmptyWord() {
20     WordAnalyzer empty = new WordAnalyzer("");
21     assertEquals(0, empty.groupsOfRepeatedCharacters());
22 }
23
```

Implementando Tests

Para ejecutar todas las pruebas
juntas AllTest

File/New/Other/JUnit/JUnit Test Suite



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 una pérdida importante de tiempo

Ejercicio obligatorio

- El enunciado del ejercicio se encuentra en el fichero:
[2021EnunciadoTareaSesión2.pdf](#)
- Los ejercicios deben estar acabados **2 días** antes de la siguiente clase de laboratorio.
- **Todas las tareas de trabajo autónomo** (no presencial) que se pidan, **deberán subirse al campus virtual**.
- Más de 2 tareas inválidas o no entregadas supone **la pérdida de la evaluación continua**

Enlaces y Bibliografía

- **Eclipse Debugging:**

<http://help.eclipse.org/mars/topic/org.eclipse.jdt.doc.user/concepts/cdebugger.htm>

- **JUnit Testing:**

P.Tahchiev, F. Leme, V. Massol, G. Gregory. JUnit in Action, Second Edition. Manning Publications. 2010.

- **Refactoring:**

Martin Fowler. Refactoring. Improving the Design of Existing Code. Addison-Wesley. 1999.