eclipse.org/
ibm.com/university/

Java development with Eclipse

Including JUnit, Ann and Javadoc

eclipse

Agenda

- Basics of the Java Development Tools environment
- Creating and running a program
- Debugging a program
- Automating testing with JUnit
- Using Ant and Javadoc

eclipse.org/
ibm.com/university/

The Java Development Tools

aka the JDT

eclipse

JDT

- A set of tools for writing, compiling, testing, and debugging Java code.
  - **Note: Compiling happens automatically whenever you save your code**.
    It's not a separate step.

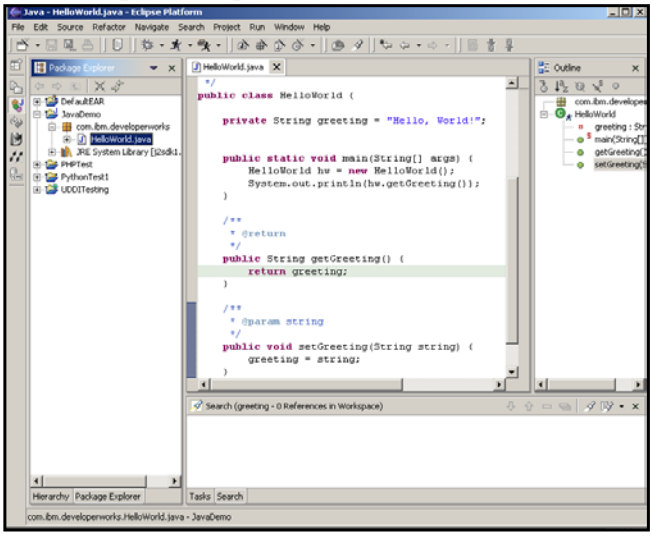
JDT perspectives

- The most useful perspectives for Java development are Java and Debug.
  - There are also the Java Browsing and Java Type Hierarchy perspectives.

Java perspective

- The Java perspective contains, among other things, an editor, an outline of your code and the package explorer.

The Java perspective

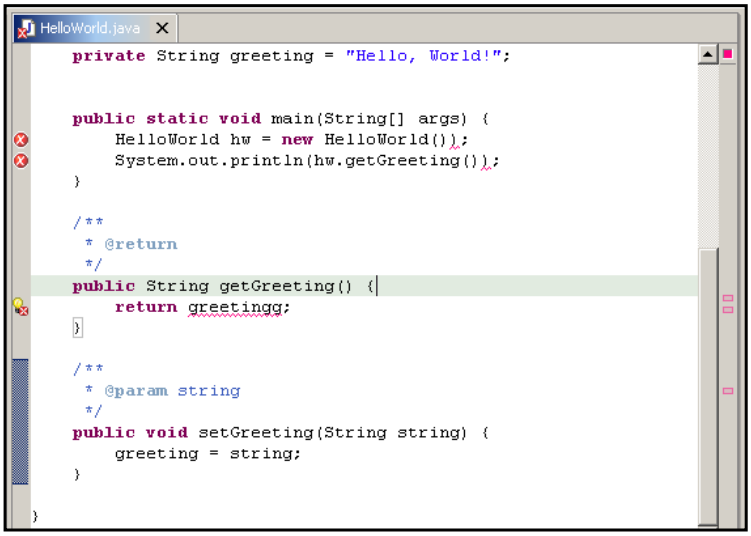



```
public class HelloWorld {

    private String greeting = "Hello, World!";

    public static void main(String[] args) {
        HelloWorld hw = new HelloWorld();
        System.out.println(hw.getGreeting());
    }

    /**
     * @return
     */
    public String getGreeting() {
        return greeting;
    }

    /**
     * @param string
     */
    public void setGreeting(String string) {
        greeting = string;
    }
```

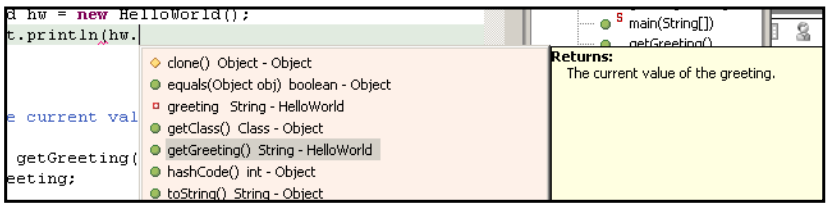

The Java editor

- As you'd expect from a world-class IDE, Eclipse has a color-coded Java editor.
- As you type, it automatically highlights the Java syntax and indents your code.
- If there are errors, they're indicated when you save the file (if not before).

The Java editor



Code assist



- If you type Ctrl+Space, Eclipse shows you the relevant method signatures and the Javadoc for each.
- This works for code you write as well as the standard Java libraries.

Other great features

- Generate getters and setters automatically
- "Quick Fix" certain problems
- Automatically fix import statements
- Refactoring code
  - Rename classes, methods, fields
  - Create an interface from a class
  - Move classes, methods, fields

eclipse.org/
ibm.com/university/

IBM

Creating and running code

eclipse

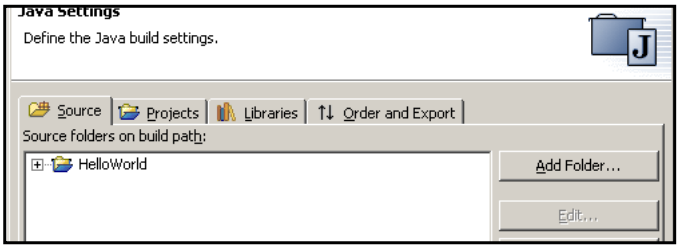
Creating and running code

- It's a short process:
  1. Create a Java project
  2. Create a Java package
  3. Create a Java class in that package
  4. Set up a run configuration
  5. Run your code
- Step 4 is the one that's most confusing to newcomers.

Creating a Java project

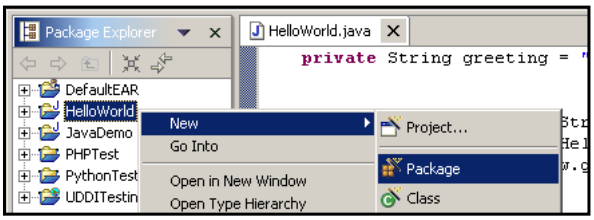
- Start with **File→New→ Project…**
- Choose Java Project, give it a name and click Finish.

Creating a Java Project



- If you click Next after you give your project a name, you'll see other options. You can use these to set the classpath of your project, among other things.

Creating a Java package

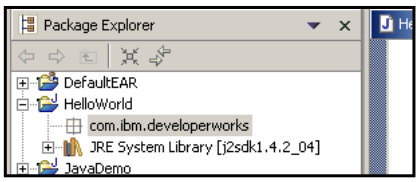


- To create a Java package, right-click on your new project in the Package Explorer, then choose **New→Package…**

Creating a Java package

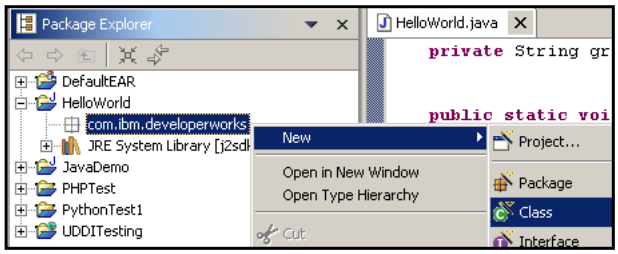
- Enter a name for your package.
- If you break Java style rules (maybe your package begins with an uppercase letter), Eclipse reminds you.

Creating a Java package



- Your new package appears in the Package Explorer beneath your project.

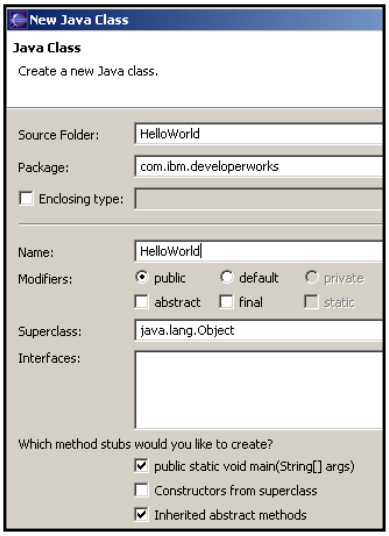
Creating a Java class



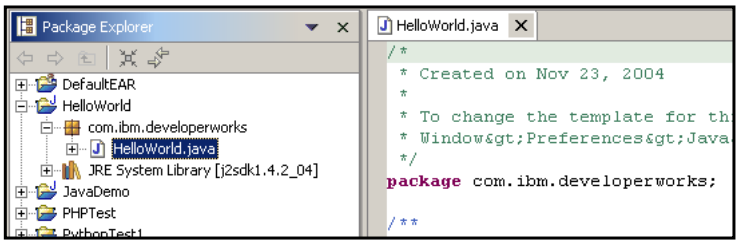
- To create a Java class, right-click on your new package in the Package Explorer, then choose **New→Class**.

Creating a Java class

- Enter a name for your class.
- Eclipse reminds you of style rules here as well.
- You can set the details of your class, including its superclasses, visibility and interfaces.



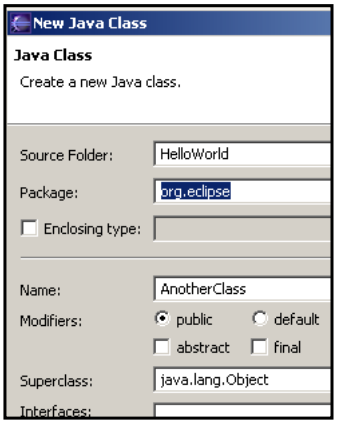
Creating a Java class



- Your new class appears in the Package Explorer beneath your package.
- Eclipse also opens the source file for your class in the Java editor.

A shortcut

- You can create a new package **and** a new class at the same time.
- Simply create a new class and enter a new package name in the wizard.



Creating a run configuration

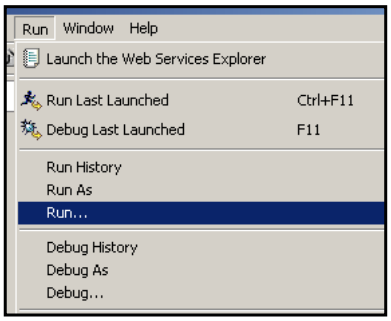
- If you were running a Java program outside of Eclipse, you'd have to make sure the appropriate JAR files were on your classpath.
- To run a Java program inside Eclipse, you have to create a run configuration that knows where to find the classes you need.
  - Think of it like a glorified classpath.

Creating a run configuration

- Running code in Eclipse is different from other IDEs you might have used because there is no separate compile step. You just run your code.
  - This also means the Java tools find any compile errors whenever you save your code (maybe even earlier).

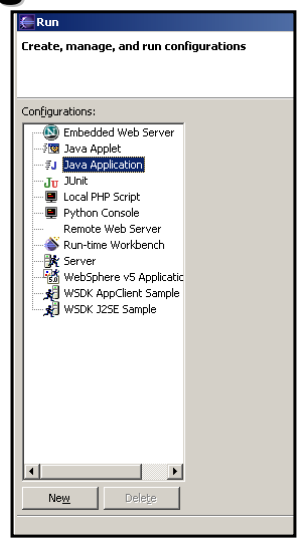
Creating a run configuration

- Select your project in the Package Explorer, then choose **Run→Run…**

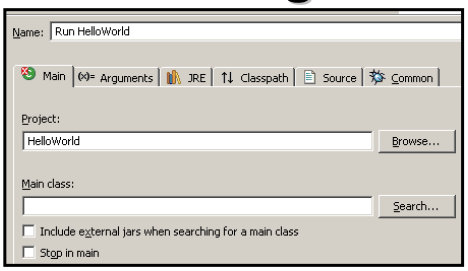


Creating a run configuration

- In the Run dialog, choose Java application in the list and click New.



Creating a run configuration



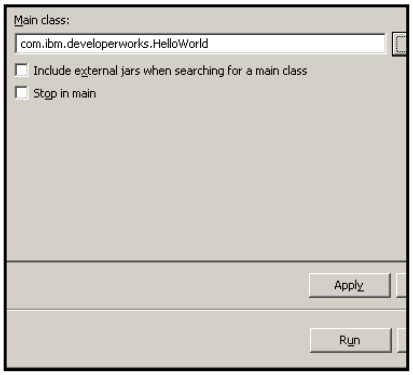
- Give your configuration a name.
- Eclipse complains because it hasn't found a class with a **`main()`** method. Click the Search… button to find it.
  - Eclipse V3.x is more clever.

Creating a run configuration

- Eclipse finds the class with a **`main()`** method; click OK to accept it.

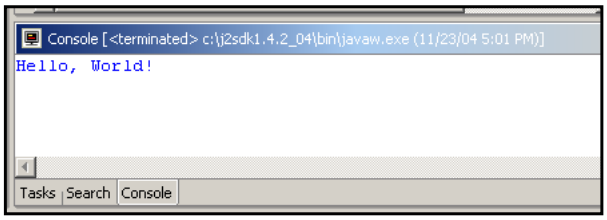
Running your code

- Now that you've filled in all the parameters for the run configuration, you're ready to go.
- Click Run to start your program.

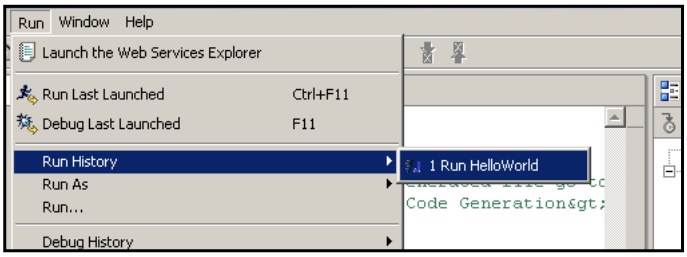


Running your code

- Because this is a console application (it uses **`System.out.println`**), you'll see the output in the Console view.
- If the Console doesn't appear, you can open it through **Window→Show View…**



Re-running your code



- Once you've run your code, a reference to your code appears in the Run menu.
  - It's also in the Debug menu.
- Run Last Launched (Ctrl+F11) does the same thing.

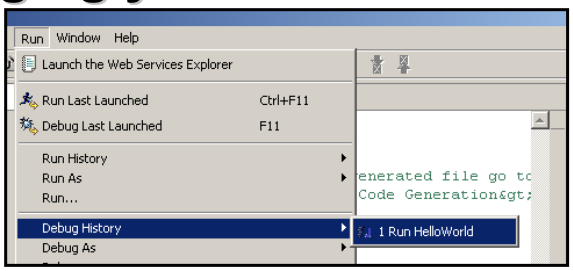
eclipse.org/
ibm.com/university/

IBM

Debugging your code

eclipse

Debugging your code

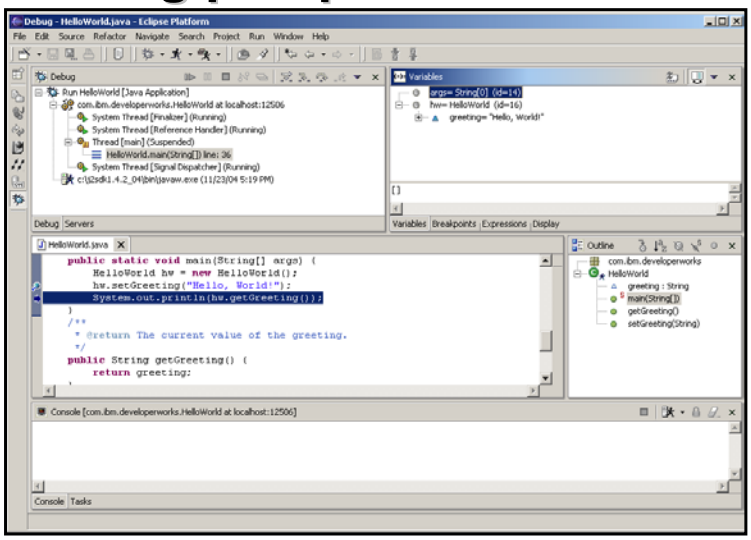


- As you would expect from a world-class IDE, Eclipse has a great debugger built in.
  - All the usual debug functions are supported.
- You use the same run configuration, but choose Debug instead of Run.

Java Spider

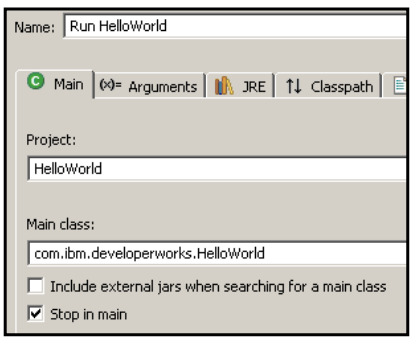
- The Java Spider navigator is available at **`www.javaspider.org`**.

The debug perspective

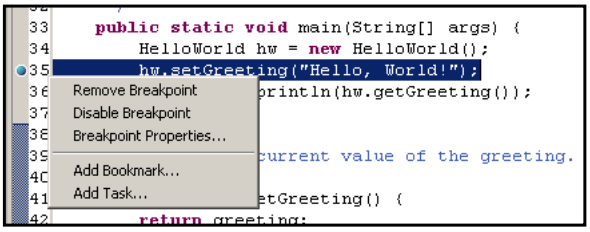


Stopping in `main()`

- The HelloWorld program has a **`main()`** method; if you want to debug it, you need to stop the debugger there.
- Choose "Stop in main" in the run configuration.



Setting breakpoints

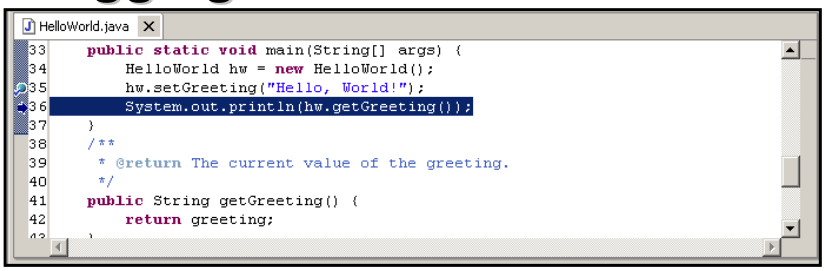


- The easiest way to set a breakpoint is to double-click in the gray area on the left-hand side of the editor.
  - A blue dot means the breakpoint is set.
  - Double-click again to clear it.

Setting breakpoints

- You can also set breakpoints on methods, Java exceptions and fields.
  - Break whenever execution enters or leaves a method
  - Break whenever an exception is thrown
  - For fields, break when a field is accessed, changed, or accessed a certain number of times.

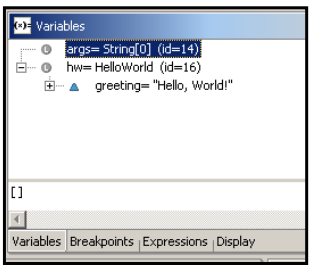
Debugging

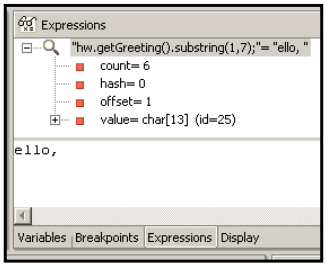


- When you're debugging your code, the editor shows you where things are.
- In this view, there's a breakpoint on line 35, and the system is stopped on line 36.
  - You can turn line numbers on in Preferences.

Monitoring variables & expressions

- Once your code is running in debug mode, you can check (and change, if you want) the value of variables and expressions.
- I added the expression `getGreeting().substring(1,7)`





eclipse.org/
ibm.com/university/

Automated testing with JUnit

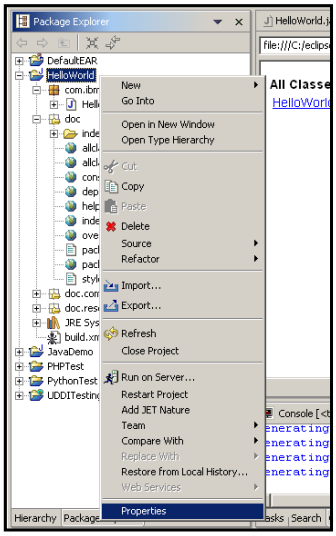
eclipse

Automated testing with JUnit

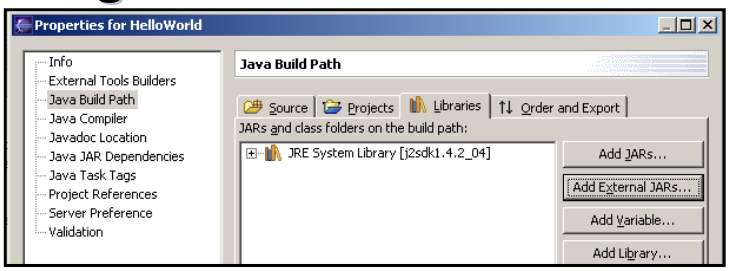
- JUnit was created by programming legends Kent Beck and Erich Gamma.
- It makes it easy to implement Test-Driven Development (TDD).
- Eclipse ships with JUnit support built in.

Creating test cases

- Before you can work with JUnit, you have to add **`junit.jar`** to your project's build path (classpath).
- Right-click your project name and choose Properties from the menu.



Creating test cases



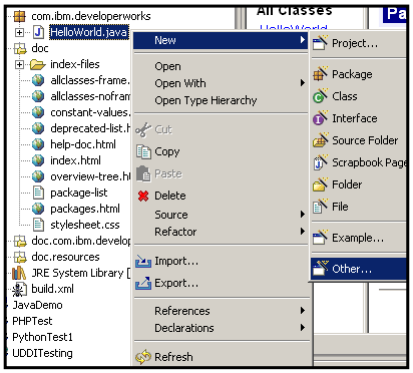
- In the Properties dialog, select Java Build Path on the left and click the Libraries tab on the right.
- Click the Add External JARs… button.

Creating test cases

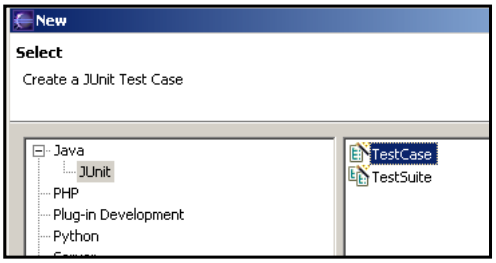
- Browse your system to find **`junit.jar`**. (It's most likely in the **`eclipse\plugins\org.junit_3.8.1`** directory.)
- Select the JAR file and click OK to save your new build path.

Creating test cases

- With your build path set, you can start creating a new JUnit test case.
- Right-click on a Java file and choose **New→ Other…**

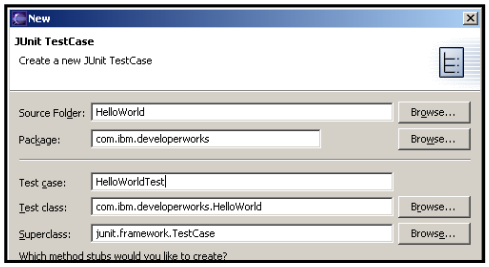


Creating test cases



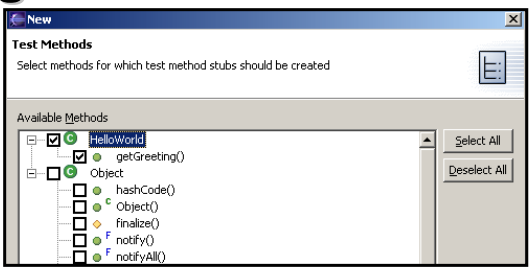
- Select Java/JUnit on the left and TestCase on the right, then click Next.

Creating test cases



- When you create a JUnit test case, you name the test case (it's a Java class) as well as the Java class tested by the test case.

Creating test cases



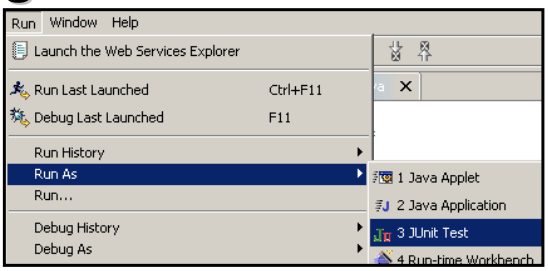
- Eclipse gives you a list of all the public methods in your class and its superclasses. You decide which ones should be part of the JUnit test class.

Creating test cases

- In this example, we ask Eclipse to generate a JUnit **`TestCase`** for the **`getGreeting()`** method.
- The complete **`testGetGreeting()`** method is:

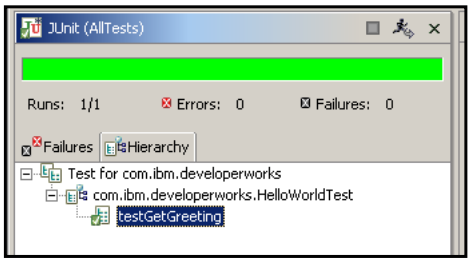
```
public void testGetGreeting() {
  HelloWorld hw = new HelloWorld();
  assertEquals("Hello, World!",
               hw.getGreeting());
}
```


Running test cases



- Our test case is the Java class **`TestHelloWorld`**.
- To run the class, select the test class in the Package Explorer, then choose **Run As→JUnit Test**.

Running test cases



- The results of running your test case appear in the JUnit view.
  - Green is good…

Test suites

- You can also create and run JUnit **`TestSuite`**s.
  - A **`TestSuite`** is an ordered collection of **`TestCase`**s.

Using JUnit

- You define more **`TestCase`**s and **`TestSuite`**s as your project progresses.
- You can run the JUnit tests whenever you want to make sure your changes haven't broken your code.

eclipse.org/
ibm.com/university/

Using Ant and Javadoc

eclipse

Using Ant

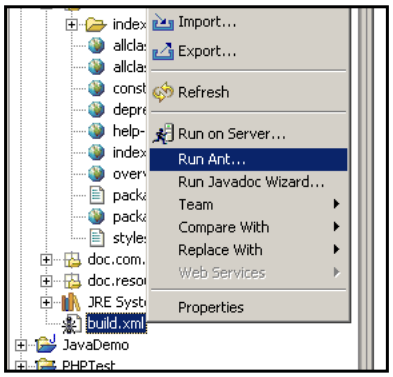
- Ant (**`ant.apache.org`**) is an XML- and Java-based build tool.
  - Designed to have the same functionality as **`make`** without its quirks
    - You don't need a tab character at the start of each line, for example.
  - You can extend Ant to do other tasks if you want.

Using Ant

- An Ant build file (named **`build.xml`** by default) can define a number of targets.
- You can define which target gets built from the command line (or the Eclipse equivalent), or let Ant figure out which one should be created.

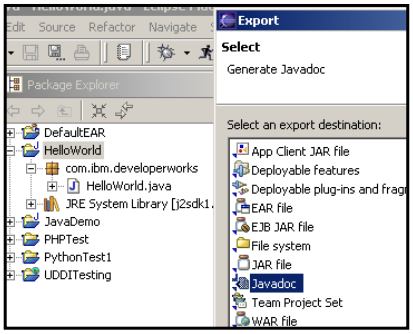
Using Ant

- Once you've created your **`build.xml`** file (or whatever you choose to call it), you can right-click on it and choose **Run Ant...**

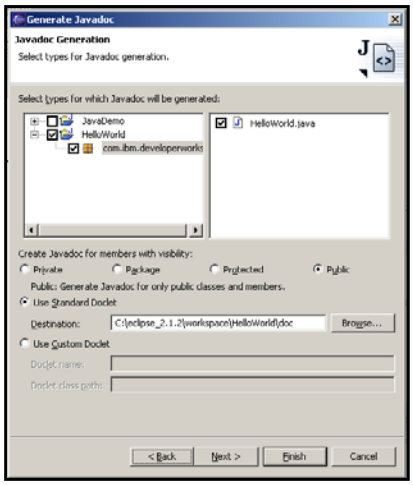


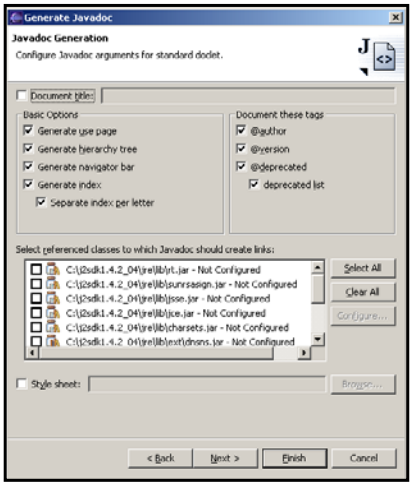
Using Javadoc

- You can export your project to Javadoc.
- When you do this, Eclipse runs **`javadoc`** against your code and exports the generated files to the directory you choose.



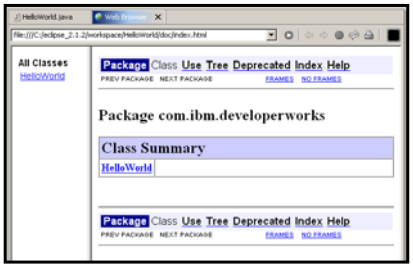
Using Javadoc





Using Javadoc

- The generated documentation is put in the **`docs`** folder of your project by default.



eclipse.org/
ibm.com/university/

IBM

Review

eclipse

Review

- We've covered (although *very* quickly) the Java development functions in Eclipse, including:
  - Various automatic editing features
  - How to create and run Java code
  - Using the Eclipse debugger
  - Automating testing with JUnit
  - Using **`ant`** and **`javadoc`** inside Eclipse