DrJava User Documentation

DrJava User Documentation

Table of Contents

Chapter 1. Introduction

DrJava is a programming environment for Java, primarily intended to help students focus more on program design than on the features of a complicated development environment. DrJava also provides many advanced features for experienced developers. These features center around DrJava's Interactions Pane, which is a "read-eval-print loop" that allows users to easily develop, test, and debug Java programs in an interactive, incremental manner.

Home Page: http://drjava.org[1]
Original Paper: http://drjava.org/papers/drjava-paper.shtml[2]

Notes

1. http://drjava.org
2. http://drjava.org/papers/drjava-paper.shtml

Chapter 2. Getting Started

This chapter describes the basics for how to start using DrJava, including where to get the program and how to run it.

Philosophy

The general idea behind DrJava is to provide powerful development tools that are as easy to use as possible. For this reason, we try to make DrJava easy to run and easy to understand, through a simple user interface with few panes and a legible toolbar. Meanwhile, we want to help novice users become comfortable with writing Java code by allowing them to quickly evaluate expressions in DrJava's Interactions Pane. All of our powerful features try to build on this simple and interactive interface.

The rest of this chapter will walk you through downloading and running DrJava, but if you have the DrJava `.jar` file, you can just double-click it to get started.

Downloading DrJava

You can download the newest releases of DrJava as a `.jar` file from our home page, http://drjava.org[1], or directly from our Project Filelist[2] page on SourceForge.

Stable and Development Releases

We make a distinction between Stable and Development releases of DrJava. All releases must pass our rigorous suite of unit tests and should be safe to use, but we have found that a period of beta-testing can be helpful for finding additional bugs. Any large new features go through a beta-testing period before being included in Stable releases, ensuring these releases are safe for all users. Our Development releases contain newer features that are under development. We believe these releases are also ready to use, but they have not been widely beta-tested, so some users may prefer to use only Stable releases.

Running DrJava

On many platforms, DrJava can be started by simply double-clicking on the `.jar` file you downloaded. DrJava can also be started from a command prompt, where you can optionally give it a list of source files to open at startup:

java -jar drjava-DATE-TIME.jar [-config [CONFIG_FILE]] [filename.java...]

Replace DATE-TIME with the appropriate value for your version of DrJava. The "config" argument is optional and allows you to specify a custom configuration file, rather than the `.drjava` file stored in your home directory.

Running DrJava on Mac OS X

If you are using Mac OS X, you can download DrJava as an Application from our website. Download the `drjava-DATE-TIME-osx.tar.gz` file and decompress it. You can then copy the DrJava icon into your Applications folder or keep it on your Dock.

System Requirements

DrJava requires Java 2, version 1.3 or later. Note that you will need to have the JDK (not the JRE) installed if you wish to use the compiler or debugger within DrJava.

We recommend using Sun's JDK 1.4.2 or later (from http://java.sun.com[3]) for Solaris, Linux, and Windows. Other users should use the Java virtual machine that comes with their operating system (including Mac OS X).

License

DrJava is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

DrJava is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

The full text of the license is available from www.gnu.org[4].

DrJava incorporates DynamicJava (available at http://koala.ilog.fr/djava[5]), which is licensed under these terms:

DynamicJava - Copyright (c) 1999 Dyade

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL DYADE BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE

Except as contained in this notice, the name of Dyade shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from Dyade.

Notes

1. http://drjava.org
2. http://sourceforge.net/project/showfiles.php?group_id=44253
3. http://java.sun.com
4. http://www.gnu.org

5. http://koala.ilog.fr/djava

Chapter 3. Editing Programs

DrJava's core component is an editor for writing Java source code. Like most text editors, it supports a wide range of editing features such as "Find/Replace" and "Goto Line", while also providing more advanced features like syntax coloring, automatic indentation, and brace matching.

Definitions Pane

The Definitions Pane is the main window of DrJava, displaying the currently active source file. As you edit files in this window, DrJava helps out with several useful features.

Syntax Coloring

DrJava colors special types of text differently to help make the structure of the program more apparent. Comments appear in green, while Java keywords and types appear in blue. Strings are colored red and characters are colored magenta, with all other text colored black. These colors are all configurable (see Configuring DrJava).

One notable difference between syntax coloring in DrJava and other common editors (such as Emacs) is that DrJava uses *fully correct* coloring as the document is edited. For example, simply typing the beginning of a block comment ("/*") will immediately update the coloring of the entire document, unlike some other editors which will only update the color of a line when that line is edited. Having an accurate view of the program is an important aspect of understanding its structure.

Automatic Indentation

The key to indenting code in DrJava is the Tab key. Rather than simply inserting a tab or spaces, pressing Tab properly indents the current line (or selected text) using common coding conventions. As you type multiple lines of code into the Definitions Pane, DrJava automatically indents each line using the same technique. By default, two spaces are used for each indentation level, although this can be configured in the Preferences window. (In DrJava, code is always indented with spaces, and never with actual tab characters.)

Brace Matching

To help you match open and close braces, DrJava highlights the region enclosed by a pair of braces. If you place the cursor immediately after a close brace, parenthesis, or bracket, all text between that character and the corresponding open brace is highlighted in another color. Like syntax coloring, brace matching is also done in a *fully correct* manner, updated with each keystroke.

Commenting / Uncommenting

To help you easily write multi-line comments, DrJava automatically adds spaces and an asterisk on each new line. In addition, there is an option in the "Miscellaneous" section of the Preferences window that will tell DrJava to automatically close multi-line comments for you. Commands in the Edit menu are also available to comment out or uncomment a block of selected code using winged comments ("//"). The key bindings for these commands default to Ctrl+Slash and Ctrl+Shift+Slash respectively.

Commenting out a block of code will place "//" markers at the start of each line in the block, preserving the indentation of the code.

Context Menu

The Definitions Pane has a context menu, which can be used by right-clicking in the pane. (Mac users should use Ctrl+Click or Option+Click.) This menu provides shortcuts to useful features such as cut, copy, and paste, as well as indenting, commenting, and setting breakpoints in the debugger.

Multiple Documents

Most Java programs have several closely related source files, so DrJava supports having multiple documents open at the same time. A list of all of the names of the open documents appears in a pane to the left of the Definitions Pane, listing files in the order in which they were opened. You can view and edit a particular document by selecting it in the list, or by using the Previous and Next Document commands in the Edit Menu. (These commands have keyboard shortcuts as well: Ctrl+Comma and Ctrl+Period.) Only the class name of a source file is shown in the list and in the title bar, but the full file name of the current document is always displayed in the status bar at the bottom of the window.

Context Menu

The Document List also has a context menu, which can be used by right-clicking on any document in the list. (Mac users should use Ctrl+Click or Option+Click.) This menu provides shortcuts to document-related commands, such as saving, reverting, printing, compiling, testing, running Javadoc, and calling the main method.

Source Navigation

DrJava has many simple features to help you edit and navigate source files.

Find/Replace

DrJava has a Find and Replace utility, which is conveniently displayed as one of the tabs at the bottom of the window, rather than as a dialog blocking part of the window. The tab is first displayed when you Find/Replace from the Edit Menu (or using the keyboard shortcut, Ctrl+F), and it can be closed by clicking on the "X" button in the upper right corner of the tab (or by hitting the Escape key). To search for a term, type it in the Find text field and click "Find Next" (or press Enter). To replace the term with another, type the new term in the Replace text field, find an occurrence using "Find Next", and then click "Replace". The "Replace/Find Next" and "Replace All" buttons help to speed up this process. There are also three checkboxes to customize each search: "Match Case", "Search Backwards", and "Search All Documents". Unchecking the first box will return case-insensitive results, checking the second box will reverse the direction of searching, and checking the third box will tell DrJava to search through all of the open documents in sequence. Note that if "Search All Documents" is enabled, the search will not wrap to the beginning of the current document until all other documents have first been searched.

Goto Line

Selecting "Goto Line" from the Edit Menu (or hitting Ctrl+G) will display a dialog allowing you to scroll to a particular line number.

Line Numbering

DrJava displays the cursor's current line number and column number on the right side of the status bar at the bottom of the window. The line number is shown on the left and starts at 1, and the column number is shown on the right and starts at 0.

All line numbers can also be displayed in the left margin of the Definitions Pane, using the "Show All Line Numbers" option in the "Display Options" section of the Preferences window. The line number font can be changed in the "Fonts" section. (See Configuring DrJava.)

Chapter 4. Interactions Pane

One of the key distinguishing features of DrJava is its Interactions Pane, which allows you to enter and evaluate Java statements and expressions on the fly. This is remarkably useful for beginning students, who no longer have to learn to write main methods, recompile, and run programs from a command line simply to test how a new class or method behaves. From a teaching standpoint, the Interactions Pane is a very easy way to help students learn to write Java without having to explain the full meaning of syntax like "public static void main", and it also provides an ideal way to perform demonstrations in class. The Interactions Pane can also be used to experiment with your own programs or new libraries, or even to create graphical user interfaces interactively.

How to Use

The Interactions Pane supports the execution of any valid Java statements as well as the evaluation of Java expressions. Simply define variables and call methods as you would in an ordinary method, or even define new classes and methods and call them interactively. In general, any statement or expression ending without a semicolon will display its result in the Interactions Pane, while those ending with a semicolon will complete without displaying a result. Result objects are displayed using the object's toString() method. Any system output will be displayed in the Interactions Pane in green (as well as in the Console tab), while system errors will be displayed in red by default. Any system input will cause a box to be inserted in the Interactions Pane where you can type what you want System.in to read. This text will be colored purple. These colors can be modified in the "Colors" section in the Preferences window.

Here is a simple interactions session, to demonstrate how the Interactions Pane can be used to experiment with objects or show GUI components.

```
Welcome to DrJava.
> String s = "Hello World";
> s
"Hello World"
> s.length()
11
> import javax.swing.*;
> JFrame frame = new JFrame("My JFrame");
> frame.show();
>
```

Resetting the Interactions Pane

You can reset the Interactions Pane if you wish to start from scratch or if a method call hangs and does not return. Resetting removes any variables from scope and terminates any methods running in the Interactions Pane. To do this, simply choose the "Reset Interactions" command from the Tools menu. This will also reset the Debugger and any JUnit tests that are currently running.

Running the Main Method

For convenience, DrJava supports calling the main method of a class in the Interactions Pane by simply entering "java" followed by the class name and any arguments.

For example, to run MyClass with args "arg1" and "arg2", type the following into the Interactions Pane:

```
java MyClass arg1 arg2
```

Note that this feature does not support passing special flags or arguments to the JVM itself, as is supported on the command line.

Another shortcut for this feature is the "Run Document's Main Method" command, which can be found in both the Tools menu and the context menu of the document list. This command will simply insert the appropriate `"java MyClass"` text into the Interactions Pane to run the current class's main method.

Keyboard Shortcuts

Many actions in the Interactions Pane have keyboard shortcuts for ease of use. Use the Up and Down arrow keys to scroll through a history of the previously entered commands, or Ctrl+B to clear the current command. You can also use Shift+Enter to insert newlines into statements in the Interactions Pane.

Setting the Classpath

To interact with any class within the Interactions Pane, it must be included on the Interactions Classpath, which can include more than the user's own classpath. Any class which is opened in the Definitions Pane of DrJava is automatically added to this classpath, but additional classes and directories can be added using the "Extra Classpath" configuration option. (See Configuring DrJava.) The current classpath of the Interactions Pane can be viewed at any time by selecting "View Interactions Classpath" from the context menu.

Saving the Interactions History

You can save all of your past interactions to a file at any time, using the "Save Interactions History" command in the Tools and popup menus. This command gives you the option to edit any part of the history before saving it, through a separate window that supports editing. By default, up to 500 of the most recent Interaction commands are kept in this history, though this number is configurable. Histories are saved in files with a `.hist` extension, and they can be later executed in the Interactions Pane with the "Execute Interactions History" command in the Tools menu. Saving and executing histories can be particularly useful for initial setup of an often repeated task, such as importing several packages and initializing frequently used variables. To help manage the history, a "Clear Interactions History" command is also provided in the Tools menu.

Loading a History File as a Script

You can load a history file as a script that can be executed one line at a time, using the "Load Interactions History as Script" command in the Tools and popup menus. A panel will appear on the right side of the Interactions Pane with buttons that allow you to display the previous interaction, execute the current interaction, display the next interaction, and close the panel. This feature is useful during presentations because you can step through a series of interactions that has been prepared in advance, allowing the audience to see the result of each interaction.

Lifting Interactions to Definitions

One common use of the Interactions Pane is to test a line of code intended for a program, even before it is written in the program itself. For example, this can be the case when experimenting with method calls to determine their results. In this situation, it is convenient to copy a working line from the Interactions Pane into the Definitions Pane. This can be done quickly with the "Lift Current Interaction to Definitions" command in the Tools menu, which simply copies the text at the current prompt and pastes it at the cursor position in the Definitions Pane.

Context Menu

The Interactions Pane has a context menu, which can be used by right-clicking in the pane. (Mac users should use Ctrl+Click or Option+Click.) This menu provides shortcuts to useful commands for the Interactions Pane, including cut, copy, and paste, as well as resetting the Interactions Pane, executing, loading, and saving history files, viewing the current classpath, and copying the current interaction to the Definitions Pane.

Chapter 5. Compiling Programs

Java compilers check your programs for errors and translate them to class files which can be used. Any time you change the source file for a class, it must be compiled before it can be used. To do this in DrJava, you can simply click on the "Compile All" button on the toolbar to compile any open documents. Any resulting errors will be highlighted in the document.

Compiling Files

To compile the documents you have open in DrJava, click on the "Compile All" button on the toolbar, or choose either "Compile All" or "Compile Current Document" from the Tools menu. Before the compilation can begin, all open files must be saved, even if only the current document is being compiled. This is necessary because one file can depend on other files, and it is important that no files have been modified when errors are displayed. (Otherwise, an error could be highlighted on a line which has changed.) Once a compilation completes successfully, the Interactions Pane is reset so that the new class files can be used. The output on the Console Tab is also reset to begin the new session, unless the "Clear Console After Interactions Reset" option in the "Miscellaneous" section of the Preferences is unchecked.

Viewing Compiler Errors

If the compiler finds any errors in your program, DrJava displays them in the Compiler Output tab at the bottom of the window. A summary of each error is displayed in the list, including the file name and line number. You can click on any error to highlight the corresponding line in the document itself. (Note that a file will be opened automatically if it contains errors detected by the compiler.) Similarly, if the cursor is moved to a line of code that contains an error while the Compiler Output tab is displayed, that line and the corresponding error message are highlighted. You can disable highlighting compiler errors in the source by unchecking the "Highlight Source" checkbox on the Compiler Output tab, or by closing the Compiler Output tab.

Selecting a Compiler

DrJava supports the use of different Java compilers, such as the traditional "Javac" compiler supplied with the JDK or Sun's JSR-14 experimental compiler (which supports generic types). DrJava will attempt to locate the your Java compiler on startup by searching for standard installation directories, but sometimes it is unable to find a compiler. In this case, it will prompt you to specify the location of a compiler, or allow you to continue using DrJava without any compiler. Note that the location of the compiler can be specified in the Preferences (see Configuring DrJava). If more than one compiler is specified, the active compiler can be selected from a menu on the Compiler Output tab itself. You may notice that the same compiler may appear multiple times in the menu. This happens because DrJava looks for compilers in three different places: on the user classpath, the user specified location in Preferences, and in the usual locations that the compiler can be found (the default installation directory, where java is located, etc.).

Chapter 6. Testing using JUnit

While compilers can look for structural problems in a program, they cannot tell whether the results of a program or method are correct. Instead, all developers test their programs to ensure that they behave as expected. This can be as simple as calling a method in the Interactions Pane to view its results, but this technique requires you to think about the answers you expect every time you run any tests. A much better solution is to give the tests the answers you expect, and let the tests themselves do all the work.

Thankfully, a technique known as unit testing makes this quite easy. You write many small tests that create your objects and assert that they behave the way you expect in different situations. A unit test framework known as JUnit (http://www.junit.org[1]) automates the process of running these tests, letting you quickly see whether your program returns the results you expect.

DrJava makes the process of running unit tests very simple by providing support for JUnit. Once you have written a JUnit test class (as described in the next section), you can simply choose the "Test Current Document" command from the Tools menu to run the tests and view the results. The name of the tests being run will be shown in the Test Output tab, with each test method turning green if it completes successfully and red if it fails. Any failures will be displayed after the list of methods in the same way as the compiler errors. A progress bar will keep you updated on how many tests have been run.

Also, clicking the "Test" button on the toolbar or choosing "Test All Documents" from the Tools menu will run JUnit on any open testcases, making running multiple test files very simple.

Writing Unit Tests with JUnit

With the JUnit framework, unit tests are any public classes that extend the `junit.framework.TestCase` class, and that have any number of methods with names beginning with the word "`test`". JUnit provides methods to easily assert things about your own classes, as well as the ability to run a group of tests.

The requirements for writing unit test classes are described below, with an example provided in the next section. In general, though, the intent is for you to create instances of your classes in the test methods, get results from any methods that you call, and assert that those results match your expectations. If these assertions pass, then your program has behaved correctly and your tests have succeeded.

Writing a Test Case

To use DrJava's Test command on a document, you must use the programming conventions outlined below. You can also choose the "New JUnit Test Case" command from the File menu to automatically generate a template with these conventions.

- At the top of the file, include:

  ```
  import junit.framework.TestCase;
  ```

- The main class of the file must:
  - be `public`

  - `extend TestCase`

- Methods of this class to be run automatically when the Test command is invoked must:
  - be `public` and *not* `static`
  - return `void`
  - take no arguments
  - have a name beginning with "`test`"

- Test methods in this class can call any of the following methods (among others):
  - `void assertTrue(String, boolean)`

    which issues an error report with the given string if the boolean is false.

  - `void assertEquals(String, int, int)`

    which issues an error report with the given string if the two integers are not equal. The first int is the expected value, and the second int is the actual (tested) value. Note that this method can also be called using any primitives or with Objects, using their `equals()` methods for comparison.

  - `void fail(String)`

    which immediately causes the test to fail, issuing an error report with the given string.

  Test methods are permitted to throw any type of exception, as long as it is declared in the "`throws`" clause of the method contract. If an exception is thrown, the test fails immediately.

- If there is any common setup work to be done before running each test (such as initializing instance variables), do it in the body of a method with the following contract:

  `protected void setUp()`

  This method is automatically run before any tests in the class. (Similarly, you can write a `protected void tearDown()` method to be called after each test.)

- If you would rather control which methods are called when running the tests (rather than using all methods starting with "`test`"), you can write a method to create a test suite. This method should be of the form:

```
public static Test suite() {
  TestSuite suite = new TestSuite();
  suite.addTest(new <classname>("<methodname>"));
  ...
  return suite;
}
```

It is then also necessary to import `TestSuite` and `Test` from `junit.framework`. There is also a version of the `addTest` method that takes a `Test`, so test suites can be composed.

A simple example of a TestCase class is given in the next section. There are many other ways to use JUnit, as well. See the JUnit Cookbook at http://junit.sourceforge.net/doc/cookbook/cookbook.htm[2] for more examples and information.

Simple Test Example

Suppose you are writing a Calculator class that can perform simple operations on pairs of integers. Before you even write the class, take a moment to write a few tests for it, as shown below. (By writing tests early, you start thinking about which cases might cause problems.) Then write the Calculator class, compile both classes, and run the tests to see if they pass. If they do, write a few more test methods to check other cases that you have realized are important. In this way, you can build up programs with a great deal of confidence.

```
import junit.framework.TestCase;
public class CalculatorTest extends TestCase {

  public void testAddition() {
    Calculator calc = new Calculator();
    // 3 + 4 = 7
    int expected = 7;
    int actual = calc.add(3, 4);
    assertEquals("adding 3 and 4", expected, actual);
  }

  public void testDivision() {
    Calculator calc = new Calculator();
    // Divide by zero shouldn't work
    try {
      calc.divide(2, 0);
      fail("Should have thrown an exception!");
    }
    catch (ArithmeticException e) {
      // Good, that's what we expect
    }
  }

}
```

Viewing Test Failures

If one or more test methods in a JUnit test class fails, each one is displayed in the Test Output tab at the bottom of the window. This list of failures is similar to the list of compiler errors, in that a summary of the error is given in the tab, and clicking

on it highlights the corresponding line in the file (as long as the "Highlight Source" checkbox is checked). Note that DrJava displays a warning message if the test class has been modified since the last time it was compiled, since the changes will not be reflected in the behavior of the test. Closing the Test Output tab resets the current set of JUnit failures.

Aborting Tests

If a suite of tests takes a long time or goes into an infinite loop, you can abort the tests by choosing the "Reset Interactions" command from the Tools menu. An error will be displayed in the Test Output tab showing that the tests were aborted.

Viewing the Stack Trace

When a JUnit test fails or throws an exception, it is sometimes helpful to view the entire stack trace when diagnosing the problem. To view the stack trace for any test failure, right click on the failure in the Test Output tab and select "Show Stack Trace."

Notes

1. http://www.junit.org
2. http://junit.sourceforge.net/doc/cookbook/cookbook.htm

Chapter 7. Debugger

DrJava provides advanced tools for debugging your programs in the Interactions Pane. You can set breakpoints in source files in the Definitions Pane, call methods that stop at the breakpoints in the Interactions Pane, and then interact with programs while they are suspended at breakpoints. Once a breakpoint is reached, users can interact with any variables, fields, or methods that are in scope in the suspended method. Users can also resume the method call, or step through it a line at a time. Finally, the values of local variables and fields can be watched in a table as the method call progresses.

Using the Debugger

To use DrJava's debugger, select the "Debug Mode" command from the Debugger menu. An informational panel will be displayed between the Definitions Pane and the Interactions Pane, and several menu items in the Debugger menu will become enabled.

A Note on Modifying Files

When using the debugger, it is essential to remember that any modifications to source files will not be reflected in the behavior of the Interactions Pane or the debugger until the classes are recompiled. Changing a source file while the debugger is running is not recommended, since lines which are highlighted by the debugger may no longer correspond to the lines in the running class file. To help notify you of this danger, DrJava displays a warning message in the Debug Panel if the current document is out of sync with its class file.

Because the debugger depends on the classes used in the Interactions Pane, the debugger is automatically reset each time any files are compiled, or when the Interactions Pane is reset.

Breakpoints

Once DrJava is in Debug Mode, you can set a breakpoint on almost any line of code in a source file in the Definitions Pane, using either the "Toggle Breakpoint on Current Line" command in the Debugger menu or the "Toggle Breakpoint" command on the context (right-click) menu in the Definitions Pane. When a breakpoint is set, the line will be highlighted in red and an entry will appear in the Breakpoints tab of the Debug Panel. A breakpoint is reached when a method is called in the Interactions Pane and the control flow reaches a line of code where a breakpoint has been set. When this happens, DrJava highlights the line in bright blue and prints a message to the Interactions Pane. DrJava then displays a new prompt in the Interactions Pane, allowing you to interact with the suspended program until it is resumed (see Interactions at a Breakpoint).

When setting breakpoints, it is important to remember that only lines with actual executable code can be used. Blank lines and comments will never trigger a breakpoint, even if the line is highlighted in red. (Note that we do not yet support breakpoints on method contracts either, although this will be supported in a later version of DrJava.)

Debugging JUnit Tests

DrJava will also stop at breakpoints during JUnit tests. Simply set a breakpoint on a line of a test method or in a method called by a test, and then choose the "Test Using JUnit" command from the Tools menu. When control flow reaches the breakpoint, the test will be suspended.

Interactions at a Breakpoint

When DrJava reaches a breakpoint during a method call, it prints a new prompt to the Interactions Pane. This new interpreter has the context of the program which has been suspended, including all variables, fields (even "`this`"), and methods that are in scope in the suspended method. (The prompt text itself contains the name of the thread which has been suspended. In most cases, this name will include the text being interpreted.) You can type the name of any variable or field to view its value or assign it a new value, or you can call any method in scope to observe its behavior. Existing lines of code in the program cannot be changed or skipped, however, and the "`return`" keyword is not available. Any changes you make to variables or fields will be reflected in the program when it resumes execution, either by stepping or resuming.

Stepping and Resuming

After DrJava reaches a breakpoint, the method being called is suspended, and several commands in the Debug Menu and Debug Panel become available. Choosing "Resume" allows the current method to finish execution, at least until another breakpoint is reached. If any other threads are suspended when you resume, the Interactions Pane will switch to the most recently suspended thread. Otherwise, the original ("top level") prompt is restored. Alternatively, you can use the Step commands in the Debug menu to step through the execution of the method, one line at a time. Each time a step completes, the debugger highlights the next line of code to be executed. If the code is in another file, the debugger will look for the file on the Sourcepath and attempt to open it.

Step Into

The Step Into command will walk into any method calls that occur in the code, possibly opening additional files.

Step Over

The Step Over command will not enter any new method calls, treating them instead as a single operation to be stepped over.

Step Out

The Step Out command will finish running the current method and stop at the next line of code from where the method was called.

Sourcepath and Step Options

The sourcepath is the set of directories in which to look for source files when stepping. It can be set in the Debugger section of the Preferences window (which can be

opened from the Edit menu). This section in the Preferences also contains options for controlling which classes will be included as part of stepping. By default, DrJava will never step into its own source, nor its Java Interpreter (DynamicJava), nor Java itself. If you are interested, and have downloaded the source files, you can enable these options to see how Java or DrJava works. You can also specify which classes and packages you want to exclude when stepping. To exclude specific classes, type in the qualified class name (the package name followed by a period and the class name). To exclude entire packages (as well as their subpackages), type the package name followed by a period and an asterisk. Each class or package name must be separated by a comma.

Debug Panel

The Debug Panel appears when Debug Mode is on, with several informational tabs and buttons. DrJava displays all current breakpoints in a tree organized by document in the Breakpoints tab, and it displays currently watched fields and variables with their values in a table in the Watches tab. The Stack tab displays the current stack trace any time a method has been suspended, and the Threads tab shows all current threads along with their status at that point in time. Most of these tabs provide context (right-click) menus for easy access to related commands, such as scrolling to a breakpoint or a line in a stack frame, or resuming a suspended thread.

Watching Fields and Variables

You can watch the values of class fields and local variables by entering the field or variable name into a row in the Watches table. Any time a method is suspended (eg. after a breakpoint or step), the current value of the field or variable (if any) is displayed. Watches can be removed from the table by deleting the name and pressing Enter. You cannot enter expressions that need to be evaluated into the watch table. For example, "s.length" is not a valid watch entry. Type expressions like these into the Interactions Pane to see their values.

Chapter 8. Documentation with Javadoc

Documenting your code is crucial to help others understand it, and even to remind yourself how your own older programs work. Unfortunately, it is easy for most external documentation to become out of date as a program changes. For this reason, it is useful to write documentation as comments in the code itself, where they can be easily updated with other changes. Javadoc is a documentation tool which defines a standard format for such comments, and which can generate HTML files to view the documentation from a web broswer. (As an example, see Sun's Javadoc documentation for the Java libraries at http://java.sun.com/j2se/1.4/docs/api/index.html[1].)

You can easily run Javadoc over your programs from within DrJava, using the "Javadoc All Documents" and "Preview Javadoc for Current Document" commands in the Tools menu. These commands will generate Javadoc HTML files from the comments you have written and display them in a browser. This chapter gives a brief overview of these commands and how to write Javadoc comments. More detailed information on writing Javadoc comments can be found online at http://java.sun.com/j2se/javadoc/writingdoccomments/[2].

Writing Javadoc Comments

In general, Javadoc comments are any multi-line comments ("`/** ... */`") that are placed before class, field, or method declarations. They must begin with a slash and two stars, and they can include special tags to describe characteristics like method parameters or return values. The HTML files generated by Javadoc will describe each field and method of a class, using the Javadoc comments in the source code itself. Examples of different Javadoc comments are listed below.

Simple Comments

Normal Javadoc comments can be placed before any class, field, or method declaration to describe its intent or characteristics. For example, the following simple Student class has several Javadoc comments.

```
/**
 * Represents a student enrolled in the school.
 * A student can be enrolled in many courses.
 */
public class Student {

  /**
   * The first and last name of this student.
   */
  private String name;

  /**
   * Creates a new Student with the given name.
   * The name should include both first and
   * last name.
   */
  public Student(String name) {
    this.name = name;
  }

}
```

Using Tags

Tags can be used at the end of each Javadoc comment to provide more structured information about the code being described. For example, most Javadoc comments for methods include "`@param`" and "`@return`" tags when applicable, to describe the method's parameters and return value. The "`@param`" tag should be followed by the parameter's name, and then a description of that parameter. The "`@return`" tag is followed simply by a description of the return value. Examples of these tags are given below.

```
/**
 * Gets the first and last name of this Student.
 * @return this Student's name.
 */
public String getName() {
  return name;
}

/**
 * Changes the name of this Student.
 * This may involve a lengthy legal process.
 * @param newName This Student's new name.
 *                Should include both first
 *                and last name.
 */
public void setName(String newName) {
  name = newName;
}
```

Other common tags include "`@throws e`" (to describe some Exception "e" which is thrown by a method) and "`@see #foo`" (to provide a link to a field or method named "foo").

How to Use Javadoc in DrJava

In general, Javadoc HTML files are most useful when they are generated in large batches, since the HTML files for each of the related classes can link to each other. For this reason, DrJava's "Javadoc All Documents" command looks for all source files in the folders and subfolders of the open documents and includes them all in the documentation, saving the files in a "`doc`" folder nearby. (This folder will be placed either in the current folder or the top-level folder of the current package.) On the other hand, it is occasionally useful to view the Javadoc HTML for a single class, to quickly get a feel for its structure. Therefore, DrJava also provides a "Preview Javadoc for Current Document" command that only generates Javadoc for the current open document without saving it to a permanent location. (This command saves the file in a temporary location that will be automatically deleted when you quit DrJava.) If either of these commands finds errors in the source files, they will report them in a tab like compiler errors.

Viewing Javadoc

When either of the "Javadoc All Documents" or "Preview Javadoc for Current Document" commands complete successfully (or find only warnings and no errors), DrJava displays the resulting HTML files in a new window. For Windows and Mac OS X users, these files will be displayed in the system's default web browser. On other

platforms, the files will be displayed in a simple viewer, unless the "Web Browser" option has been configured in the "Resource Locations" section of the Preferences (see Configuring DrJava).

Configuring Javadoc

You can configure many aspects of how Javadoc files are generated. Most prominent is the ability to hide fields and methods below a particular access level (eg. `public` or `protected`). By default, no private fields or methods are shown. Other options include specifying a URL to link to the Java library API (which defaults to Sun's own website), specifying a default destination directory for all Javadoc files, and specifying any custom parameters to pass to the Javadoc tool itself. Finally, for programs with many nested packages (folders), DrJava provides an option to always generate Javadoc for all packages in the program, rather than just the sub-packages of the open documents.

Notes

1. http://java.sun.com/j2se/1.4/docs/api/index.html
2. http://java.sun.com/j2se/javadoc/writingdoccomments/

Appendix A. Configuring DrJava

DrJava has many configurable options which can be set using the Preferences command in the Edit menu, allowing the user to change both DrJava's appearance and behavior. Changes made to the configurable options are saved in a `.drjava` file in the user's home directory. The Preferences window is the preferred method for setting these options, although more experienced users may also edit the configuration file itself.

Preferences Window

The Preferences window is available in the Edit menu, and provides a graphical means to edit all configurable options in DrJava. It displays the options in several categories, each of which can be displayed as a panel of options. Each option has a tool tip with a short description, which can be displayed by holding the mouse arrow over the option.

The Apply button submits the changes on all panels and saves them to the config file, while the OK button does the same and also closes the window. The Cancel button closes the window without applying or saving the changes. Each panel also has a Reset to Defaults button, which resets each of the options on that panel to its original value. Resetting does not take effect until the changes are applied with the Apply or OK buttons.

Editing the Config File

All configured options are stored in the `.drjava` file in the user's home directory. (The location of this file varies on different platforms.) This file is a standard Java properties file, with one option on each line and with option keys and values separated by an equals sign. Any options not defined in this file will have their default value. While it is recommended to use the Preferences window in most cases, the config file can also be edited manually to adjust values as desired. The correct option keys and default values for each option are given in the Available Options section.

Note: All parameters are parsed as standard Java strings, so escape characters must be considered. Notably, to include a Windows-style path in a parameter value, all backslashes must be escaped. For example:

```
javac.location=c:\\jdk1.4\\lib\\tools.jar
```

Available Options

All available configuration options are displayed here. The option keys and default values are also provided for users who wish to edit their configuration file.

Resource Locations

These options specify where to find Java resources on your computer, such as com-

pilers or classpath directories.

Web Browser (`browser.file = ""`)

Optional filename of your computer's web browser, to be used to view Javadoc and links from the Help. If blank, the platform's default browser (on Windows or Mac OS X) will be used unless you specify a Web Browser Command (explained below).

Web Browser Command (`browser.string = ""`)

Any command or arguments you need to start your web browser. If both this option and the Web Browser option are blank, the platform's default browser will be used. In this option, any occurrence of "<URL>" will be replaced with the URL to open. (If "<URL>" is not specified, the URL will be appended to the end of the command.)

On Windows and Mac OS X, you can just leave this option blank and your default web browser will be used. For other platforms (eg. Linux), here are commands to open common browsers:

- Mozilla (if it is already running)

  ```
  mozilla -remote "openurl(<URL>)"
  ```

- Mozilla (if it is not already running)

  ```
  mozilla <URL>
  ```

- Konqueror (the KDE web browser)

  ```
  konqueror <URL>
  ```

Tools.jar Location (`javac.location = ""`)

Specifies the location of the JDK's `tools.jar`, which contains the classes necessary for the compiler and the debugger. This file is usually found in the JDK's `lib` directory.

JSR14 Location (`jsr14.location = ""`)
JSR14 Collections Path (`jsr14.collectionspath = ""`)

Specifies the location of the JSR-14 versions of `javac.jar` and `collect.jar`, respectively, for use as an alternative compiler in DrJava. JSR-14 is an experimental compiler provided by Sun which supports generic types. The `collect.jar` file contains parameterized collection classes for JSR-14. (We currently support JSR-14 versions 1.0, 1.2, 1.3, 2.0, and 2.2. Note that for versions 2.0 and later, `javac.jar` has been renamed `gjc-rt.jar`.)

Extra Classpath (`extra.classpath = ""`)

Used to specify any directories or `jar` files to append to the classpath of the Interactions window and the compiler. Separate the directories using the system-specific path separator (eg. colon on Unix, semicolon on Windows).

Display Options

These configurable options affect how DrJava's user interface is displayed.

Look and Feel (`look.and.feel = ""`)

Name of the Swing LookAndFeel class which determines the general appearance of DrJava. If this option is changed while DrJava is running, the changes will not apply until you restart.

Toolbar Buttons (`toolbar.icons.enabled = true, toolbar.text.enabled = true`)

These radio buttons control whether the toolbar buttons contain text, icons, or both. When set manually in the config file, each of the two options can be set to true or false, though icons will be displayed if both are set to false.

Show All Line Numbers (`linenum.enabled = false`)

Whether to display all line numbers along the left margin of the Definitions Pane.

Save Main Window Postion (`window.store.position = true`)

Whether to save the position and size of the DrJava window between sessions.

Font Options

Each font option is specified as a string containing the font name, style, and size, separated by dashes. The style should be in upper-case (ie. PLAIN, BOLD, ITALIC, or BOLDITALIC), while the font name must be a valid font on the system. (In most cases, using the font chooser in the Preferences window is the simplest approach.)

Main Font (`font.main = Monospaced-PLAIN-12`)

This font is used for the definitions pane and the tabs at the bottom of the window.

Line Numbers Font (`font.doclist = Monospaced-PLAIN-12`)

This font is used for the line numbers on the left side of the Definitions Pane, if the "Show All Line Numbers" option in the "Display Options" section is enabled. The actual font size will be limited by the size of the Main Font.

Document List Font (`font.doclist = Monospaced-PLAIN-10`)

This font is used in the list of all open documents on the left side of the window.

Toolbar Font (`font.toolbar = dialog-PLAIN-10`)

This font is used on the toolbar buttons, if the button names are configured to be displayed.

Color Options

Colors are defined similarly to HTML colors: as six hexadecimal digits preceded by a pound sign. The first two digits specify a red value, the next two specify a green value, and the next two specify a blue value. For example, `#00FF00` would be a bright green. (In most cases, using the color chooser in the Preferences window is the simplest approach.)

Syntax Colors for Definitions

Normal Color (`definitions.normal.color = #000000`)

Used as the default color for program text.

Keyword Color (`definitions.keyword.color = #0000FF`)

Used as the color for known keywords (eg. "public", "for").

Type Color (`definitions.type.color = #00007C`)

Used for known primitive types (eg. "int") and capitalized words, which usually correspond to class names.

Comment Color (`definitions.comment.color = #007C00`)

Used as the color for all comments.

Double-quoted Color (`definitions.double.quoted.color = #B20000`)

Used as the color for strings, which use double quotation marks.

Single-quoted Color (`definitions.single.quoted.color = #FF00FF`)

Used as the color for characters, which use single quotation marks.

Number Color (`definitions.number.color = #00B2B2`)

Used as the color for all numbers.

Other Colors

Background Color (`definitions.background.color = #FFFFFF`)

Used as the background color for all panes.

Brace-matching Color (`definitions.match.color = #BEFFE6`)

Used as the highlight color when matching braces.

Compiler Error Color (`compiler.error.color = #FFFF00`)

Used as the highlight color for compiler errors and JUnit test failures.

Debugger Breakpoint Color (`debug.breakpoint.color = #FF0000`)

Used as the highlight color for breakpoints set from the debugger.

Debugger Location Color (`debug.thread.color = #64FFFF`)

Used as the highlight color for the location of the current thread in the debugger, shown after a breakpoint is hit or a step has occurred.

System.out Color (`system.out.color = #007C00`)

Used as the color for text from System.out.

System.err Color (`system.err.color = #FF0000`)

Used as the color for text from System.err.

System.in Color (`system.in.color = #7C007C`)

Used as the color for text to be read by System.in.

Interactions Error Color (`interactions.error.color = #B20000`)

Used as the color for text that indicates errors in the Interactions Pane.

Debug Message Color (`debug.message.color = #0000B2`)

Used as the color for text displayed by the debugger.

Key Bindings

Most menu items in DrJava have configurable keyboard shortcuts, along with several other navigational commands (such as moving to the beginning or end of a line). All such options are displayed on the Key Bindings panel in the Preferences window, along with their current value. Clicking on the value displays a window which allows the user to type a new key, showing any conflict with an existing key if there is one. (We recommend editing these options in the Preferences window.)

Debugger

All configurable options relating to the debugger.

Sourcepath (`debug.sourcepath = ""`)

A list of directories on which to search for source files when stepping through code. The debugger will attempt to open files from these directories automatically when stepping.

Step Into Java Classes (`debug.step.java = false`)

Whether to step into Java source files when stepping through a suspended method call. It is recommended to put the Java source (usually distributed with the JDK) on the Sourcepath if this option is selected.

Step into Interpreter Classes (`debug.step.djava = false`)

Whether to step into DynamicJava source files when stepping through a suspended method call. DynamicJava is the Java interpreter used in the Interactions pane, and the source can be obtained from http://koala.ilog.fr/djava[1]. Useful primarily when debugging DrJava itself.

Step into DrJava Classes (`debug.step.drjava = false`)

Whether to step into DrJava source files when stepping through a suspended method call. Useful primarily when debugging DrJava itself.

Classes/Packages To Exclude (`debug.step.exclude = ""`)

Classes and packages that you do not wish DrJava to step into. These must be fully qualified class names or package names ending in ".*" (e.g. "java.util.*") separated by commas.

Javadoc

All configurable options relating to generating Javadoc.

Access Level (`javadoc.access.level = "package"`)

Specifies the lowest access level for fields and methods to include in the generated documentation. Legal values are "public", "protected", "package", and "private".

Java Version for Javadoc Links (`javadoc.link.version = (JDK Version)`)

Specifies which URL to use when generating links to Java library classes. Legal values are "1.3", "1.4", and "none" if no links to Java library classes are desired. (This option defaults to the version of the user's JDK.)

Javadoc 1.3 URL (`javadoc.1.3.link = "http://java.sun.com/j2se/1.3/docs/api"`)

The URL to use when generating links to JDK 1.3 library classes.

Javadoc 1.4 URL (`javadoc.1.4.link = "http://java.sun.com/j2se/1.4/docs/api"`)

The URL to use when generating links to JDK 1.4 library classes.

Default Destination Directory (`javadoc.destination = ""`)

If a directory is specified, it will be used as the default when generating new documentation.

Custom Javadoc Parameters (`javadoc.custom.params = ""`)

Any custom parameters to pass to the Javadoc tool, separated by spaces. Use "javadoc -help" at a command line to view the available parameters.

Generate Javadoc From Source Roots (`javadoc.from.roots = false`)

If this option is enabled, then Javadoc will not only search the current package and all subpackages for files, it will also search all "enclosing" packages (those at a higher level).

Notifications

Configures how often DrJava notifies you for certain events. The notifications in this section can all be suppressed by clicking on a "Do not show this message again" checkbox (or similar) on the notification itself.

Prompt Before Quit (`quit.prompt = true`)

Whether to display a confirmation message before DrJava quits.

Prompt Before Resetting Interactions Pane (`interactions.reset.prompt = true`)

Whether to display a confirmation message before resetting the Interactions Pane.

Prompt if Interactions Pane Exits Unexpectedly (`interactions.exit.prompt = true`)

Whether to display a message if the Interactions Pane is exited without the Reset button being clicked.

Prompt for Javadoc Destination (`javadoc.prompt.for.destination = true`)

Whether to always display the destination selection dialog when starting Javadoc.

Automatically Save Before Compiling (`save.before.compile = false`)

Whether to automatically save all files each time a Compile command is chosen.

Automatically Save Before Generating Javadoc (`save.before.javadoc = false`)

Whether to automatically save all files each time a Javadoc command is chosen.

Warn on Breakpoint Out of Sync (`warn.breakpoint.out.of.sync = true`)

Whether to warn if setting a breakpoint in a source file that is not in sync with its class file.

Warn if Debugging Modified File (`warn.debug.modified.file = true`)

Whether to warn if using the debugger on a file which has been modified since its last save.

Warn to Restart to Change Look and Feel (`warn.change.laf = true`)

Whether to warn that changes to the Look and Feel do not take effect until after a restart.

Miscellaneous

These are the remaining configurable options in DrJava.

Indent Level (`indent.level = 2`)

Sets how many spaces to use for each level of indenting. Note that tab characters are not allowed in DrJava.

Working Directory (`working.directory = ""`)

Specifies the default directory for DrJava to use when starting up. The open and save dialogs will start here, rather than in the user's current directory, if this option is set. The Interactions classpath will also include this directory.

Size of Interactions History (`history.max.size = 500`)

Specifies how many commands will be remembered in the history of the Interactions window. Previous commands can be recalled using the up and down arrow keys.

Recent Files List Size (`recent.files.max.size = 5`)

Specifies how many recently used files to display in the File menu.

Automatically Close Block Comments (`auto.close.comments = false`)

Whether to automatically insert the string designating the end of a multi-line comment after beginning one.

Allow Assert Keyword in Java 1.4 (`javac.allow.assert = false`)

Whether to support the `assert` keyword when compiling with a JDK 1.4 or later compiler.

Keep Emacs-style Backup Files (`files.backup = true`)

Whether DrJava should keep a backup copy of each file that the user modifies, saved with a "~" at the end of the filename.

Clear Console After Interactions Reset (`reset.clear.console = true`)

Whether DrJava should clear the contents of the Console Tab each time the Interactions Pane is reset.

Notes

1. http://koala.ilog.fr/djava

Appendix B. Setting up DrJava on your Platform

DrJava can be set up to handle files similarly to other applications on your platform. For example, Windows allows you to associate types of files with particular applications.

Associating Java files to DrJava, on Windows

To set up DrJava as your default editor for `.java` files in Windows, you will need a few pieces of information:

1. The complete path to your Java executable, likely in `C:\Program Files`. We recommend using `javaw.exe`, since it launches programs without displaying a command prompt window.
2. The complete path to your DrJava Jar file. To simplify upgrading, we recommend renaming the Jar file to `drjava.jar` each time you download it, and always keeping it in the same folder.

Using the above information, you can change the file associations for `.java` files:

1. Find the Folder Settings dialog. Depending on your version of Windows, it might be in one of several locations:
   - The "Settings" group of the Start Menu
   - The "Tools" menu in Windows Explorer
   - An icon in the Control Panel
2. Within the Folder Settings dialog, click on the File Types tab.
3. Search the list of file types for the `.java` extension. If it is not already in the list:
   a. Click the "New" button to create a new file type.
   b. Type in "java" as the extension and click OK.
4. Select the `.java` extension from the list and click on the "Advanced" button. The "Edit File Type" dialog will be displayed.
5. If the File Type does not have a name in the top text field, type in "Java Source File".
6. Click the "New" button next to the list of Actions to display the "New Action" dialog.
7. Type "open" as the action.
8. For the application to perform the action, type:

   "C:\PATH\javaw.exe" -jar "C:\PATH\drjava.jar" "%1"

   (Replace C:\PATH\javaw.exe with the full path to your Java executable, and C:\PATH\drjava.jar with the full path to your DrJava Jar file, as found in the first few steps.)

9. Optionally select an icon for all `.java` files, then click OK.
10. Click the "Close" button.
11. All `.java` files should now be associated with DrJava, so double-clicking on one will open it in a new copy of DrJava.

Appendix C. Indenting Files from the Command Line

DrJava has a very useful indenting algorithm, but indenting several large files can be a time consuming process. Because of this, DrJava provides a command line interface to its indenter that can be run on a series of files.

Running the Command Line Indenter

Use the following command at a command prompt to run the indenter on a series of files.

java -classpath drjava-DATE-TIME.jar edu.rice.cs.drjava.IndentFiles [-indent N] [filename.java...]

Replace DATE-TIME with the appropriate value for your DrJava file. The "-indent" argument is optional, where N is the number of spaces to use for an indentation level.