



A Pedagogic Programming Environment for Java that Scales to Production Programming

Charles S. Reis
Master's Thesis Defense
Rice University
April 16, 2003



Pedagogic IDEs

- Useful tools in courses
 - Simple, easy to learn
 - Guiding philosophy for features
 -
- Usually limited to intro level
 - Lack of powerful features
 - Restrictive interfaces (eg. UML)



Professional IDEs

- Many advanced features
- Large, Cumbersome
 - Significant overhead
 - Complex user interfaces
 - Not designed for students
 -
- Avoided by many professionals!



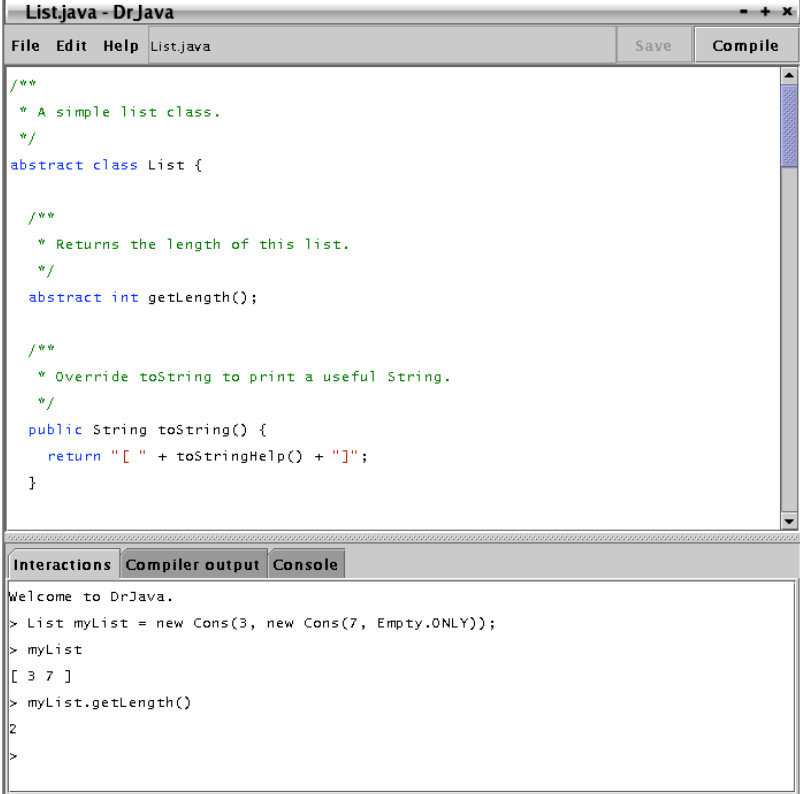
Motivation

- Can pedagogic IDEs be useful at the level of production development?



DrJava

- Pedagogic IDE for intro level
 - Simple, intuitive
 - Interactive (REPL)
 - Focus on source language



```
List.java - DrJava
File Edit Help List.java Save Compile

/**
 * A simple list class.
 */
abstract class List {

    /**
     * Returns the length of this list.
     */
    abstract int getLength();

    /**
     * Override toString to print a useful String.
     */
    public String toString() {
        return "[" + toStringHelp() + "]";
    }
}

Interactions Compiler output Console
Welcome to DrJava.
> List myList = new Cons(3, new Cons(7, Empty.ONLY));
> myList
[ 3 7 ]
> myList.getLength()
2
>
```



DrJava Development

- Created by students at Rice
 - Object oriented, design patterns
 - Extreme Programming (XP)
 - Open Source
 -
- Customers worldwide



Goal

- Extend DrJava to support production programming
 - Small set of new features
 - Ease transition to professional IDEs
 - Teach production programming with DrJava

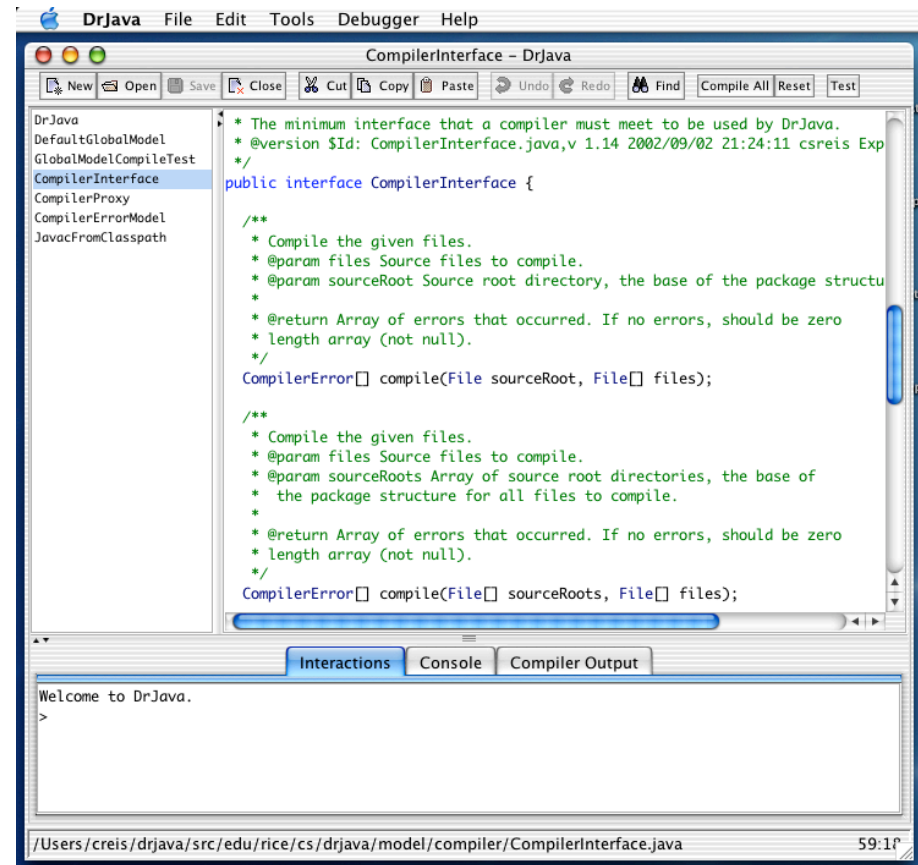


Necessary Features

- REPL still useful
- Easy access to multiple files
- Traditional debugger
 - Suspend execution, query values
- Test-driven development
 - Integrated support for unit tests

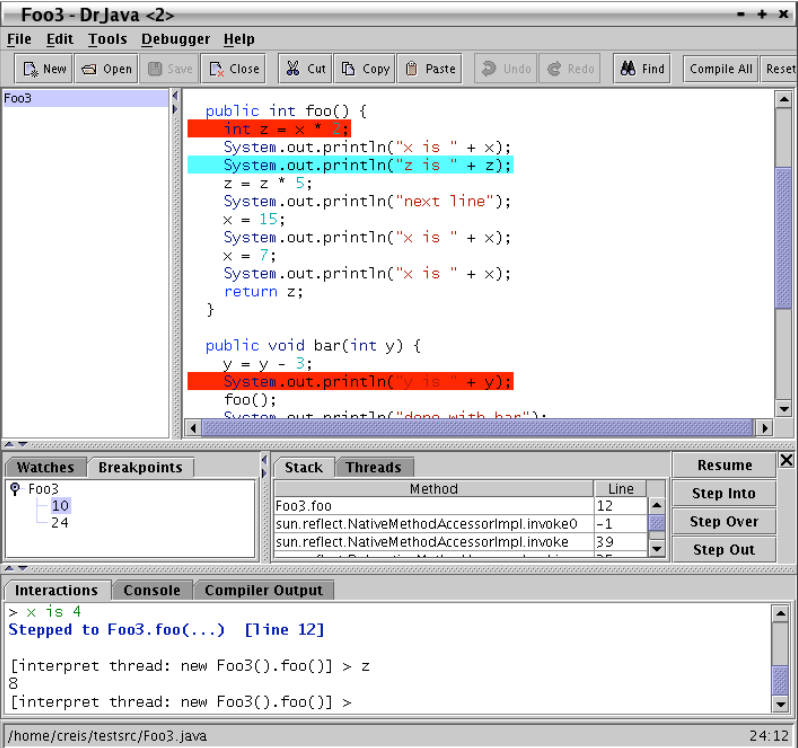
Editing Multiple Files

- Java projects span many files
 - Need convenient access to many classes at once
 -
- Add document selector



Interactive Debugger

- Traditional features
 - Breakpoints
 - Stepping
 - Query values
 -
- Integrated with Interactions Pane (REPL)



The screenshot displays the DrJava IDE with the following components:

- Code Editor:** Shows the source code for `Foo3`. The current execution point is at line 12: `System.out.println("z is " + z);`.
- Stack:** Shows the current stack frame for `Foo3.foo` at line 12, along with native method frames.
- Interactions Pane (REPL):** Shows the command prompt with the output `> x is 4` and the message `Stepped to Foo3.foo(...) [line 12]`.
- Console:** Shows the output of the program, including `[interpret thread: new Foo3().foo()] > z` and `8`.

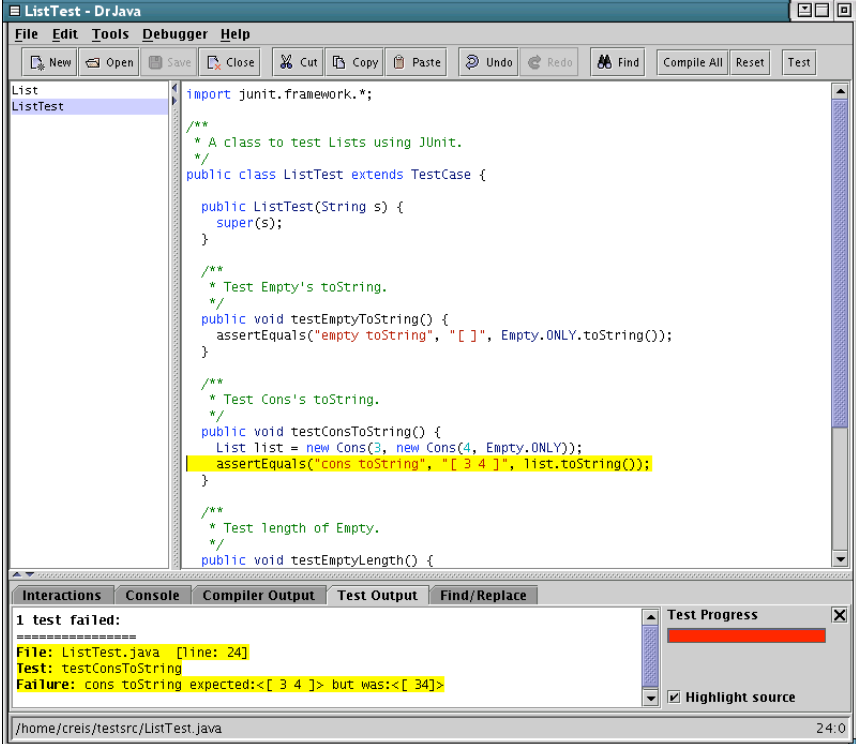


Debugger + REPL

- Flexible Points of Entry
 - Not just `main` method
 - Easily repeat experiments
 -
- Interact with state in Java
 - Query, modify values
 - Call methods, etc

Unit Testing Support

- Key to incremental development
- Quality Safeguard
-
- Easy to write, run
 - JUnit framework
 - “Test” button
 - Visual feedback



The screenshot shows a Java IDE window titled "ListTest - Dr Java". The main editor displays the following Java code:

```
import junit.framework.*;

/**
 * A class to test Lists using JUnit.
 */
public class ListTest extends TestCase {

    public ListTest(String s) {
        super(s);
    }

    /**
     * Test Empty's toString.
     */
    public void testEmptyToString() {
        assertEquals("empty toString", "[ ]", Empty.ONLY.toString());
    }

    /**
     * Test Cons's toString.
     */
    public void testConsToString() {
        List list = new Cons(3, new Cons(4, Empty.ONLY));
        assertEquals("cons toString", "[ 3 4 ]", list.toString());
    }

    /**
     * Test length of Empty.
     */
    public void testEmptyLength() {
```

The IDE interface includes a menu bar (File, Edit, Tools, Debugger, Help), a toolbar with icons for New, Open, Save, Close, Cut, Copy, Paste, Undo, Redo, Find, Compile All, Reset, and Test. A file explorer on the left shows "List" and "ListTest". Below the editor, there are tabs for "Interactions", "Console", "Compiler Output", "Test Output", and "Find/Replace". The "Test Output" tab is active, displaying the following error message:

```
1 test failed:
=====
File: ListTest.java [line: 24]
Test: testConsToString
Failure: cons toString expected:<[ 3 4 ]> but was:<[ 34]>
```

At the bottom right, there is a "Test Progress" section with a red progress bar and a "Highlight source" checkbox. The status bar at the bottom shows the file path "/home/creis/testsrc/ListTest.java" and the time "24:0".

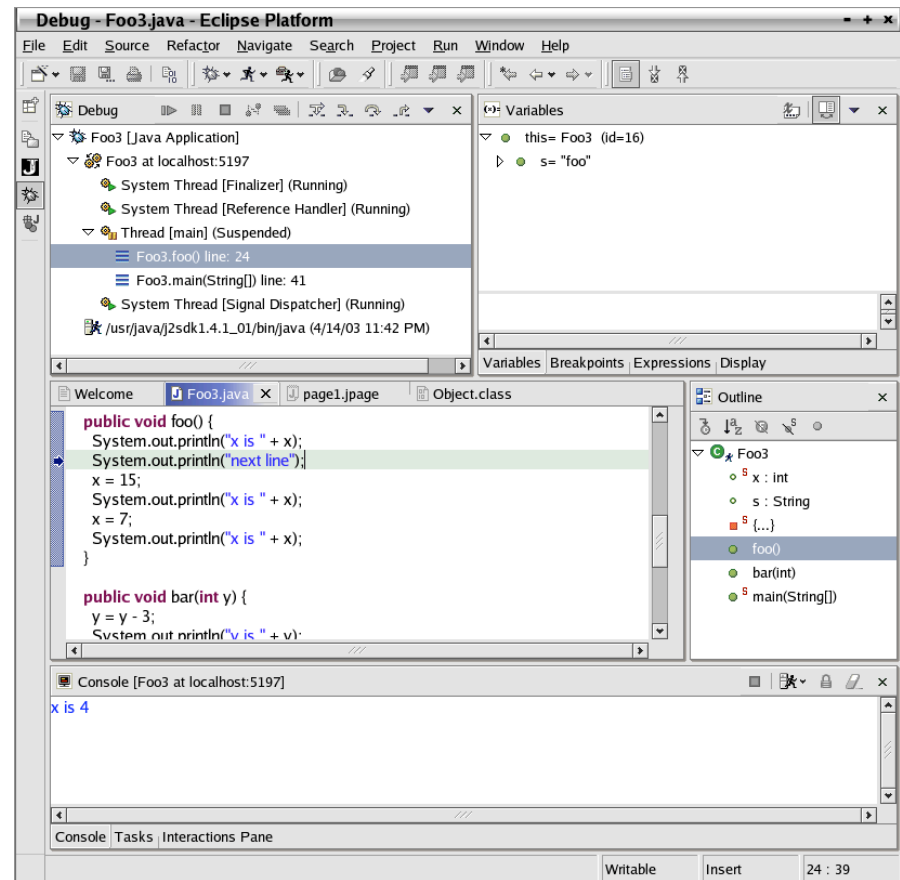


Leveraging Professional IDEs

- Occasionally useful
 - Powerful refactoring tools
- Desire an easier transition from pedagogic IDEs

IBM's Eclipse

- Widely used, open source IDE
- Everything is a "plug-in"
- Many advanced features
- Active ties with academia





DrJava Plug-in for Eclipse

- Innovation Grant from IBM
- Ease transition to Eclipse
 - Simplify user interface
 - Provide Interactions Pane (REPL)
 - Debugger + REPL
 -
- REPL also useful for professional developers



Plug-in Development

- Code Re-use
 - All logic directly from DrJava
 - Single point of control for bug fixes, feature improvements
- Refactoring DrJava
 - More modular design
 - Safe and easy: unit tests!



DrJava's Scalability

- DrJava team uses DrJava
 - Effective tool for its own development
 -
- Scales to Production Programming



Teach Production Programming

- Use DrJava to teach production programming skills
 - Common, familiar environment
 - Select DrJava as course project!



Extend DrJava in a Course

- Students can:
 - Learn effective practices (XP)
 - Join an existing product team
 - Maintain a product
 - Support customers



Extreme Programming

- Expose students to effective development practices
 - Ubiquitous unit testing
 - Pair programming
 - On-site customer
 - Incremental releases



Classroom Challenges

- Time constraints
- Scarce resources
-
- How to:
 - Quickly transfer knowledge?
 - Adapt Extreme Programming?
 - Manage development?



XP: Knowledge Transfer

- Pair Programming
 - With experienced TAs
 - With other students
 -
- Unit Tests
 - Executable documentation



Adapting XP for Classroom

- Pair Programming
 - Lab time, students select own pairs
- On-site Customers
 - Students themselves (using DrJava)
- No fixed deadlines
 - Require 10 logged hours per week



Managing Development

- TA's as Project Managers
- SourceForge.net
 - Free open source project hosting
 - Professional quality management
 - Bug reports
 - Feature requests
 - Task management



Case Study: COMP 312

- Early unit test assignment
- Bug fixes
- Large features in small tasks
 - 2002: JUnit support, debugger, configurability
 - 2003: Interactive debugger, Javadoc, interactions pre-processor



Results

- DrJava effective for production programming
 - Used in its own development
- Eclipse plug-in eases transition
- Excellent results from 312
 - Many core features implemented
 - Students exposed to process



Conclusion

- Pedagogic IDEs can scale to production development
 - DrJava's simplicity preserved
 - Effective for large projects
 - Useful for teaching production programming skills