Seminar 2
Eclipse and Junit
PROGRAMMING 3

David Rizo, Pedro J. Ponce de León
Department of Software and Computing Systems
University of Alicante

Contents

Installation

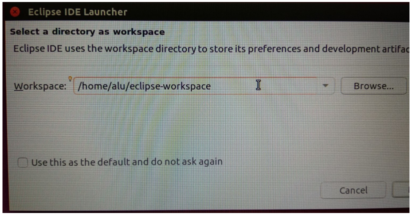
In *Programming 3* we will use version 2022-06 (although later versions should also work)

- Find it in `https://www.eclipse.org/downloads/packages/`
- Download *Eclipse IDE for Java Developers*
- Uncompress it and run the `eclipse` program

David Rizo, Pedro J. Ponce de León

Workspace

- Eclipse stores all the configuration and projects under a *workspace* folder
- When Eclipse starts, you have to choose a location for the *workspace*.



Eclipse IDE Launcher

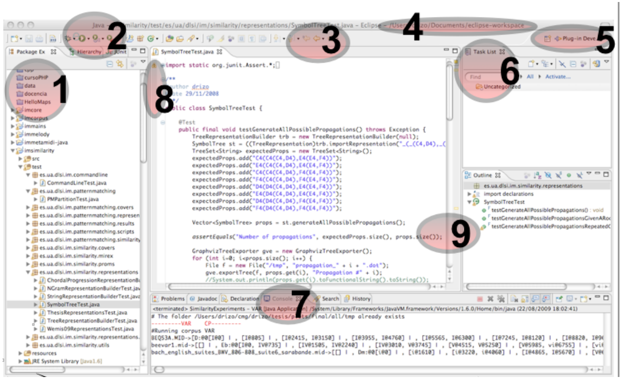
Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifac

Workspace: /home/alu/eclipse-workspace

- Select a folder in your home directory (in the lab). Eclipse will create if it does not exist.
- Select `File>Switch workspace` to change workspace whenever you want

Interface


1
2
3
4
5
6
7
8
9
Package Ex
Hierarchy
JUnit
SymbolTreeTest.java
Task List
Find
All
Activate...
Uncategorized
Outline
es.ua.dlsi.im.similarity.representations
import declarations
SymbolTreeTest
testGenerateAllPossiblePropagations() : void
testGenerateAllPossiblePropagationsGivenARo
testGenerateAllPossiblePropagationsRepeated
Plug-in Deve
Problems
Javadoc
Declaration
Console
Search
History
cursoPHP
data
docencia
HelloMaps
imcore
imcorpus
immains
immelody
immetamidi-java
imsimilarity
src
test
es.ua.dlsi.im.commandline
CommandLineTest.java
es.ua.dlsi.im.patternmatching
PMPartitionTest.java
es.ua.dlsi.im.patternmatching.covers
es.ua.dlsi.im.patternmatching.represen
es.ua.dlsi.im.patternmatching.results
es.ua.dlsi.im.patternmatching.scripts
es.ua.dlsi.im.patternmatching.similarity
es.ua.dlsi.im.similarity.covers
es.ua.dlsi.im.similarity.mirex
es.ua.dlsi.im.similarity.proms
es.ua.dlsi.im.similarity.representations
ChordalProgressionRepresentationB
NGramRepresentationBuilderTest.jav
StringRepresentationBuilderTest.java
SymbolTreeTest.java
ThesisRepresentationsTest.java
TreeRepresentationBuilderTest.java
Wemis09RepresentationsTest.java
es.ua.dlsi.im.similarity.representations.
es.ua.dlsi.im.similarity.utils
resources
JRE System Library [Java1.6]

ilarity/test/es/ua/dlsi/im/similarity/representations/SymbolTreeTest.java – Eclipse – /Users/drizo/Documents/eclipse-workspace

```
import static org.junit.Assert.*;

/**
 uthor drizo
 ate 29/11/2008
 */
 lic class SymbolTreeTest {

    @Test
    public final void testGenerateAllPossiblePropagations() throws Exception {
        TreeRepresentationBuilder trb = new TreeRepresentationBuilder(null);
        SymbolTree st = ((TreeRepresentation)trb.importRepresentation("_(_((C4,D4),_(
        TreeSet<String> expectedProps = new TreeSet<String>();
        expectedProps.add("C4(C4((C4,D4),E4(E4,F4))");
        expectedProps.add("E4(C4((C4,D4),E4(E4,F4))");
        expectedProps.add("C4(C4((C4,D4),F4(E4,F4))");
        expectedProps.add("F4(C4((C4,D4),F4(E4,F4))");
        expectedProps.add("D4(D4((C4,D4),E4(E4,F4))");
        expectedProps.add("E4(D4((C4,D4),E4(E4,F4))");
        expectedProps.add("D4(D4((C4,D4),F4(E4,F4))");
        expectedProps.add("F4(D4((C4,D4),F4(E4,F4))");

        Vector<SymbolTree> props = st.generateAllPossiblePropagations();

        assertEquals("Number of propagations", expectedProps.size(), props.size());

        GraphvizTreeExporter gve = new GraphvizTreeExporter();
        for (int i=0; i<props.size(); i++) {
            File f = new File("/tmp", "propagation_" + i + ".dot");
            gve.exportTree(f, props.get(i), "Propagation #" + i);
            //System.out.println(props.get(i).toFunctionalString().toString());
```

```
<terminated> SimilarityExperiments - VAR [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java (22/08/2009 18:02:41)
# The folder /Users/drizo/cmg/drizo/tesis/plots/final/all/tmp already exists
---------VAR    CP---------
#Running corpus VAR
BEQS3A.MID->[D:00[I00] | , [I0805] | , [I02415, I03150] | , [I03955, I04760] | , [I05565, I06300] | , [I07245, I08120] | , [I08820, I09
beevar1.mid->[[] | , Eb:00[I00, IV0735] | , [IV01505, IV02240] | , [IV03010, V03745] | , [V04515, V05250] | , [V05985, vi06755] | , [vi
bach_english_suites_BWV_806-808_suite6_sarabande.mid->[[] | , Dm:00[i00] | , [i01610] | , [i03220, i04060] | , [I04865, I05670] | , [V0
```

Tools

1. Projects and packages
2. Run and debug
3. File explorer
4. Current workspace
5. Perspective
6. A view: tasks
7. Console
8. Breakpoints, link to solve errors
9. Errors, warnings, TO-DO

Project creation

- `File > New > Java project`
  - Project name
- A directory cotaining the following sub-directories and files will be created:
  - A folder named `src/` for the source code
  - A folder named `bin/` for the compiled code
  - Hidden files `.project` and `.classpath`
    - These files contain project metadata, such as the JDK version to be used and the *classpath*, which will point to the folder `bin`.

Importing a project

To import a project, select `File > Import > General > Existing Projects into Workspace` and choose `Select root directory:` or `Select archive file:`, depending on wether the project to be imported is in a directory or in a compressed file.

Task

Download the preconfigured Eclipse project from the assignments web page and import it into Eclipse. This should create an Eclipse project with name **prog3-base** that contains two source code folders `src` and `test`. The first one is where your code goes. The second one is for testing code.
This project is configured for using . . .

- . . . the preconfigured JDK version (JDK 1.8 in the computer labs).
- . . . UTF-8 as character encoding for the new source files to be created.
- . . . Unix-style line breaks (char `'\n'`) in the source-code files.

Importing classes

To import external `.java` files, open the operating-system file browser, copy the files into the clipboard and paste them into the package view.

Task

- Add package `es.ua.dlsi.prog3.p1` to source folder `src`
  - Right-click on the folder, then `New...  -> Package.`
- Add, if you have it, the source file `Coordinate.java` of the 1st Practical Assignment to the package you have just created. If, you didn't have it, move on to next page.

Classes

- To create a new class: `File > New > Class`
- Introduce name, package, and, optionally, if you want an empty `main` method to be added

Task

- Create a new class named *Coordinate* in package `es.ua.dlsi.prog3.p1`, and add the private attribute `double[] components`. Type `/**` before their declaration, hit *enter* and write the *javadoc* documentation.
- Create one of the constructors of the class according to description of the 1st Practical Assignment. Add the constructor's documentation as explained above.
- In case your code contains errors, use the hints on the left edge of the code editor.

Run from Eclipse

- Since a particular project may include more than one class with a `main` method, the easiest way is to right-click on the class containing the `main` method to run and select `Run as > Java application`.
- This will create a new run configuration (menu `Run > Run configurations`), which can be edited to add command-line parameters to your program.

Task

- Add a method `main` to `Coordinate`. Leave it empty:
  `public static void main(String[] args) { }`
- Run it as described above.

Run from a terminal

Actividad

Do the following:

- Open a terminal (console).
- Go to the Eclipse project's folder.
- Run the command `java -cp bin es.ua.dlsi.prog3.p1.Coordinate` (*Eclipse* automatically compiles classes and puts the .class files into folder `bin`).

Debug

- Select `Run > Debug` (there is a button for this in the toolbar as well) to run your application in debug mode.
- To set a *breakpoint*, walk through the code and place your cursor on the marker bar (along the left edge of the editor area) on the line with the suspected code; double-click to set the breakpoint.
- Notice that Eclipse has switched to the *Debug* perspective.

Help

Step into (F5)	Run step by step stepping into every method.
Step over (F6)	Run next code line in a single step.
Step return (F7)	Run the remaining code in the current method and return to the invoking point.
Resume (F8)	Resume the execution till the next breakpoint (or the end of applicaction).
Run to line (^R)	Resume the execution till the line where the curso is.

Debug

Task

1. Add this code to your `main` method:

```
double[] d1 = new double[] { 2.5, 3.4 };
double[] d2 = new double[] { 2.5, 3.4, -3.2 };
Coordinate c1 = new Coordinate(d1);
Coordinate c2 = new Coordinate(d2);
System.out.println(c1.getDimensions());
System.out.println(c2.getDimensions());
```

2. Set a *breakpoint* at the first code line in method `main`, and
3. run the method line by line.

Code generation

- Implementing some operations (e.g., `equals`, `hashCode` or `toString`) is usually routine.
- Eclipse can write some draft excerpts of code for you; right-click on the source file and select `Source > Generate toString()` or `Source > Generate hashCode and equals()`.

Task

Automatically generate the methods `hashCode` and `equals` of the class `Coordinate` assignment.

WARNING

Methods generated in this way do not always do what we want them to do. For example, `toString()` might create a string with a different format, or `equals()` might compare objects in a different way to how we want them to be compared.

Unit tests

- A **unit test** is a piece of code that verifies a specific use case of a software component according to its specification.
- Each test is configured to test a particular use case of a class interface.
- Tests are organized into test sets or **suites**. Each test suite is associated with a class.
- For example, conditions or limit values of the method arguments are tested, or conditions causing a method to throw an exception.

JUnit

- **JUnit** is the most widely used tool unit testing in Java.
- In Eclipse it is configured in `Project > Properties > Java Build Path > Libraries > Add Library`
- We use *JUnit 4*. This library is already included in the base project you imported.

JUnit

Source code for unit test files is placed in independent `.java` files

Task

- Uncompress the file `tests_p1.tgz` containing the tests. Copy and paste the folder `es` into the project's source folder `test` (the files containing the source code for the tests also belong to the package `es.ua.dlsi.prog3.p1`).
- Update the project in Eclipse (F5)

To run the tests, right-click on the package or class containing them and choose `Run as > JUnit test`

JUnit

Open the file with unit tests *CoordinateTest.java*

- Look at the attributes. They are references to the objects to be used by the tests.
- Methods with annotations `@Before` configure the tests. They are executed before each method annotated as `@Test`.
- Methods `@Test` contain unit tests (methods *assert* or assertions)
- `assertEquals` checks that the expected value matches the actual one. The parameters are in this order: title (optional), expected value, real value, difference in absolute value allowed (optional, useful for real values).
- `assertTrue` and `assertFalse` check that their arguments are `true` or `false`, respectively.
- `fail` produces a test failure when executed.

JUnit

Actividad

- Run the test: `Run -> Debug as... -> JUnit Test` on the file with the test (those having `fail` instructions will fail). The tab `JUnit` is opened ad you will see the result of the execution of the test.
- Choose a test that fails. In the panel `Failure trace` double click on the first line indicating `at es.ua.dlsi.prog3.p1.CoordinateTest ....` It will take you to the line that produced the error.
- Change some expected value in a test that does not fail. Now it will fail and by selecting the test in the the panel `Failure trace` you will see why in the first line.

New unit test

To create a new unit test for a class, right-click on its name and select `New > JUnit test case`.

- Choose JUnit 4
- Type `test` (instead of `src`) in the directory field `Source folder`.

Task

- Implement a method in `Coordinate` that returns the sum of its components.
- Create a unit test (or several) to check that your method works as expected.
- To run all the tests, right-click on the project name and select `Run as > JUnit test`
- You can also run a specific test class, a specific test within a class or just the tests that failed.
- Delete the method and its tests when you are done.