Immediate connection between creator and creation

Supervisor: Joseph Kiniry, kiniry@itu.dk.

Student: Anders Bech Mellson, anbh@itu.dk, 310780.

July 31, 2012

Acknowledgments

Thank you Joe Kiniry for supervising the project, it has been a pleasure working with you! Thank you David Christiansen for inspiring me and taking the time to answer questions. Thanks to Lars Vogel for his great tutorials. Thanks to Swaminathan Bhaskar for his article on dynamic compilation. Finally thanks to Bret Victor for giving an inspirational talk that poked my curiosity.

Contents

Chapter 1

Abstract

"Immediate connection between creator and creation" is a project about realtime textual feedback (*immediate connection*) to the programmer (*creator*) whilst writing Java code (*creation*) in Eclipse. This is realized with a prototype Eclipse plugin called Mirror. Mirror is able to connect to variable declaration statements in the source code and visualize them in the plugin window. Mirror works with Eclipse version 3.7 aka Indigo.

Chapter 2

Introduction

2.1 Organization Of This Report

The report starts with my motivation for creating this project. This is followed by the formal project description. Then I will look at what the problem is and how it can be solved.

The report ends with my conclusion, which includes goals from the project description. Both those I have reached and those I have not.

2.2 Motivation

I recently started learning programming. To be exact it was three years ago when I started my bachelor's degree in software engineering. I remember being scared of journeying into the world of programming. Why? Because I thought it was too hard for me to understand. Turns out it was hard!

Can we do something to ease the learning curve of beginning programmers? I think so. The bulk of mainstream languages and IDE's on the market today lack realtime feedback.

I come from a musical background. When you play music, realtime feedback is essential for tuning and perfecting your output. I do not think artists and programmers are different in that aspect. Creation and creativity is something that comes from within. In order to perform well, we need to see, hear, feel, touch, etc. what we are creating at the instant we do it. Programmers have not been given this opportunity yet!

There are many possible ways to approach this problem. I will try to give realtime feedback on Java source code in Eclipse.

Chapter 3

Project Description

Background

Writing code requires a lot from the creator. Current tools for writing software not only require you to master the programming language itself - they also enforce you to play computer in your head. There is no immediate feedback on what your code will do whilst typing.

This disconnects the creator from the creation.

Theory

A theory of immediate connection is an idea by Bret Victor, presented during his talk at CUSEC 2012.

An example of how this theory could be applied is shown during the talk, in the form of a binary search algorithm. The algorithm is written in code, and as it is being typed there is an immediate visualization of the variables and calculations.

This is one example, which gives that missing immediate connection.

Project Work

This project will investigate the theory and materialize it in a proof of concept prototype. Understanding the coding process and finding the appropriate place to attach the prototype will be essential. Should it be in the compiler, interpreter or a third place?

Visualizing the code will be another very important aspect, getting a textual representation is the first priority. An optional visualization will be to look at third party libraries for creating a graphical representation.

The investigation will also involve looking at which languages this theory could be applied to.

Primary Goals

- Create a proof of concept prototype.
- Create a textual visualization.

Secondary Goals

- Make the prototype as a plugin to an open source IDE.
- Create an open source community around the prototype on github.com.
- Create a graphical visualization.

Chapter 4

Problem Analysis - Design

4.1 Problem

Why does a beginning programmer need immediate feedback from code?

Let us think of a programmer as an artist. Imagine a musician recording a song. However the musician cannot hear the instrument he or she is playing until the recording is done. That song could, by luck, be very creative and good. But in most cases we cannot rely on luck. We strive after full control of the creation process.

Today a programmer needs to compile and execute code manually. This adds an additional layer between you and your creation. Just as a musician tries various notes, a programmer tries different statements. This project will focus on how these statements can be visualized in realtime, removing the layer added by manual compilation and execution. My hope is that it can help intuition and learning.

4.2 Scope

When I started programming three years back, the first language I learned was Java. Java was a good beginner language for me because of the following reasons.

- It is very mainstream, which makes it easy to get help when you are stuck (which we all are at some point).
- It is an object oriented language, which makes modeling fairly easy.

Because of these factors and the popularity of Java[1], I think that many others like me will start their programming career using Java. This is why I choose Java as the language I will support in this project.

Since this is a time and resource limited project, I will have to focus my effort. Thinking three years back, what would I have wanted to help my learning process?

- Help to see variables interact.
- Help to see what math like modulo do.

[1] Java is number 2 in the TIOBE Programming Community Index for July 2012 - `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`

One way to help with these issues could be to look at concrete values for variables. So instead of looking at names like `int i` and `double d` the real values would be visible.

I will try to create a plugin that does exactly that. I will only try to give feedback on variable declaration statements. This is a restraint I will make because of the time frame. Hopefully that will increase the likely hood of meeting the project deadline. If I however get these statement types implemented quickly I will try to support more statement types.

To sum it all up. I will do a plugin for Eclipse. The plugin will work on Java source files. The plugin will compile and execute the source code, and it will give realtime feedback on variable declaration statements.

4.3 Initial Concerns

Because I have chosen to do instant compilation and execution for Java source code, some concerns come to mind. Executing code can have some unwanted side effects. As an example the user could write a piece of code that would delete the contents of the hard drive.

Before executing a method I could prompt the user for permission. If the user says 'ok', then the method will go on a white list. If not it will be black listed and never executed. This approach could be valid if safety was a concern at this point. Although I think it would be cumbersome for the user.

When should I break out of a loop? Loops could be running forever. I could start each execution in a thread with a timeout. This could solve the executions that never stop. But it would also kill valid calculations that are not finished.

These are just two concerns. Instead of finding more problematic cases I choose to emit those concerns from this project. The reason for this is that I am building a prototype which needs to prove that the concept[2] is valid. I am afraid it would take my focus away from working the actual concept if I started guarding against all problematic cases.

4.3.1 Development Model

Every project needs a structured way of working. I will use iterative development for this project. Why choose iterative development?

- I am unfamiliar with the application area.
- I have no idea if I got the prerequisites right (or even all of them).
- The design is challenging.

According to Steve McConnell[3], these factors favor an iterative model. If I was knowledgeable in this application area, I could have considered a sequential model.

4.4 Eclipse Architecture

Eclipse is a very comprehensive application. Although it is modular, it packs quite a load even in the smallest distribution (173 mb). I will try to create an overview of the application, with a focus on what I need for this project.

[2] The concept of creating an immediate connection between creator and creation.

[3] [Code Complete, Second Edition, chap. 3]

Eclipse itself is a framework for building integrated development environments (IDE). When most people say Eclipse, what they actually refer to is the Eclipse Software Development Kit (SDK). The Eclipse SDK consists of several Eclipse projects including Platform and the Plugin Development Environment (PDE). From here on out I will refer to the Eclipse SDK as Eclipse.

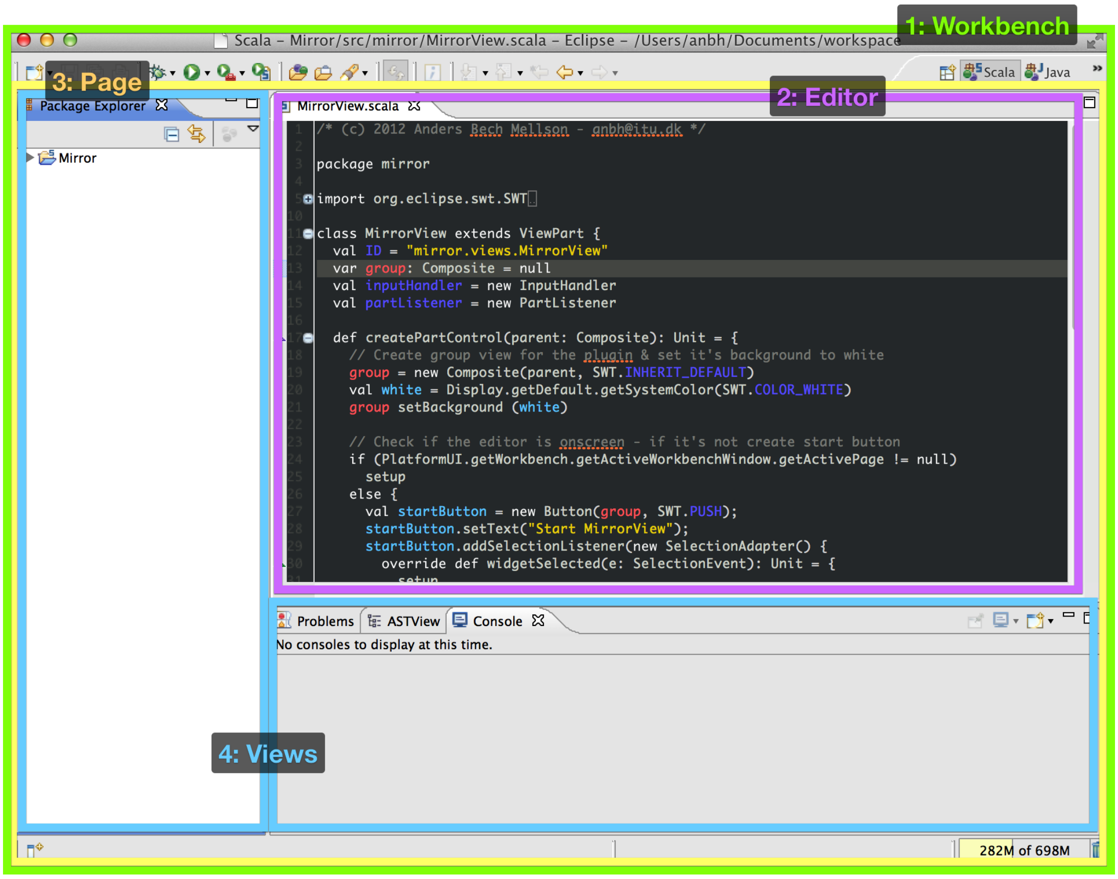
Eclipse uses a dynamic plugin model. It is built for discovering, integrating and running plugins. Plugins are based on the OSGi[4] specification. Everything in Eclipse, except for the kernel (Platform Runtime), is made of plugins. Each plugin has a manifest file, which declares its connections to other plugins via extensions. It also declares its own extension points, where other plugins can extend it.

Eclipse uses the notion of a workspace. A workspace is a collection of projects. The platform provides a mechanism for tracking changes in the workspace resources. This mechanism is called resource change listeners.

Lets have a look at the user interface (UI) for Eclipse. This interface is called the workbench in Eclipse. The workbench is what people think of, when the say Eclipse. The UI paradigm consists of views, editors and perspectives. Perspectives will not be visible in figure 4.1. A perspective is a collection of visible views and editors. Therefor it cannot directly be seen. The perspective in the figure however resembles a standard Java development perspective. The workbench and its application programming interface (API) is built on the following two toolkits.

- SWT - A widget and graphics library. Integrated with the native OS' window system.
- JFace - A library built on SWT that makes common UI programming tasks easier.

[4] The OSGi Alliance is a worldwide consortium of technology innovators that advances a proven and mature process to create open specifications that enable the modular assembly of software built with Java technology - http://osgi.org



```
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror

import org.eclipse.swt.SWT

class MirrorView extends ViewPart {
  val ID = "mirror.views.MirrorView"
  var group: Composite = null
  val inputHandler = new InputHandler
  val partListener = new PartListener

  def createPartControl(parent: Composite): Unit = {
    // Create group view for the plugin & set it's background to white
    group = new Composite(parent, SWT.INHERIT_DEFAULT)
    val white = Display.getDefault.getSystemColor(SWT.COLOR_WHITE)
    group setBackground (white)

    // Check if the editor is onscreen - if it's not create start button
    if (PlatformUI.getWorkbench.getActiveWorkbenchWindow.getActivePage != null)
      setup
    else {
      val startButton = new Button(group, SWT.PUSH);
      startButton.setText("Start MirrorView");
      startButton.addSelectionListener(new SelectionAdapter() {
        override def widgetSelected(e: SelectionEvent): Unit = {
          setup
```

No consoles to display at this time.

Figure 4.1: User Interface for Eclipse 3.7.

The two parts I am most interested in is the editor, and the views. The editor can hopefully get me the source code via a resource listener. And the plan is to create the plugin in a view with SWT and JFace.

4.5 Understanding Eclipse Development

There are many IDE's available. I have chosen to develop in Eclipse and for Eclipse. There are also many languages available. I have decided to split my development effort in two. The plugin I will build, will support Java. But the plugin itself I will write in Scala.

Why use Eclipse?

- Eclipse is built to be extended.
- It has a very large user base.
  - Which makes it possible to reach many users.
  - It also makes it more likely that others can help with the further development of this project.

- It is free.

Before finally deciding on Eclipse I had a look at IntelliJ IDEA by Jetbrains. Personally I prefer to code in Jetbrains products. But as Joe Kiniry reminded me, their products do not have the massive user base that Eclipse has. This is quite important when I have to look for help online. When I tried to search for plugin development in IntelliJ, the result were few and not very helpful.

Why use Scala?

- It is a multi paradigm language. It is both a functional and an object oriented language.
- It has a Read Evaluate Print Loop (REPL).
- It is interoperable with Java code.
- It runs on the Java Virtual Machine (JVM).

Eclipse plugins are normally written in Java. But because of Scala's interoperability with Java, it is possible to write a plugin in Scala. Luckily David Christensen and Hannes Mehnert from the IT University has already done a plugin in Scala[5]. This is a helpful resource to have. Because at this point it is not easy to find guides on doing Eclipse plugins in Scala.

I will use the book [Scala for the Impatient] to get my Scala skills going. I do not expect that I will master the language during this project, but I hope I will learn the basics of the language.

[5] Kopitiam - modular incremental interactive full functional static verification of Java code. `https://github.com/hannesm/Kopitiam`

Chapter 5

Problem Analysis - Creation

5.1 Mirror Is Born

Every project needs a name. The traits of this project made me think of a mirror. A reflection of what is happening inside the computer. A faithful representation of what it is doing with your code as you type it.

mir·ror
something that gives a minutely faithful representation, image, or idea of something else

Figure 5.1: Definition of Mirror from Dictionary.com.

5.2 Starting Eclipse Development

First thing I need to do is create an Eclipse plugin project. Since I never did one before I search for a tutorial. Because Eclipse has a massive user base, there are many opinions online. And because of its many versions, there is as many versions of the documentation. This makes it hard to find current and relevant knowledge.

Maybe Lars Vogel, creator of Vogella.com, wanted to solve that problem. At least he has solved it for me. His site helps me get started with Eclipse development. I follow his tutorial [Extending Eclipse - Plug-in Development Tutorial], which is one of the best tutorials I have ever seen. It is easy to follow and it explains the topic in a straight forward manner.

Vogel's tutorial is for Java. So I translate the parts I can into Scala instead. I am quickly impressed of how fun it is to write Scala code. It looks much better than other languages I have used. I like that I can do many method calls without the parentheses and line endings without semicolons;

I have left one file in the Eclipse plugin project as Java, the activator class. Simply because Eclipse needs it to be Java to start the plugin.

After a few days doing plugin development I have some thoughts about it.

- Eclipse is a big application, both in a good and bad way.
  - The good being the many possibilities.
  - The bad is finding my way amongst all these possibilities.
- Eclipse is very slow on my MacBook Air from 2010.

- Many versions of Eclipse leads to problems finding current knowledge.
- Eclipse looks very old compared to something like Sublime Text.
- I really enjoy Scala so far.
- The Plugin manifest file is very cumbersome to work with.

I now have a plugin running in Eclipse. In the next sections I will talk about how to get the source code, and how to execute it. I want the plugin to execute which ever method the user has the caret placed in.

5.3 Getting The Source Code

I need the source code from the editor as it is being typed. I have spent several days playing with various code pieces. Snippets that I have from tutorials and Stackoverflow[1], however none of them really do what I want.

After some days of frustration I finally stumble across a good starting point[2] which has led me directly to IDocument.

Eclipse uses the notion of resources for files. When an editor has a file open, it holds that file as a resource. An editors resource is called a document. It implements the interface IDocument. This means that I can attach a resource listener, which then gives me access to the source code. Whenever the resource is updated the listeners gets notified. This way I can get the source code in realtime, with changes. Perfect for what I need!

5.3.1 Which Parts Do I Need?

Now that I can get the source code. It is time to think about which parts I need. If I execute the entire source unmodified, then I will not have anything to show in my plugin view. So I need to peek inside the source. Find the relevant spot and attach a connection.

Eclipse Java development tools (JDT)

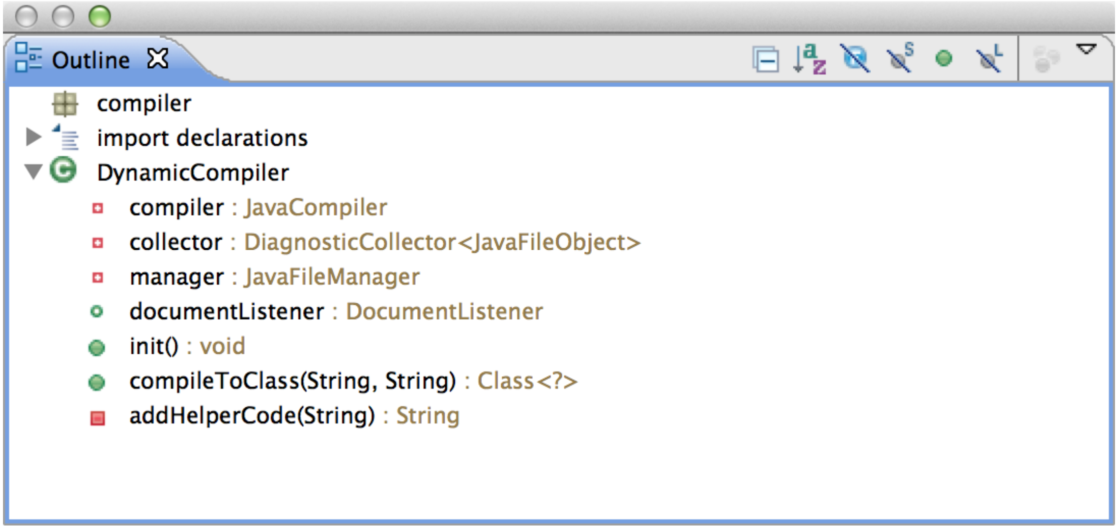
One of the most used plugins for Eclipse is the JDT. JDT is what turns Eclipse into a full featured Java IDE.

The JDT actually contains a project that resembles this project. Scrapbook pages. With Scrapbook pages you can write Java code snippets and execute them without thinking of imports, main methods etc. I like the idea and hope it will be developed further at some point.

In Eclipse (with JDT) you have an outline in the standard Java perspective. This outline shows a quick overview of your source code as you can see in figure 5.2 below. This outline is created using the JDT.

[1] Coding Community - `http://www.stackoverflow.com`

[2] Eclipse Plugin Development FAQ - `http://wiki.eclipse.org/Eclipse_Plug-in_Development_FAQ`



compiler
import declarations
DynamicCompiler
compiler : JavaCompiler
collector : DiagnosticCollector<JavaFileObject>
manager : JavaFileManager
documentListener : DocumentListener
init() : void
compileToClass(String, String) : Class<?>
addHelperCode(String) : String

Figure 5.2: The Outline View in Eclipse.

I want to visualize variable declaration statements. Statements such as shown here in figure 5.3.

```
int y = Array[2012];
String s = "mirror".toUpperCase();
```

Figure 5.3: Example of Variable Declaration Statements.

The outline in figure 5.2 is created using the JDT. JDT provides an API so that we can look deeper into the source code. This has the benefit that I do not have to parse the source myself. Instead I can use what the API hands me.

So can I use the JDT to get the statements I want? Well yes and no.

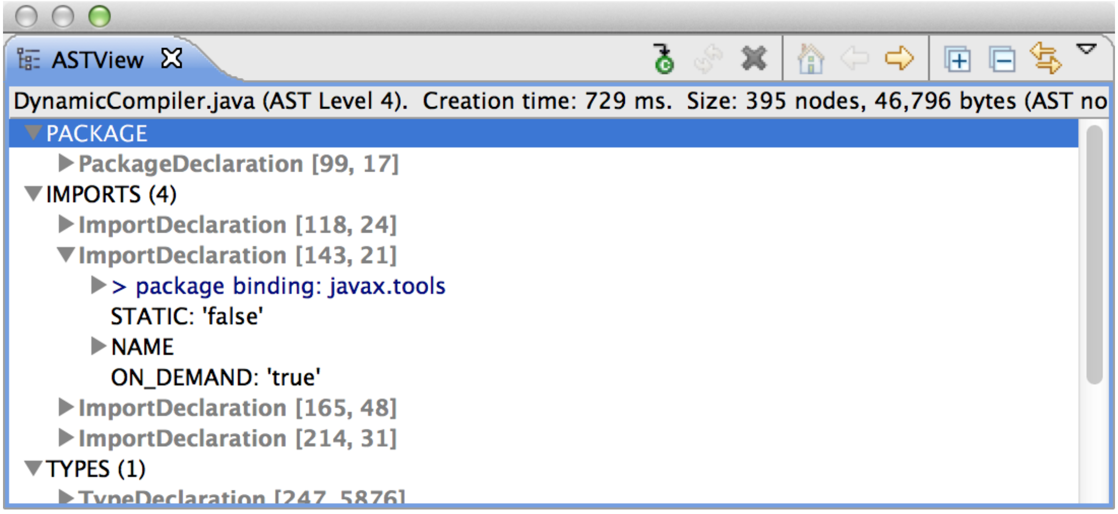
JDT has two different ways you can access the source code. The quick one, which the outline uses, can only get class level declaration of variables. So I cannot use the quick version if I want to show statements declared inside methods. Which I do want to support. The other way to access the source code is by creating an Abstract Syntax Tree (AST) from the source. This is much slower, but here I get the statements I need inside the method scope.

I will use both JDT and AST. I will use JDT to get the list of variables a method needs in order to execute. I will use the AST to get all the variable declaration statements inside a method. This is explained in the next section.

5.3.2 Using The AST

When creating an AST from source, Eclipse builds a tree of the source code. With the plugin AST View[3], you can see how your source is structured in an AST representation.

[3] http://marketplace.eclipse.org/content/ast-view



```
DynamicCompiler.java (AST Level 4).  Creation time: 729 ms.  Size: 395 nodes, 46,796 bytes (AST no
▼ PACKAGE
    ▶ PackageDeclaration [99, 17]
▼ IMPORTS (4)
    ▶ ImportDeclaration [118, 24]
    ▼ ImportDeclaration [143, 21]
        ▶ > package binding: javax.tools
          STATIC: 'false'
        ▶ NAME
          ON_DEMAND: 'true'
    ▶ ImportDeclaration [165, 48]
    ▶ ImportDeclaration [214, 31]
▼ TYPES (1)
    ▶ TypeDeclaration [247, 5876]
```

Figure 5.4: AST View in Eclipse.

Traversal

I have built a tree traverser for the AST's I am creating from the users source code. In this traverser I can decide which statements to support. The traverser visits each node of the AST. When it visits a variable declaration, it is added to a list of statements for that method.

I use this list to inject a method call after each statement. This method call sends a string representation of the variable to my plugin. I have chosen to share resources between the plugin and the editor via files. Since the plugin is running inside Eclipse, they are running on the same JVM. When I execute the users source code, that will run on another JVM. So I thought files would be an easy way to communicate betweens JVM's. That way I need know nothing about the JVM's. The files are created in the Eclipse application folder and are deleted on exit.

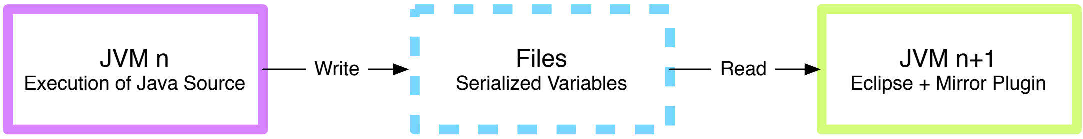


Figure 5.5: Communication between JVM's.

5.4 Type Limitations

I have thought about which types I should support. At first I wanted to support all types. Not really realistic in this project. Again in order to make the deadline I decided to focus my effort. I will support only primitive types in this first iteration of Mirror.

There are some problems with generics and primitive types. Primitive types are not objects and in Java anything that is used as generics has to be convertible to an object. Primitive types are special cases in Java, which means I cannot use the advantages of generics. Had I chosen to support Scala instead of Java this would have been easier. Because in Scala everything is an object.

5.5 Dynamic Compilation

I did not know much about the possibilities of doing dynamic compilation when I started this project. My gut feeling was, that it was fairly easy. I was wrong. I have looked at the following solutions to this problem. I will write briefly about each and explain why I ended up choosing the one I did.

- Janino - Super small Java compiler.
- Beanshell - Lightweight Scripting for Java.
- Java Compiler API - Interface to compile source files on the fly.

5.5.1 Janino

Janino was the first solution that turned up after my initial research on Google and Stackoverflow. Snippet of Janino's description from its homepage[4]

"Janino is a super-small, super-fast Java^{TM} compiler. Not only can it compile a set of source files to a set of class files like the JAVAC tool, but also can it compile a Java^{TM} expression, block, class body or source file in memory, load the bytecode and execute it directly in the same JVM."

Perfect, that sounded exactly like what I was looking for. So I downloaded Janino and followed their examples. I was quite hopeful that it would work at this point. I could use the script evaluator, which takes a method body and compiles it with the parameter values added. Turned out however that this approach cannot call methods outside its own scope. That was not what I was hoping for.

Before going onwards with Janino's class compiler I decided to look around.

5.5.2 Beanshell

I found numerous references to Beanshell online. All of them positive. So I decided to give scripting another go. I really like the way scripting works. In the same way that I like to play around in Scala's REPL. I was hoping that Beanshell would let me script code. And let me call code outside the method scope. Turned out that it was not possible.

I had a feeling that I needed the execution process to be as lightweight as possible. I knew that the Compiler API existed, but I was hoping for something that would let me execute small parts of code. Which both Janino and Beanshell can do. But they lack some features I need. I want a scope that is bigger than one method.

5.5.3 Java Compiler API

My hopes for an easy fix to the compilation problem got shot down. I was ready to look at Java's compiler API. Since it is an official tool from Oracle, there is a lot of good information available. I have selected a few classes from the API I would like to highlight.

- JavaCompiler - Used to create a compilation task
- CompilationTask - Has a call method which starts the actual compilation.

[4] `http://docs.codehaus.org/display/JANINO/Home`

- JavaFileObject - An abstraction of the Java source file.
- ToolProvider - Used to get the systems underlying compiler.

These are the essential classes which I have used for this project. By and large I appreciate the way the API is intended to be used. With one exception! I really wanted to do everything in memory. And the API has a strong focus on you making files for each compilationunit. I would prefer not to make any additional files.

Turns out that it was not a big problem. The JavaFileObject can be implemented in such a way that it never serializes anything to disk. When I was trying to implement this I found an article by Swaminathan Bhaskar[5] which talks about how to do dynamic compilation in memory using the compiler API. I wrote an email to Bhaskar to thank him for his article. And also to ask permission to use his code in this project. He was kind enough to let me use his code. So the dynamic compilation is based on Bhaskar's code which I have modified a bit to fit the project.

Compiler Of Choice

I ended up choosing the Java Compiler API introduced in Java 6. I could perhaps have chosen Janino or Beanshell's class compilers as well. But the Compiler API has the added benefit that I don't need to redistribute anything, because it is built into Java. Had I chosen Janino or Beanshell, I would have to include them in the plugin distribution.

5.5.4 Modifying Source Code

After choosing a compiler I can now put it to work. For Mirror to function properly I need to add a method call after each variable declaration statement. This call will take the variable and turn it into a string. For arrays I have made an array decoration method which creates a string from an array. For other types I am dependent on the objects own toString method.

This extra code is added in my DynamicCompiler class. You can see a flow of the source code in figure 5.6 below.

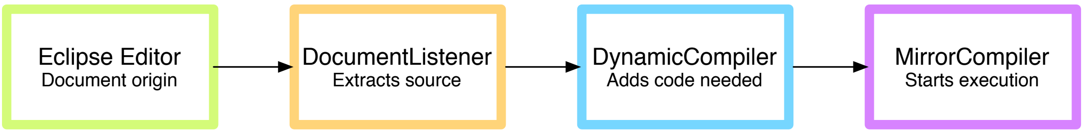


Figure 5.6: Flow of Source Code in Mirror.

5.6 Design Overview

I will give you a top level view of Mirror in figure 5.7. It shows how the plugin is built. I will give you some information on each entity to assist the figure.

- **Eclipse 3.7 Indigo** - The Eclipse SDK.
  This is the Eclipse application that the plugin runs in.
- **MirrorView** - The plugins main class.
  This class is the first to get instantiated by Eclipse. It has a reference to the plugin view.

[5] [Dynamic Code Generation with Java Compiler API in Java 6]

- **Plugin View** - The UI.
  This is the part of Mirror that the user sees and interacts with.

- **IEditor** - An editor in Eclipse.
  For Mirror to work, the editor has to be working on Java source code.

- **PartListener** - Listens to changes in the Eclipse UI.
  Detects when an editor gets opened. If the source is Java, Mirror is started.

- **IDocument** - The source the editor is working on.
  Lets me attach a listener to the source.

- **DocumentListener** - Listens to changes in the source code.
  When the source has changes, it sends the code to MirrorCompiler.

- **Files** - Serialized variables.
  Used to communicate between JVM's.

- **Input** - Input to method variables.
  For a method to execute it needs concrete values. The user either types them or they get randomly generated.

- **InputHandler** - Input parser.
  Translates the input to the correct primitive types needed for compilation. Saves values in memory for each method's input.

- **MirrorCompiler** - Executes the source.
  Creates the AST and the Compiler, and starts the execution of the users source code.

- **Compiler API & Helpers** - Collection of Compiler API classes.
  Adds the method calls needed by Mirror to the source. Also compiles the source.

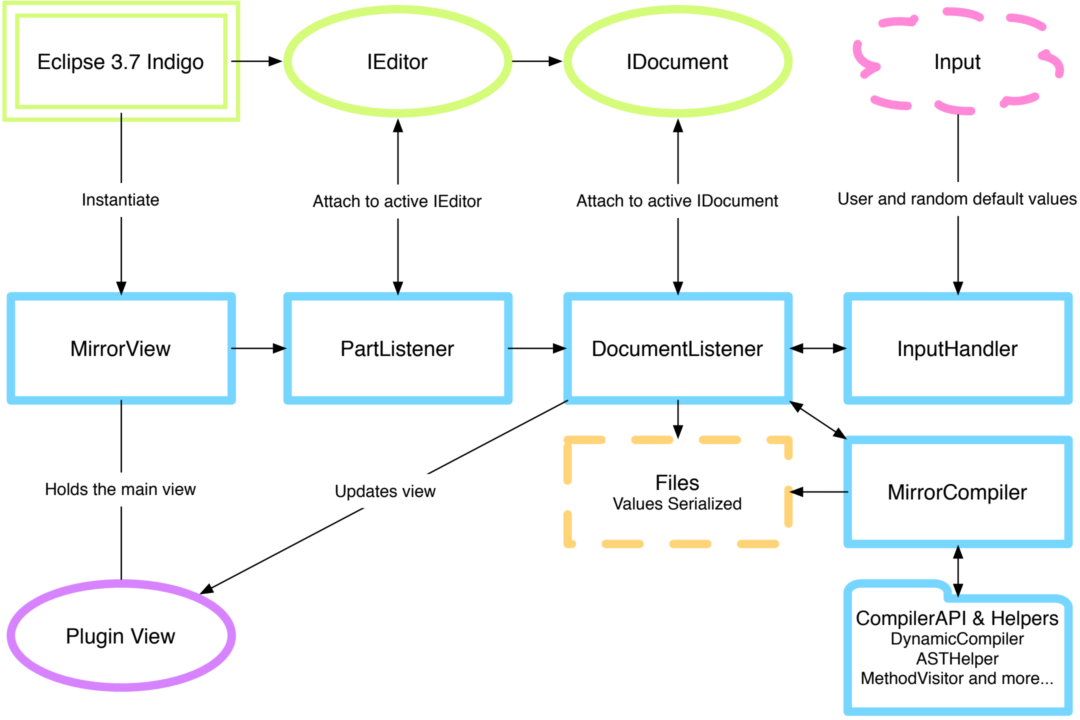


Figure 5.7: Design Overview of Mirror.

5.7 Community

The internet has reacted to Bret Victor's presentation. Just as I did. He struck a chord with many people. Here are two of the most interesting things that have come online since his talk at CUSEC[6].

- Chris Granger's Kickstarter Project Light Table - a reactive work surface for the creation and exploration of our programs
`http://www.kickstarter.com/projects/ibdknox/light-table`

- Jonathan Edward's critique of the talk - An IDE is not enough
`http://alarmingdevelopment.org/?p=680`

Granger's project has been a crowd funding success. He has raised money for the development of a new kind of IDE on Kickstarter[7]. Ligth Table both tries to do the instant visualization I am looking at, and it also tries to drop the concept of files. A very interesting project indeed.

Edward's blog post on the other hand talks about the problems with an approach like Granger's and mine. His conclusion is that you cannot just put a new IDE on top of a language which was not designed for it. New languages needs to be made for it to solve anything more than factorial functions.

[6] Canadian University Software Engineering Conference

[7] Funding platform for creative projects - http://www.kickstarter.com/

I think they both have interesting views. My personal addition to the debate is that we should always strive to make the best possible tools. We do not need only one application that can do all things. Nor do we need only one language. When making music, I am using many applications together. Applications that each do their thing. I would not mind to have many IDE's, each doing something particularly good.

I have made my work available on the popular code sharing community site, Github. You can find Mirror here `https://github.com/mofus/Mirror`. Maybe someone can help make it better in the future.

Chapter 6

Guide And Example

This chapter will show how to use Mirror in Eclipse. You need to have the correct version (3.7) of Eclipse installed[1]. You also need the Scala IDE features in Eclipse[2]. Finally you need Mirror[3].

I will demonstrate using a simple method that takes an array of integers.

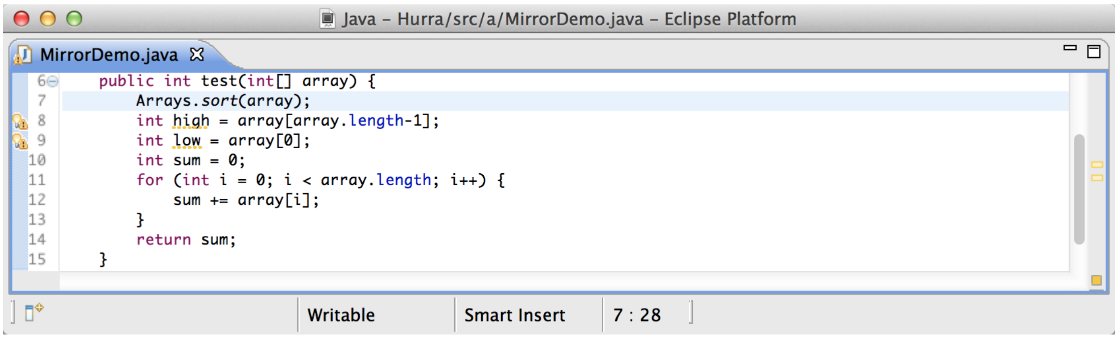
```
public int test(int[] array) {
        Arrays.sort(array);
        int high = array[array.length-1];
        int low = array[0];
        int sum = 0;
        for (int i = 0; i < array.length; i++) {
            sum += array[i];
        }
        return sum;
    }
```

The method is a sort of utility method for an array. It finds the lowest and highest values in the array. And it returns the sum of all the integers in the array. This type of method should show the strength of Mirror because of the many variable declaration statements. Open Eclipse and type this method into an editor. In figure 6.1 you can see how this should look like.

[1] Eclipse 3.7 - http://www.eclipse.org/downloads/packages/release/indigo/sr2

[2] Scala IDE - http://scala-ide.org/download/current.html

[3] Mirror - https://github.com/mofus/Mirror



```
    public int test(int[] array) {
        Arrays.sort(array);
        int high = array[array.length-1];
        int low = array[0];
        int sum = 0;
        for (int i = 0; i < array.length; i++) {
            sum += array[i];
        }
        return sum;
    }
```

Figure 6.1: Java editor in Eclipse.

Next open the list of available plugins for Eclipse (Window->Show View->Other...). Select Mirror View from the list.

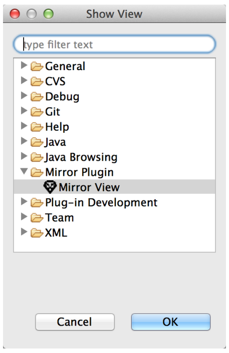
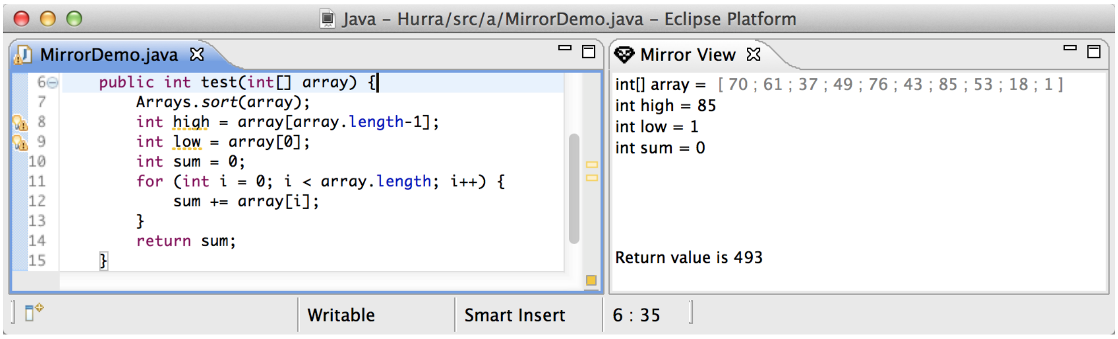


Figure 6.2: Available Plugins.

Now your Eclipse should look like figure 6.3 here. Mirror is running.



```
    public int test(int[] array) {
        Arrays.sort(array);
        int high = array[array.length-1];
        int low = array[0];
        int sum = 0;
        for (int i = 0; i < array.length; i++) {
            sum += array[i];
        }
        return sum;
    }
```

```
int[] array = [ 70 ; 61 ; 37 ; 49 ; 76 ; 43 ; 85 ; 53 ; 18 ; 1 ]
int high = 85
int low = 1
int sum = 0

Return value is 493
```

Figure 6.3: Java Editor and Mirror. Mirror compiles and executes whenever the source changes.

Our test method takes an int array as the only variable. You will notice that Mirror automatically generates a random array with 10 integers the first time you place the caret inside a method. If you wish to make your own array or modify the generated array, click the gray representation of the array in Mirror's view. If you wanted to make an array like `[ 1 ; 2 ; 3 ; 4 ; 5]` you would just type `1;2;3;4;5` in the field (and hit enter).

As you will see when playing with this method, the variable sum is 0. This is because the value is sent to Mirror just after it has been declared. And as you can see it is declared as being zero. Notice however that the method returns sum. Here the value is as expected at this point in the code, that is the sum of the array.

This example shows the potential of the project. Imagine a beginner learning about arrays. I think even something as simple as this could ease the learning curve.

Chapter 7

Further Work

Because of my limited time for the project there is many things I did not have time to do.

- Matching line numbers, so variables in the source view would align with the same variables in Mirror view. Maybe I could even decorate the editor instead of having a separate view.
- I only support one level of nesting in arrays. This is a side effect of not being able to use generics. Because primitives are a special case, I need to know what is at each level of the array. More levels could easily be implemented, but in the current form that would be way too much boilerplate code. So I am sticking to one level of nesting before the underlying design is better.
- I use the caret position to calculate which method the user is in. But that position does not work if anything gets folded in the editor.
- Method input types could be expanded to support all types.
- Iterations are not visualized right now. It would be nice to be able to see what happens at each iteration in a loop. A loop is another statement in the JDT. I restricted myself to do variable declaration statements in this first version. And quite frankly, I did not have the time to implement further statement types when I got the basics working.
- I would like to have better use of Scala. Some, None etc. which could make the code better and safer.
- Another approach could be to modify the compiler. An option would be to get the Open JDK and tamper with the included compiler. Were I to redo this project with the knowledge I have gained, I would try that approach.

Chapter 8

Conclusion

Is it possible to create an immediate connection between the creator and creation for programmers? I still think so. But it is slightly more a hope than a strong belief after doing this project. Starting out I thought that I could write a prototype plugin that would be further evolved by the end of this project. I underestimated the workload and complexity of the underlying problems. I believe those problems are rooted in the language designs. Most of the languages we use today are designed to be used in a text editor.

When I have talked to people about the project there has generally been two kinds of reaction. The first is "Wow, that sounds so cool and helpful!". The other is "So you are building a debugger?". The last reply seems to be favored by computer scientists. As the project has progressed I have gotten a sense for what it has become. Were I forced to put it in a category, I would also label it a debugger. A realtime, simplified, debugger.

Is this project solving any of the deeper problems? Definitely not. Does this project show that there is a need for realtime connection, debugging, visualisation (call it what you want)? I sure think it does!

I sadly have to conclude that making a project like this work for an existing language like Java is not easy. Although it can work for some hand picked cases, it is not nearly as effective and versatile as I had hoped for. For it to work properly the visualization needs to be a part of the language design.

This project has taken the form of discovery and learning, much more than programming a product. I am glad it did. I feel that the knowledge I have gained will make me a better computer scientist.

Looking at the goals from the project description on page 5, I have missed one. I did not manage to create a graphical visualization. What was the most successful part of this project for me? The execution at each keystroke! This alone starts to resemble the immediate connection I was hoping for. It gives me a playground where I quickly can try different things and instantly see the outcome. This is something I would have loved to have when I started programming.

So can we do something to ease the learning curve of starting programmers? Yes. This project has presented at least one way to do it.

Bibliography

[R.E.A.L] Report by Eliott Bartley
University College Dublin, 2009

[Code Complete, Second Edition] Book by Steve McConnell
Microsoft Press, 2009

[Art of Doing Science and Engineering: Learning to Learn] Book by Richard W. Hamming
Taylor & Francis, 2004

[Scala for the Impatient] Book by Cay Horstmann
Addison-Wesley Professional, 2012

[Inventing On Principle] Video talk by Bret Victor
CUSEC, 2012
`http://vimeo.com/36579366`

[Extending Eclipse - Plug-in Development Tutorial] Tutorial by Lars Vogel
Vogella, 2012
`http://www.vogella.com/articles/EclipsePlugIn/article.html`

[Dynamic Code Generation with Java Compiler API in Java 6] Article by S. Bhaskar
2009
`http://www.polarsparc.com/pdf/JavaCompiler.pdf`

[Eclipse Platform Technical Overview] Report by an anonymous writer
International Business Machines Corp (IBM), 2006
`http://eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf`

Appendix A

Source Code

The following pages contain the source code from Mirror.

```java
package mirror;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.console.MessageConsoleStream;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

    // The plug-in ID
    public static final String PLUGIN_ID = "Mirror"; //$NON-NLS-1$

    // The shared instance
    private static Activator plugin;

    public static MessageConsoleStream out;

    /**
     * The constructor
     */
    public Activator() {

    }

    /*
     * (non-Javadoc)
     *
     * @see
     * org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.BundleContext
     * )
     */
    public void start(BundleContext context) throws Exception {
        super.start(context);
        plugin = this;
    }

    /*
     * (non-Javadoc)
     *
     * @see
     * org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext
     * )
     */
    public void stop(BundleContext context) throws Exception {
        plugin = null;
        super.stop(context);
    }

    /**
     * Returns the shared instance
     *
     * @return the shared instance
     */
    public static Activator getDefault() {
        return plugin;
    }

    /**
     * Returns an image descriptor for the image file at the given plug-in
     * relative path
     *
     * @param path
     *            the path
     * @return the image descriptor
     */
    public static ImageDescriptor getImageDescriptor(String path) {
        return imageDescriptorFromPlugin(PLUGIN_ID, path);
    }
}
```

```java
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror;

import java.io.*;
import java.util.*;

import org.eclipse.jdt.core.dom.MethodInvocation;

// Helper class for handling files and adding uncertain types to method invocations
public class ASTHelper {
    @SuppressWarnings("unchecked")
    public static void argAdder(MethodInvocation me, Object o, Object name) {
        List<Object> list = new ArrayList<Object>();
        list.add(o);
        list.add(name);
        me.arguments().addAll(list);
    }

    public static String readFile(String file) throws IOException {
        // Create the file needed for communication
        File mirrorFile = new File(file);
        if (!mirrorFile.exists())
            mirrorFile.createNewFile();
        mirrorFile.deleteOnExit();
        BufferedReader reader = new BufferedReader(new FileReader(file));
        String line = null;
        StringBuilder stringBuilder = new StringBuilder();
        String ls = System.getProperty("line.separator");

        while ((line = reader.readLine()) != null) {
            stringBuilder.append(line);
            stringBuilder.append(ls);
        }
        return stringBuilder.toString();
    }
}
```

```scala
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror

import scala.collection.mutable.ArrayBuffer
import org.eclipse.jdt.core._
import org.eclipse.jdt.core.dom._
import org.eclipse.jface.text._
import org.eclipse.swt.SWT
import org.eclipse.swt.events._
import org.eclipse.swt.widgets._
import org.eclipse.ui.IEditorPart

class DocumentListener extends IDocumentListener {
  var document: IDocument = null
  var group: Composite = null
  var inputHandler: InputHandler = null
  val compiler = new MirrorCompiler()
  compiler.documentListener = this
  var editor: IEditorPart = null
  var unit: ICompilationUnit = null
  var packageName: String = null
  var className: String = null
  var parameters: ArrayBuffer[Object] = null
  var errorLabel: Label = null
  var errorMessage = ""
  var returnLabel: Label = null
  var returnMessage = ""
  var y = 0
  var caretPosition = 0
  var ar: Array[Statement] = null

  // React to changes in the source code from the editor
  def documentAboutToBeChanged(event: DocumentEvent) = {}
  def documentChanged(event: DocumentEvent) = {
    compile
  }

  def update: Int = {
    if (methodName != null) {
      // Dispose old text in the group view
      for (child <- group.getChildren)
        child.dispose

      parameters = new ArrayBuffer[Object]

      // Set up labels and inputs for each parameter a method takes
      y = 0
      for (input <- getMethodInputs(unit)) {
        val inputLabel = new Label(group, SWT.NONE)
        inputLabel setLocation (0, y)
        inputLabel setText input + " = "
        inputLabel pack
        val point = inputLabel getSize
        val inputValue = new Text(group, SWT.SINGLE)
        inputValue setLocation (point.x, y)
        inputValue setSize (group.getSize.x - point.x, inputValue.getLineHeight)

        if (inputHandler.allowedInput(input)) {
          // Check if there is saved any values for the input
          var message = inputHandler.savedInputs.get(input + methodName)
          if (message != None) {
            inputValue setMessage message.get.toString
            parameters += inputHandler objectFromString (input, methodName, message.get.toString)
          } else {
            val x = inputHandler randomObjectFromString (input, methodName)
            inputValue setMessage x._2
            parameters += x._1
            /* Userinput welcome below disabled in favor of random generated input
            inputValue setMessage "Set the value for " + input + " here"
            if (inputHandler.isInputArray(input))
              inputValue setMessage inputValue.getMessage + " (use ; as separater)" */
          }

          // When the textfield gets focused, set the saved value
          inputValue addFocusListener (new FocusListener() {
            def focusGained(event: FocusEvent) {
              if (message != None) {
                inputValue setText message.get.toString
              }
            }

            // Save the value the user has typed when the focus is lost
            def focusLost(event: FocusEvent) {
              if (inputValue.getText != "")
                parameters += inputHandler objectFromString (input, methodName, inputValue getText)
            }
          })

          inputValue addListener (SWT.KeyDown, new Listener() {
            def handleEvent(event: Event) = {
              // Save the value the user has typed when the user presses the enter key, update the display and run the code
              if (event.keyCode == 13) {
                parameters += inputHandler objectFromString (input, methodName, inputValue getText)
                update
                compile
              }
            }
          })
        } else
          inputValue setText "unsupported type."

        // Add to the y value, so that the possible next input box will be below the previous
        y += point.y
      }

      // Add vars from parsing AST
      compiler.startParsing(unit, methodName)
      var i = 0
      for (v <- ar) {
        var text = ""
        if (v.isInstanceOf[VariableDeclarationStatement]) {
          val stmt = v.asInstanceOf[VariableDeclarationStatement]
          val name = stmt.fragments.get(0).asInstanceOf[VariableDeclarationFragment].getName
          val result = ASTHelper.readFile(name.toString)
          text = stmt.getType + " " + name.toString + " = " + result.replace("\n", "")
        } else if (v.isInstanceOf[ExpressionStatement]) {
//          val stmt = v.asInstanceOf[ExpressionStatement]
//          text = stmt.toString
        }

          val inputLabel = new Label(group, SWT.NONE)
          inputLabel setLocation (0, y)
          inputLabel setText text
          inputLabel pack

          y += inputLabel.getSize.y
          i += 1
      }
    }

    // Add label for printing error message
    errorLabel = new Label(group, SWT.NONE)
    errorLabel setText errorMessage
    errorLabel setLocation (0, group.getSize.y - 18)
    errorLabel pack

    // Add label for printing return value
    returnLabel = new Label(group, SWT.NONE)
    returnLabel setText returnMessage
    returnLabel setLocation (0, group.getSize.y - 18 - errorLabel.getSize.y)
    returnLabel pack

    // Required to return some type, because I'm calling this method recursively
    0
  }

  // Error message clearing and setting
  def clearErrorMessage() = errorMessage = ""
  def setErrorMessage(s: String) = {
    val index = s indexOf className + ".java"
    errorMessage = "Line " + (s substring (index + className.length + ".java".length))
  }

  // Return message clearing and setting
  def clearReturnMessage() = returnMessage = ""
  def setReturnMessage(s: String) = {
    returnMessage = "Return value is " + s
    returnLabel setText returnMessage
    returnLabel pack
  }

  // Get the current caret position in the source file
  //  def caretPosition = editor.getAdapter(classOf[Control]).asInstanceOf[StyledText].getCaretOffset

  def getMethodInputs(unit: ICompilationUnit) = {
    val inputs = new ArrayBuffer[String]
    val allTypes = unit.getAllTypes
    for (itype <- allTypes) {
      val methods = itype.getMethods
      for (method <- methods) {
        if (method.getElementName.equals(methodName)) {
          val parameters = method.getParameters
          for (parameter <- parameters) {
            inputs += Signature.toString(parameter.getTypeSignature) + " " + parameter.getElementName
          }
        }
      }
    }
    inputs
  }

  def methodName = {
    val types = unit.getAllTypes
    var name: String = null;
    for (itype: IType <- types) {
      val methods = itype.getMethods
      for (method <- methods) {
        if (caretPosition >= method.getSourceRange.getOffset && caretPosition <= method.getSourceRange.getLength + method.getSourc
          name = method.getElementName
        }
      }
    }
    name
  }

  // Compile and run the current method
  def compile = {
    compiler.compile(document.get, (packageName + "." + className), methodName, parameters.toArray, unit)
  }

  def dispose(): Unit = {
    document.removeDocumentListener(this)
  }
}
```

```scala
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror

import scala.Array.canBuildFrom
import scala.collection.mutable.HashMap

class InputHandler {
  val savedInputs = new HashMap[String, String]
  val r = new scala.util.Random

  def objectFromString(parameterName: String, methodName: String, userInput: String) = {
    savedInputs.put(parameterName + methodName, userInput)

    // Look for letters andgGet the basic type. int, double etc
    val lettersRegEx = """[A-Za-z]+""".r
    val baseType = lettersRegEx.findFirstIn(parameterName).get.toLowerCase

    // Look for nested types - have to escape the first [ even though it is in 3 quotes
    val nestingRegEx = """\[[]]""".r

    // How many arrays are nested
    val depth = nestingRegEx.findAllIn(parameterName).length

    // Regexs to clean up the input & split the userinput so it becomes iterable
    val arraySplitterRegEx = """;""".r
    val removeChars = """\[|\]""".r
    var values = arraySplitterRegEx.split(removeChars.replaceAllIn(userInput, ""))
    if (values.length == 0)
      values = Array(userInput)

    if (depth <= 1)
      valueFromInput(values, depth, baseType).asInstanceOf[Object]
    else
      null
  }

  def valueFromInput(value: Array[String], depth: Int, typeString: String): Any = (depth, typeString) match {
    case (0, "byte") => value(0).trim.toByte
    case (1, "byte") => value.map(_.trim.toByte)
    case (0, "short") => value(0).trim.toShort
    case (1, "short") => value.map(_.trim.toShort)
    case (0, "int") => value(0).trim.toInt
    case (1, "int") => value.map(_.trim.toInt)
    case (0, "long") => value(0).trim.toLong
    case (1, "long") => value.map(_.trim.toLong)
    case (0, "float") => value(0).trim.toFloat
    case (1, "float") => value.map(_.trim.toFloat)
    case (0, "double") => value(0).trim.toDouble
    case (1, "double") => value.map(_.trim.toDouble)
    case (0, "char") => value(0).trim.toInt.toChar
    case (1, "char") => value.map(_.trim.toInt.toChar)
    case (0, "string") => value(0)
    case (1, "string") => value
    case (0, "boolean") => boolFromString(value(0).trim)
    // If it's a bool array, I can't use my helper method as it will get the wrong type
    case (1, "boolean") => value.map(_.trim.toBoolean)
  }

  def boolFromString(s: String) = s.toLowerCase match {
    case "true" => true
    case "false" => false
    case "yes" => true
    case "no" => false
    case "1" => true
    case "0" => false
    case _ => false
  }

  def allowedInput(parameterName: String) = {
    // Look for nested types - have to escape the first [ even though it is in 3 quotes
    val nestingRegEx = """\[[]]""".r

    // How many arrays are nested
    val depth = nestingRegEx.findAllIn(parameterName).length

    if (depth <= 1)
      true
    else
      false
  }

  def isInputArray(parameterName: String) = {
    // Look for nested types - have to escape the first [ even though it is in 3 quotes
    val nestingRegEx = """\[[]]""".r

    // How many arrays are nested
    val depth = nestingRegEx.findAllIn(parameterName).length

    for (i <- 1 to 10)
      println(i)

    if (depth == 1)
      true
    else
      false
  }

  def randomObjectFromString(parameterName: String, methodName: String) = {
    // Look for letters andgGet the basic type. int, double etc
    val lettersRegEx = """[A-Za-z]+""".r
    val baseType = lettersRegEx.findFirstIn(parameterName).get.toLowerCase

    // Look for nested types - have to escape the first [ even though it is in 3 quotes
    val nestingRegEx = """\[[]]""".r

    // How many arrays are nested
    val depth = nestingRegEx.findAllIn(parameterName).length

    if (depth <= 1) {
      val x = randomValueFromType(depth, baseType).asInstanceOf[Object]
      val s = TypeDecorator.stringRepresentation(x)
      savedInputs.put(parameterName + methodName, s)
      (x, s)
    } else
      null
  }

  def randomValueFromType(depth: Int, typeString: String): Any = (depth, typeString) match {
    case (0, "byte") => r.nextInt(100).toByte
    case (1, "byte") => for (i <- 1 to 10 toArray) yield r.nextInt(100).toByte
    case (0, "short") => r.nextInt(100).toShort
    case (1, "short") => for (i <- 1 to 10 toArray) yield r.nextInt(100).toShort
    case (0, "int") => r.nextInt(100)
    case (1, "int") => for (i <- 1 to 10 toArray) yield r.nextInt(100)
    case (0, "long") => r.nextLong
    case (1, "long") => for (i <- 1 to 10 toArray) yield r.nextLong
    case (0, "float") => r.nextFloat
    case (1, "float") => for (i <- 1 to 10 toArray) yield r.nextFloat
    case (0, "double") => r.nextDouble
    case (1, "double") => for (i <- 1 to 10 toArray) yield r.nextDouble
    case (0, "char") => r.nextInt(100).toChar
    case (1, "char") => for (i <- 1 to 10 toArray) yield r.nextInt(100).toChar
    case (0, "string") => r.nextString(3)
    case (1, "string") => for (i <- 1 to 10 toArray) yield r.nextString(3)
    case (0, "boolean") => r.nextBoolean
    case (1, "boolean") => for (i <- 1 to 10 toArray) yield r.nextBoolean
  }
}
```

```scala
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror

import scala.collection.mutable.ArrayBuffer
import org.eclipse.jdt.core.dom._

class MirrorMethodVisitor extends ASTVisitor {
  var methodName: String = null
  val declarations = new ArrayBuffer[Statement]

  override def visit(node: MethodDeclaration) = {
    if (node.getName.toString.equals(methodName)) {
      for (s <- node.getBody.statements.toArray) {
        if (s.isInstanceOf[VariableDeclarationStatement]) {
          val x = s.asInstanceOf[VariableDeclarationStatement]
          declarations += x
        } else if (s.isInstanceOf[ExpressionStatement]) {
          // Disabling more statements for now.
//          val x = s.asInstanceOf[ExpressionStatement]
//          declarations += x
        }
      }
    }
    super.visit(node)
  }
}
```

```scala
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror

import compiler.DynamicCompiler
import org.eclipse.jdt.core.ICompilationUnit
import org.eclipse.jdt.core.dom._
import org.eclipse.jface.text.Document
import scala.collection.mutable.ArrayBuffer
import java.io.File
import java.util.ArrayList
import java.util.Collection
import org.eclipse.jdt.core.dom.rewrite.ASTRewrite
import java.util.List

class MirrorCompiler() {
  var documentListener: DocumentListener = null
  var modifiedSource: Document = null;
  var varDeclStmts: Array[Statement] = null
  var parser: CompilationUnit = null

  def compile(source: String, className: String, methodName: String, parameters: Array[Object], unit: ICompilationUnit) {
    // Parse the AST and get the variables
    startParsing(unit, methodName)

    // Add the code needed for retrieving values
    rewrite(unit)

    // Create a dynamic compiler and get it ready
    val compiler = new DynamicCompiler
    compiler.documentListener = documentListener
    compiler init

    val c = compiler.compileToClass(className, modifiedSource.get)

    // Create an object
    val o = c.newInstance

    // Get the correct method (both public and private)
    val m = c.getDeclaredMethods find { x => x.getName == methodName }
    val method = m.get

    // Only compile if the correct amount of parameters are given
    if (method.getParameterTypes.length == parameters.length) {

      // If the method was private override that accessibility
      method.setAccessible(true)

      // Invoke the method on the object - parameters needs to be mapped to support varargs like behavior
      val returnValue = method.invoke(o, parameters.map(_.asInstanceOf[Object]): _*)

      if (returnValue != null)
        documentListener.setReturnMessage(returnValue.toString)

      documentListener.update
    }
  }

  def rewrite(unit: ICompilationUnit) {
    val ast = parser.getAST
    val rewriter = ASTRewrite.create(ast)

    // Get insertion position
    val typeDecl = parser.types().get(0).asInstanceOf[TypeDeclaration]
    for (m <- typeDecl.getMethods()) {
      if (m.getName.toString.equals(documentListener.methodName)) {
        // Which line in the block should the extra calls be inserted at?
        var i = 1
        for (v <- varDeclStmts) {
          if (v.isInstanceOf[VariableDeclarationStatement]) {
            val stmt = v.asInstanceOf[VariableDeclarationStatement]
            val block = m.getBody

            // create new statements for insertion
            val stringMethodCall = ast.newMethodInvocation
            stringMethodCall.setName(ast.newSimpleName("stringRepresentation"))

            // Get the object and the name of the object and add that as arguments to the methodcall
            val name = stmt.fragments.get(0).asInstanceOf[VariableDeclarationFragment].getName.toString
            val nameSL = ast.newStringLiteral
            nameSL.setLiteralValue(name)

            // This bit needs to be done in Java because Scala doesn't like the type uncertainty
            ASTHelper.argAdder(stringMethodCall, ast.newSimpleName(name), nameSL)
            val newStatement = ast.newExpressionStatement(stringMethodCall)

            // Insert the new code and apply the edits
            val listRewrite = rewriter.getListRewrite(block, Block.STATEMENTS_PROPERTY)
            listRewrite.insertAt(newStatement, listRewrite.getOriginalList.indexOf(v) + i, null)
            i += 1

            val edits = rewriter.rewriteAST()
            modifiedSource = new Document(unit.getSource())
            edits.apply(modifiedSource)
          }
        }
      }
    }
  }

  def startParsing(unit: ICompilationUnit, methodName: String) = {
    parser = parse(unit)
    val visitor = new MirrorMethodVisitor
    visitor.methodName = methodName
    parser.accept(visitor)
    documentListener.ar = visitor.declarations.toArray
    varDeclStmts = visitor.declarations.toArray
  }

  def parse(unit: ICompilationUnit): CompilationUnit = {
    val parser = ASTParser.newParser(AST.JLS3)
    parser.setKind(ASTParser.K_COMPILATION_UNIT)
    parser.setSource(unit)
    parser.setResolveBindings(true)
    parser.createAST(null).asInstanceOf[CompilationUnit] // parse
  }
}
```

```scala
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror

import org.eclipse.swt.SWT
import org.eclipse.swt.events._
import org.eclipse.swt.widgets._
import org.eclipse.ui.PlatformUI
import org.eclipse.ui.part.ViewPart

class MirrorView extends ViewPart {
  val ID = "mirror.views.MirrorView"
  var group: Composite = null
  val inputHandler = new InputHandler
  val partListener = new PartListener

  def createPartControl(parent: Composite): Unit = {
    // Create group view for the plugin & set it's background to white
    group = new Composite(parent, SWT.INHERIT_DEFAULT)
    val white = Display.getDefault.getSystemColor(SWT.COLOR_WHITE)
    group setBackground (white)

    // Check if the editor is onscreen - if it's not create start button
    if (PlatformUI.getWorkbench.getActiveWorkbenchWindow.getActivePage != null)
      setup
    else {
      val startButton = new Button(group, SWT.PUSH);
      startButton.setText("Start MirrorView");
      startButton.addSelectionListener(new SelectionAdapter() {
        override def widgetSelected(e: SelectionEvent): Unit = {
          setup
          startButton.dispose
          val inputLabel = new Label(group, SWT.NONE)
          inputLabel setText "Now focus the editor"
          inputLabel pack
        }
      })
      startButton.pack
    }
  }

  def setup = {
    // Listen to the workbench for open and close events
    PlatformUI.getWorkbench.getActiveWorkbenchWindow.getActivePage.addPartListener(partListener)

    // Set the reference to the group view
    partListener.group = group
    partListener.inputHandler = inputHandler
  }

  // Method called whenever the view gets focused
  def setFocus(): Unit = {

  }

  // Removing the listener when the plug-in gets disposed
  override def dispose(): Unit = {
    super.dispose
  }

  def log(s: String) = Activator.out.println(s)
}
```

```scala
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror

import org.eclipse.core.resources.ResourcesPlugin
import org.eclipse.jdt.core._
import org.eclipse.swt.custom._
import org.eclipse.swt.widgets._
import org.eclipse.ui._
import org.eclipse.ui.texteditor.ITextEditor

class PartListener extends IPartListener2 with CaretListener {
  var group: Composite = null
  var inputHandler: InputHandler = null
  var caretOffset = 0
  val listener: DocumentListener = new DocumentListener

  def compilationUnitForDocument = {
    // Get the root of the workspace
    val workspace = ResourcesPlugin.getWorkspace
    val root = workspace.getRoot
    // Get all projects in the workspace
    val projects = root.getProjects

    var compilationUnit: ICompilationUnit = null
    var packageName: String = null
    var className: String = null

    // Loop over all projects
    for (project <- projects) {
      // Check if we have a Java project
      if (project.isNatureEnabled("org.eclipse.jdt.core.javanature")) {
        val javaProject = JavaCore.create(project)
        val packages = javaProject.getPackageFragments
        for (mypackage <- packages) {
          // Check if it is a source file
          if (mypackage.getKind() == IPackageFragmentRoot.K_SOURCE) {
            for (unit <- mypackage.getCompilationUnits) {
              val page = PlatformUI.getWorkbench.getActiveWorkbenchWindow.getActivePage
              val activeEditor = page.getActiveEditor
              // Check if the source code is in the active editor
              if (unit.getElementName().equals(activeEditor.getTitle())) {
                packageName = mypackage.getElementName
                compilationUnit = unit
                // Remove .java from the unit to get the classname
                className = unit.getElementName.substring(0, (unit.getElementName.length - 5))
              }
            }
          }
        }
      }
    }
    (compilationUnit, packageName, className)
  }

  def partActivated(partRef: IWorkbenchPartReference): Unit = {
    // Check if the part activated is a Java source file
    val title = partRef.getTitle.toLowerCase
    if (title.endsWith(".java")) {
      // Get the current editor
      val activeEditor = PlatformUI.getWorkbench.getActiveWorkbenchWindow.getActivePage.getActiveEditor
      if (activeEditor.isInstanceOf[ITextEditor]) {

        // Get the source code from the editor
        val document = activeEditor.asInstanceOf[ITextEditor].getDocumentProvider.getDocument(activeEditor.getEditorInput)

        // Listen to caret events and update after an event
        activeEditor.getAdapter(classOf[Control]).asInstanceOf[StyledText].addCaretListener(this)

        // Give the listener the correct references
        listener.document = document
        listener.group = group
        listener.editor = activeEditor
        val x = compilationUnitForDocument
        listener.unit = x._1
        listener.packageName = x._2
        listener.className = x._3
        listener.inputHandler = inputHandler

        // React to changes in the source
        document.addDocumentListener(listener)

        // Update the view with the currently loaded source code
        listener.update
      }
    }
  }

  def partClosed(partRef: IWorkbenchPartReference): Unit = {
    // Check if the part closed is a Java source file
    val title = partRef.getTitle.toLowerCase
    if (title.endsWith(".java")) {
      // Don't listen to changes in the document anymore
      listener.dispose
    }
  }

  def caretMoved(event: CaretEvent) {
    if (event.caretOffset != caretOffset) {
      listener.caretPosition = event.caretOffset
      listener.update
      caretOffset = event.caretOffset
    }
  }

  // Unused methods inherited from interface
  def partBroughtToTop(partRef: IWorkbenchPartReference): Unit = {}
  def partDeactivated(partRef: IWorkbenchPartReference): Unit = {}
  def partOpened(partRef: IWorkbenchPartReference): Unit = {}
  def partHidden(partRef: IWorkbenchPartReference): Unit = {}
  def partVisible(partRef: IWorkbenchPartReference): Unit = {}
  def partInputChanged(partRef: IWorkbenchPartReference): Unit = {}
}
```

```java
/* (c) 2012 Anders Bech Mellson - anbh@itu.dk */

package mirror;

// Helperclass that can decorate types, eg turn an array into a pretty string.
public class TypeDecorator {
    public static String stringRepresentation(Object s) {
        if (s.getClass().isArray())
            return arrayDecorator(s);
        else
            return s.toString();
    }

    private static String arrayDecorator(Object a) {
        String decoratedString = "[ ";
        if (int[].class == a.getClass()) {
            int[] b = (int[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (double[].class == a.getClass()) {
            double[] b = (double[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (float[].class == a.getClass()) {
            float[] b = (float[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (byte[].class == a.getClass()) {
            byte[] b = (byte[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (short[].class == a.getClass()) {
            short[] b = (short[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (long[].class == a.getClass()) {
            long[] b = (long[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (char[].class == a.getClass()) {
            char[] b = (char[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (String[].class == a.getClass()) {
            String[] b = (String[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        if (Boolean[].class == a.getClass()) {
            Boolean[] b = (Boolean[]) a;
            for (int i = 0; i < b.length; i++) {
                String separator = i < b.length - 1 ? " ; " : "";
                decoratedString += b[i] + separator;
            }
        }
        decoratedString += " ]";
        return decoratedString;
    }
}
```

```java
/*
 * Name: Bhaskar S
 *
 * Date: 10/10/2009
 *
 * Modified by Anders Bech Mellson 23.07.12
 */

package compiler;

import java.util.Arrays;
import javax.tools.*;
import javax.tools.JavaCompiler.CompilationTask;
import mirror.DocumentListener;

public class DynamicCompiler {
    private JavaCompiler compiler;
    private DiagnosticCollector<JavaFileObject> collector;
    private JavaFileManager manager;
    public DocumentListener documentListener;

    public void init() throws Exception {
        compiler = ToolProvider.getSystemJavaCompiler();
        collector = new DiagnosticCollector<JavaFileObject>();
        manager = new DynamicClassFileManager<JavaFileManager>(
                compiler.getStandardFileManager(null, null, null));
    }

    public Class<?> compileToClass(String fullName, String javaCode)
            throws Exception {
        Class<?> clazz = null;
        javaCode = addHelperCode(javaCode);
        StringJavaFileObject strFile = new StringJavaFileObject(fullName,
                javaCode);
        Iterable<? extends JavaFileObject> units = Arrays.asList(strFile);
        CompilationTask task = compiler.getTask(null, manager, collector, null,
                null, units);
        boolean status = task.call();
        if (status) {
//            System.out.printf("Compilation successful!!!\n");
            clazz = manager.getClassLoader(null).loadClass(fullName);
            documentListener.clearErrorMessage();
        } else {
            for (Diagnostic<?> d : collector.getDiagnostics()) {
                System.out.printf(d.getMessage(null));
                documentListener.clearReturnMessage();
                documentListener.setErrorMessage(d.getMessage(null));
            }
//            System.out.printf("***** Compilation failed!!!\n");
        }

        return clazz;
    }

    // Helper source code needed in every class to represent the values
    private String addHelperCode(String source) {
        String sourceCode = "public void stringRepresentation(Object s, String name) {\n"
                + "        try {\n"
                + "            PrintWriter out = new PrintWriter(name);\n"
                + "            if (s.getClass().isArray())\n"
                + "                out.println(arrayDecorator(s));\n"
                + "            else\n"
                + "                out.println(s);\n"
                + "            out.close();\n"
                + "        } catch (FileNotFoundException e) {\n"
                + "        }\n"
                + "    }\n"
                + "\n"
                + "    public static String arrayDecorator(Object a) {\n"
                + "        String decoratedString = \"[ \";\n"
                + "        if (int[].class==a.getClass()) {\n"
                + "            int[] b = (int[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (double[].class==a.getClass()) {\n"
                + "            double[] b = (double[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (float[].class==a.getClass()) {\n"
                + "            float[] b = (float[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (byte[].class==a.getClass()) {\n"
                + "            byte[] b = (byte[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (short[].class==a.getClass()) {\n"
                + "            short[] b = (short[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (long[].class==a.getClass()) {\n"
                + "            long[] b = (long[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (char[].class==a.getClass()) {\n"
                + "            char[] b = (char[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (String[].class==a.getClass()) {\n"
                + "            String[] b = (String[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        if (Boolean[].class==a.getClass()) {\n"
                + "            Boolean[] b = (Boolean[])a;\n"
                + "            for (int i = 0; i < b.length; i++) {\n"
                + "                String separator = i < b.length - 1 ? \" ; \" : \"\";\n"
                + "                decoratedString += b[i] + separator;\n"
                + "            }\n"
                + "        }\n"
                + "        decoratedString += \" ]\";\n"
                + "        return decoratedString;\n" + "    }\n";

        String importSource = "\n" + "import java.io.FileNotFoundException;\n"
                + "import java.io.PrintWriter;\n";
        // Add the imports needed for the above source code to work
        int i = 0, index = 0;
        for (String s : source.split("package.*?;")) {
            if (i < 2) {
                i++;
                index = source.indexOf(s);
            } else
                break;
        }
        source = source.substring(0, index) + importSource
                + source.substring(index);

        // Add the extra source code before the last }
        index = source.lastIndexOf('}');
        source = source.substring(0, index) + sourceCode
                + source.substring(index);
        return source;
    }
}
```

```java
/*
 * Name: Bhaskar S
 *
 * Date: 10/10/2009
 */

package compiler;

import java.util.*;

public class ByteArrayClassLoader extends ClassLoader {
    private Map<String, ByteArrayJavaFileObject> cache = new HashMap<String, ByteArrayJavaFileObject>();

    public ByteArrayClassLoader() throws Exception {
        super(ByteArrayClassLoader.class.getClassLoader());
    }

    public void put(String name, ByteArrayJavaFileObject obj) {
        ByteArrayJavaFileObject co = cache.get(name);
        if (co == null) {
            cache.put(name, obj);
        }
    }

    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        Class<?> cls = null;

        try {
            ByteArrayJavaFileObject co = cache.get(name);
            if (co != null) {
                byte[] ba = co.getClassBytes();
                cls = defineClass(name, ba, 0, ba.length);
            }
        } catch (Exception ex) {
            throw new ClassNotFoundException("Class name: " + name, ex);
        }

        // System.out.printf("Method findClass() called for class %s\n", name);

        return cls;
    }
}
```

```java
/*
 * Name: Bhaskar S
 *
 * Date: 10/10/2009
 */

package compiler;

import java.io.*;
import java.net.URI;
import javax.tools.SimpleJavaFileObject;

public class ByteArrayJavaFileObject extends SimpleJavaFileObject {
    private final ByteArrayOutputStream bos = new ByteArrayOutputStream();

    public ByteArrayJavaFileObject(String name, Kind kind) {
        super(
                URI.create("string:///" + name.replace('.', '/')
                        + kind.extension), kind);
    }

    public byte[] getClassBytes() {
        return bos.toByteArray();
    }

    @Override
    public OutputStream openOutputStream() throws IOException {
        return bos;
    }
}
```

```java
/*
 * Name: Bhaskar S
 *
 * Date: 10/10/2009
 */

package compiler;

import java.io.IOException;
import javax.tools.*;
import javax.tools.JavaFileObject.Kind;

public class DynamicClassFileManager<FileManager> extends
        ForwardingJavaFileManager<JavaFileManager> {
    private ByteArrayClassLoader loader = null;

    DynamicClassFileManager(StandardJavaFileManager mgr) {
        super(mgr);
        try {
            loader = new ByteArrayClassLoader();
        } catch (Exception ex) {
            ex.printStackTrace(System.out);
        }
    }

    @Override
    public JavaFileObject getJavaFileForOutput(Location location, String name,
            Kind kind, FileObject sibling) throws IOException {
        ByteArrayJavaFileObject co = new ByteArrayJavaFileObject(name, kind);
        loader.put(name, co);
        return co;
    }

    @Override
    public ClassLoader getClassLoader(Location location) {
        return loader;
    }
}
```

```java
/*
 * Name: Bhaskar S
 *
 * Date: 10/10/2009
 * Modified by Anders Bech Mellson 23.07.12
 */

package compiler;

import java.net.URI;
import javax.tools.SimpleJavaFileObject;

public class StringJavaFileObject extends SimpleJavaFileObject {
    private String source;

    public StringJavaFileObject(String name, String source) {
        super(URI.create("string:///" + name.replace('.', '/')
                + Kind.SOURCE.extension), Kind.SOURCE);
        this.source = source;
    }

    @Override
    public CharSequence getCharContent(boolean ignoreEncodingErrors) {
        return this.source;
    }
}
```