Remove Duplicate Code in Refactoring Documentation

Joseph Gonzalez, Andrew Otto, Will Parkin, Jon Tedesco

Contents

1 Remove Duplicate Code In Constructors Refactoring Description 2

2 Refactoring Examples 2
2.1 Using an Existing Master Constructor 2
2.2 Creation of a Master Constructor 3
2.3 Choosing Access Modifier 4
2.4 Using a Helper Function 5

3 UI Componenet 5

4 Code Summary 6
4.1 Refactoring 6
4.2 Test Suite 7

5 Future Plans 7

6 Personal Reflections 8

Appendices 9
.1 Installation 9
.2 How to Run Refactoring 9
.3 How to Run Tests 9

1 Remove Duplicate Code In Constructors Refactoring Description

Remove Duplicate Code In Constructors (RDCIC) intends to remove code duplication across multiple constructors. Duplication is defined when multiple constructors assign to the same class field. To remove duplication in each constructor, there are three possible remedies:

1. The constructor calls an existing constructor, delegating it's field assignments to the existing master constructor[1] (MC).
2. A MC is created, and all existing constructors delegate work to this new MC.
3. A helper function is created to do the work of a MC.

If we obtain a set that is the intersection of all fields assigned in all constructors, we can define the presence of duplicate code as:

- There is an overlap in field assignments between any constructors.
  OR
- There is no constructors that assigns to the described set of fields.
  OR
- A current constructor call does not exist in every single constructor.

The refactoring will fail in any of the following cases:

1. There are less than 2 constructors.
2. Any constructor contains compile errors.
3. No duplicate code exists, as described above.
4. There exists, in any constructor with duplicate code, a field assignment with dependencies on previous statements.
5. When duplicate code exists, a constructor exists that matches the signature of a master constructor but does not qualify for a master constructor because of its body.

2 Refactoring Examples

Below are some examples of the RDCIC refactoring.

2.1 Using an Existing Master Constructor

The following code:

[1]A master constructor assigns to the set of fields that is the union of all fields assigned in any constructor within the class.

```java
public class Example {

        int x;
        double y;

        public Example() {
                this.x = 0;
                this.y = 0.0;
        }

        public Example(int x) {
                this.x = x;
                this.y = 0.0;
        }

        public Example(int x, double y) {
                this.x = x;
                this.y = y;
        }
}
```

turns into:

```java
public class Example {

        int x;
        double y;

        public Example() {
                this(0, 0.0);
        }

        public Example(int x) {
                this(x, 0.0);
        }

        public Example(int x, double y) {
                this.x = x;
                this.y = y;
        }
}
```

2.2 Creation of a Master Constructor

The following code:

```java
public class Test {

        int x;
        double y;

        public Test() {

        }

        public Test(int x) {
                this.x = x;
        }

        public Test(double y) {
                this.y = y;
        }
}
```

will turn into this after RDCIC refactoring:

```
public class Test {

        int x;
        double y;

        public Test() {
                this(0, 0);

        }

        public Test(int x) {
                this(x, 0);
        }

        public Test(double y) {
                this(0, y);
        }

        public Test(int x, double y) {
                this.x = x;
                this.y = y;
        }
}
```

Note that a new master constructor was created.

2.3 Choosing Access Modifier

The user can also specify the access modifier of the new constructor. The following code

```
public class TestAccessPrivate {

        int x;
        boolean y;

        public TestAccessPrivate(boolean y) {
                this.y = y;
        }

        public TestAccessPrivate(int x) {
                this.x = x;
        }
}
```

turns into

```
public class TestAccessPrivate {

        int x;
        boolean y;

        public TestAccessPrivate(boolean y) {
                this(0, y);
        }

        public TestAccessPrivate(int x) {
                this(x, false);
        }

        private TestAccessPrivate(int x, boolean y) {
                this.x = x;
                this.y = y;
        }
}
```

2.4 Using a Helper Function

If a new master constructor is needed, the user can choose to use a helper method instead. The following code

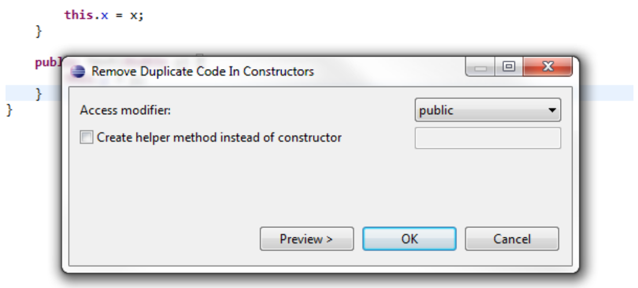
```
class TestHelperTwoFields {

        int a;
        boolean b;

        public TestHelperTwoFields(boolean y) {
                this.b = y;
        }

        public TestHelperTwoFields(int x) {
                this.a = x;
        }
}
```

turns into

```
class TestHelperTwoFields {

        int a;
        boolean b;

        public TestHelperTwoFields(boolean y) {
                helper(0, y);
        }

        public TestHelperTwoFields(int x) {
                helper(x, false);
        }

        public void helper(int a, boolean b) {
                this.a = a;
                this.b = b;
        }
}
```

3 UI Componenet

The plugin is controlled by a simple wizard GUI launched from the refactoring menu, as shown in Figure-1. The wizard allows changing settings for two optional features of the refactoring. In addition, access is given to the built-in preview feature of the refactoring engine.



```
        this.x = x;
    }

    publ
    }
}
```

Remove Duplicate Code In Constructors

Access modifier: public

☐ Create helper method instead of constructor

Figure 1: Refactoring wizard.

The access modifier menu, Figure-2 lets the user choose which access modifier to apply to the new method. The wizard gives the user the choice in a drop menu between the access modifiers `public`, `private`, and `protected`.

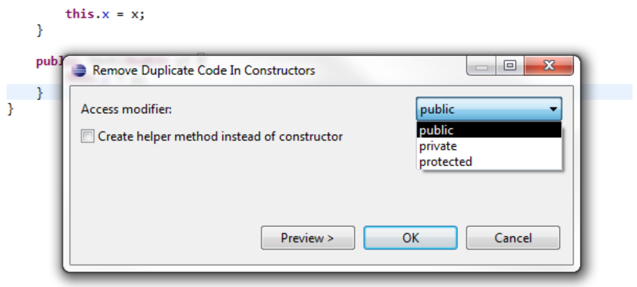

Remove Duplicate Code In Constructors
Access modifier:
public
public
private
protected
Create helper method instead of constructor
Preview >
OK
Cancel

Figure 2: Access modifier menu.

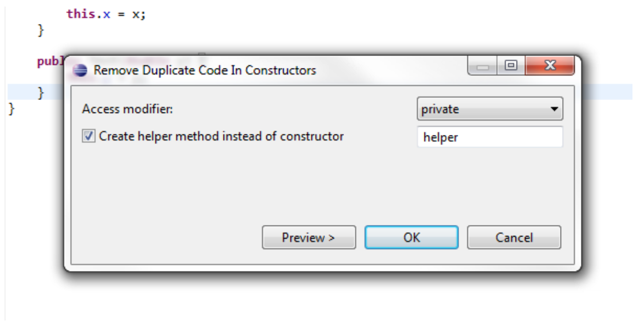
When the "Create helper method instead of constructor" box is checked (Figure-3), the text box is enabled and the user can input the desired name for the helper function that will be created instead of the MC. Invalid method names (names starting with numbers or names containing characters besides 'a-z', 'A-Z', '_', or '0-9') and existing method names are not allowed and prevent the wizard from proceeding. The helper method's access modifier may also be set, but the return type is always void.

4 Code Summary

4.1 Refactoring

Our core refactoring source code consists of the refactoring itself, the UI components for the refactoring, utility methods, and visitors, and is broken down into three packages. The full refactoring source documentation was autogenerated using Doxygen, and can be found *here*. Likewise, autogenerated documentation for the full test suite can be found *here*.

1. Core & UI (`edu.illinois.canistelCassabanana`)
   - This package contains the core refactoring source, including the refactoring itself, the eclipse UI components, and boiler plate code to launch the refactoring source.
2. Utilities (`edu.illinois.canistelCassabanana.utility`)



Access modifier: private
[✓] Create helper method instead of constructor: helper

Figure 3: Creating a helper method instead of a master constructor.

- Contains modular, reusable utility functions that do not require any state. These functions include type lookup functions, finding constructors from a compilation unit and vice versa, and additional shared functions.

3. Visitors (`edu.illinois.canistelCassabanana.visitors`)

   - Contains visitors that traverse the AST tree to retrieve particular types of AST nodes, including assignments, constructor declarations & invocations, field variables, and type declarations. We chain these visitors to perform tasks such as finding all field variables appearing in assignments inside of constructors.

4.2 Test Suite

Our test suite thoroughly tests possible situations for our refactoring to handle. Our tests fall into two broad categories, namely invalid and valid tests, where we expect the refactoring to fail and succeed, respectively. Within each category, we have additional categories for tests.

1. Invalid Tests
   - Test classes that contain no constructors
   - Test classes that contain only one constructor
   - Test classes that contain compile error in constructors
   - Test classes that contain no duplicate code in constructors
   - Test classes where assigned field contain dependencies
   - Test classes where user parameters are invalid
2. Valid Tests
   - Test classes that contain two constructors
   - Test classes that contain three constructors
   - Test classes where new constructor must be created
   - Test classes where existing constructor must be used
   - Test classes with user-provided parameters (helper method & custom access modifiers)

Auto-generated documentation is likewise available for the refactoring test suite.

5 Future Plans

For our project, we have big plans for the future. Our project has already been uploaded to Github, in separate public repositories for the plugin source and test source. We will be shortly packaging the plugin for download using Github's web hosting.

After this course, we plan to add further functionality to the plugin as well. Specifically, we have the following features in mind:

1. Better UI Integration
   - Allow users to trigger the refactoring from the editor, rather than only from the Eclipse refactoring menu.

2. Better Dependency Analysis
   - Improve dependency analysis for field assignments, perhaps by inlining earlier lines or forcing use of a helper method if dependencies are detected.
3. Add Inheritance Analysis
   - Expand duplicate code analysis to deal with type hierarchies, so that constructors can eliminate duplicate code by making super constructor calls if possible.
4. Allow Users to Explicitly Add Unused Fields
   - Currently, our refactoring ignores unused fields in the class, but we should allow users to explicitly include unused fields in the refactoring if they so choose.

6 Personal Reflections

Andrew

This was my first time programming with more than one other person, so it was an educational experience for me. I've also had very little experience speaking technically, so it often felt difficult communicating my thoughts to the group. It was especially tough when it seemed like I had jumped halfway into an unfinished task and then had to try and acquaint myself with what they were doing so that I could help. The project itself was also frustrating at times, particularly when we had some type of ASTnode and were trying to find a particular piece of information from it without creating an abomination made of function calls and casts. In the end, however, it was good practice and I feel that I improved at least slightly both in communication and actual programming skill.

Joe

Learning how to create a refactoring in Eclipse has been a very tedious and mind numbing process. Throughout the process there were times where the the refactoring classes provided in the tutorials made absolutely no sense, times where the concept made sense but the implementation not so much, and times where we breezed through code. Needless to say it has been a learning experience especially in terms of reverse engineering which was one of the goals of the assignment. It is also gratifying to know that our refactoring actually does what it was intended to do and that I mostly understand how.

Jon

Overall, I enjoyed this project, and look forward to seeing where we can take it in the future. I think this project was challenging, but rewarding to work with a project like Eclipse. Challenging, because most documentation is out of date and incomplete, the source code is huge, and examples are difficult to find, but rewarding because you get to see your project working in an elegant framework that millions of developers use regularly. I hope that with some additional work, our project will be added to the Eclipse source. For our project, I found the logistics the most challenging component of all. It was difficult to manage an entire team's school schedules to find times to meet, and frustrating to find computers that would actually run the project we were developing. I can see why pair programming would be effective in the real world, and in spite of the difficulties we faced, I found the experience rewarding.

Will

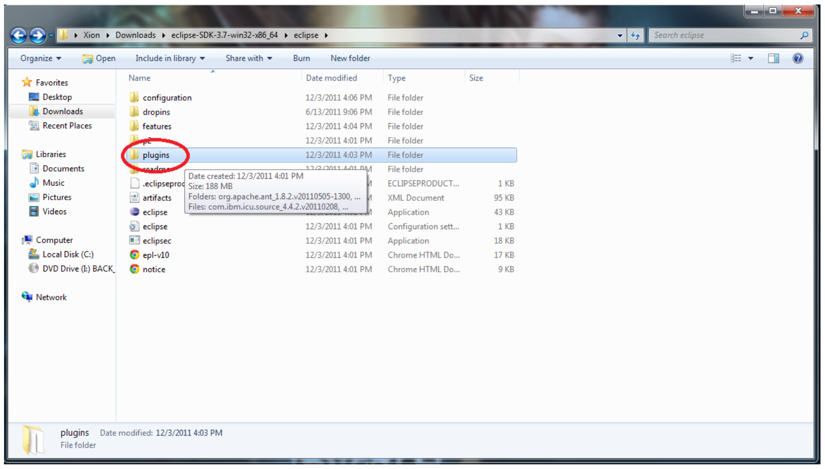
This was my first experience at software engineering, and overall, I enjoyed this project. It was tough figuring out how to implement our refactoring without worked-through examples. But, it was fun to see our project start with just a few user stories evolve into tests, then into working code. Our group did a great job of sticking to the principles of extreme programming. I enjoyed pair programming, because it having two sets of eyes helped when we are a team of student programmers. I think this refactoring is something I might actually use in the future. Overall, the project was good software engineering experience, and I look forward to improving our refactoring in the future.

Appendices

.1 Installation

The plugin can be downloaded from:
`https://github.com/downloads/jtedesco/RemoveDuplicateCodeFromConstructors/org.eclipse.refactoring.removeDuplicateCodeFromConstructors.jar`

To complete the installation, drop the downloaded `org.eclipse.refactoring.removeDuplicateCodeFromConstructors.jar` into the "plugins" folder of Eclipse, as shown in Figure-4.



Name	Date modified	Type	Size
configuration	12/3/2011 4:06 PM	File folder	
dropins	6/13/2011 9:06 PM	File folder	
features	12/3/2011 4:04 PM	File folder	
p2	12/3/2011 4:01 PM	File folder	
plugins	12/3/2011 4:03 PM	File folder	
readme	12/3/2011 4:01 PM	File folder	
.eclipseproc	[illegible]	ECLIPSEPRODUCT...	1 KB
artifacts	[illegible]	XML Document	95 KB
eclipse	[illegible]	Application	43 KB
eclipse	12/3/2011 4:01 PM	Configuration sett...	1 KB
eclipsec	12/3/2011 4:01 PM	Application	18 KB
epl-v10	12/3/2011 4:01 PM	Chrome HTML Do...	17 KB
notice	12/3/2011 4:01 PM	Chrome HTML Do...	9 KB

Date created: 12/3/2011 4:01 PM
Size: 188 MB
Folders: org.apache.ant_1.8.2.v20110505-1300, ...
Files: com.ibm.icu.source_4.4.2.v20110208, ...

plugins Date modified: 12/3/2011 4:03 PM
File folder

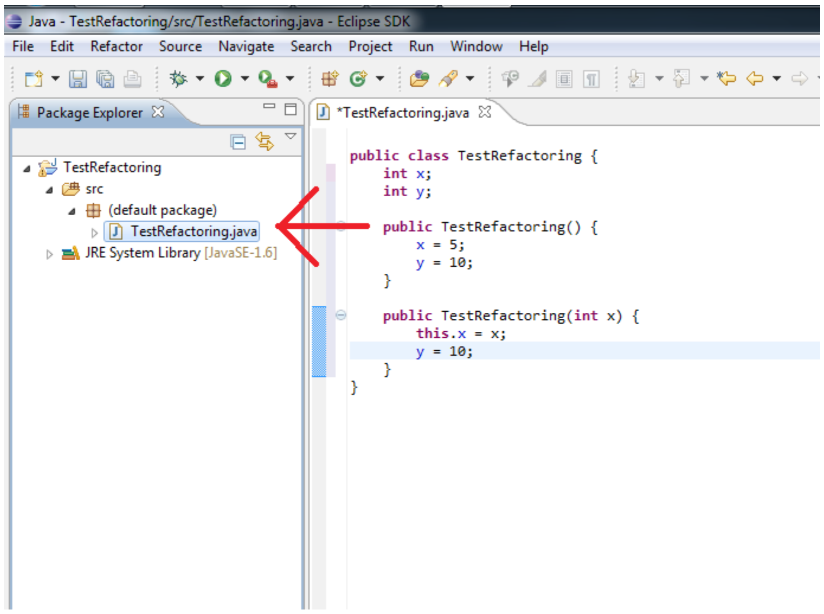
Figure 4: Install plugin by dropping .jar file into Eclipse "plugin" folder.

.2 How to Run Refactoring

Highlight/select the class in the Package Explorer that you would like to refactor, as seen in Figure-5 where `TestRefactoring.java` is highlighted. Next, select the "Refactor" menu from the top of Eclipse and select "Remove Duplicate Code in Constructors..." as seen in Figure-6.

.3 How to Run Tests

Check out Iteration 3 code from:
`https://subversion.ews.illinois.edu/svn/fa11-cs427/CanistelCassabanana/RefactoringProject/Iteration-3.0-Implementation/edu.illinois.canistelCassabanana`


Java - TestRefactoring/src/TestRefactoring.java - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
*TestRefactoring.java
TestRefactoring
src
(default package)
TestRefactoring.java
JRE System Library [JavaSE-1.6]

```java
public class TestRefactoring {
    int x;
    int y;

    public TestRefactoring() {
        x = 5;
        y = 10;
    }

    public TestRefactoring(int x) {
        this.x = x;
        y = 10;
    }
}
```

Figure 5: Select class to refactor in the package explorer.

and check out the tests from:
`https://subversion.ews.illinois.edu/svn/fa11-cs427/CanistelCassabanana/`
`RefactoringProject/Iteration-3.0-Tests/edu.illinois.canistelCassabanana.tests`

Once checked out, navigate to the `RemoveDuplicateCodeInConstructorsRefactoringTests.java` in the `edu.illinois.CanistelCassabanana.tests` project under `src/edu.illinois.canistelCassabanana.tes` as seen in Figure-7. Right click that particular class and do "Run As" → "JUnit Plug-in Test," as seen in Figure-8.

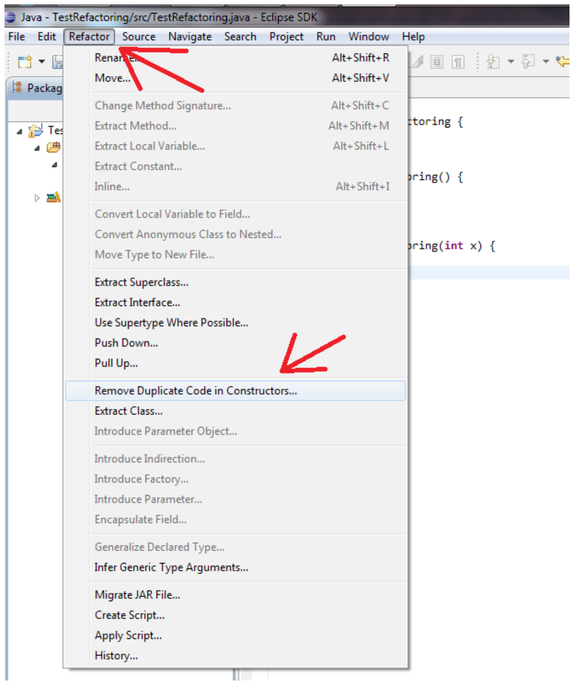
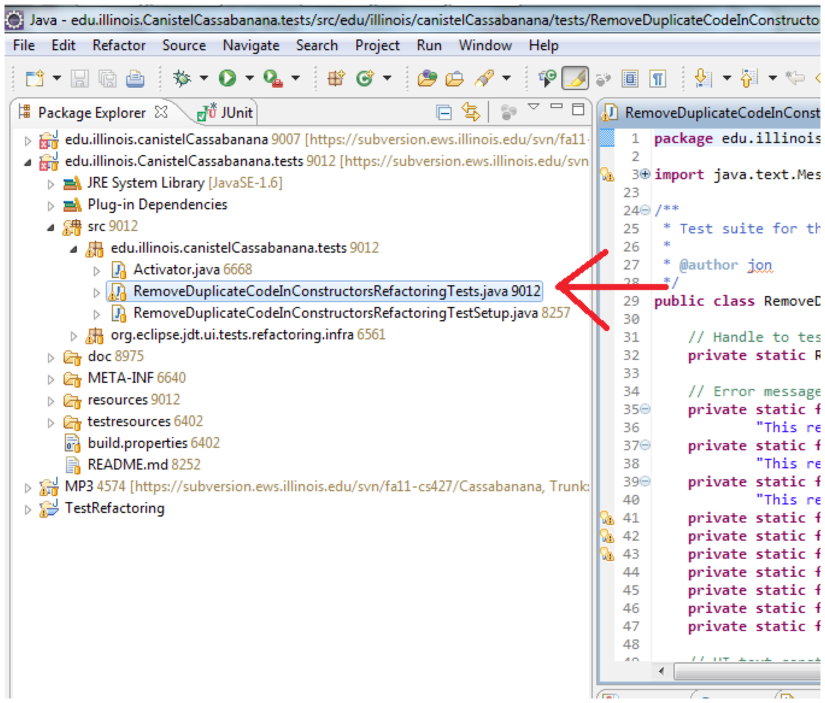
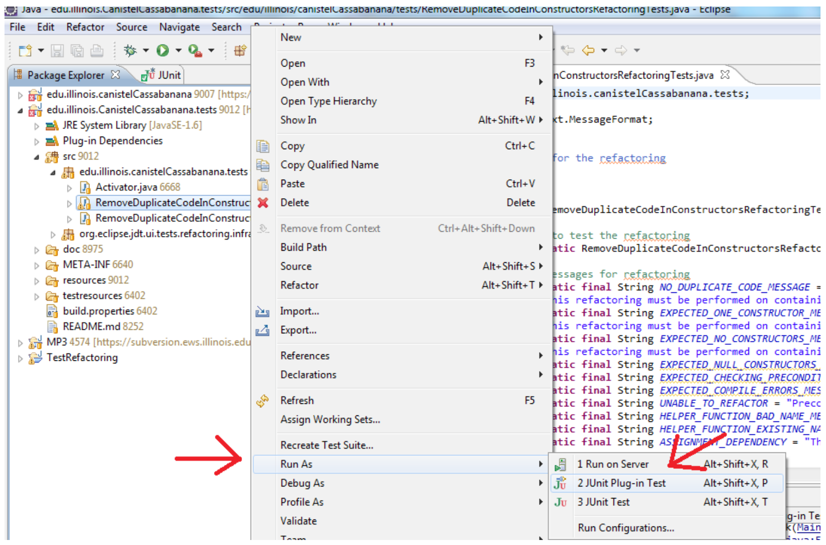

Java - TestRefactoring/src/TestRefactoring.java - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
Rename... Alt+Shift+R
Move... Alt+Shift+V
Change Method Signature... Alt+Shift+C
Extract Method... Alt+Shift+M
Extract Local Variable... Alt+Shift+L
Extract Constant...
Inline... Alt+Shift+I
Convert Local Variable to Field...
Convert Anonymous Class to Nested...
Move Type to New File...
Extract Superclass...
Extract Interface...
Use Supertype Where Possible...
Push Down...
Pull Up...
Remove Duplicate Code in Constructors...
Extract Class...
Introduce Parameter Object...
Introduce Indirection...
Introduce Factory...
Introduce Parameter...
Encapsulate Field...
Generalize Declared Type...
Infer Generic Type Arguments...
Migrate JAR File...
Create Script...
Apply Script...
History...

Figure 6: Select to run refactoring.


Java - edu.illinois.CanistelCassabanana.tests/src/edu/illinois/canistelCassabanana/tests/RemoveDuplicateCodeInConstructo
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
JUnit
edu.illinois.canistelCassabanana 9007 [https://subversion.ews.illinois.edu/svn/fa11
edu.illinois.CanistelCassabanana.tests 9012 [https://subversion.ews.illinois.edu/svn
JRE System Library [JavaSE-1.6]
Plug-in Dependencies
src 9012
edu.illinois.canistelCassabanana.tests 9012
Activator.java 6668
RemoveDuplicateCodeInConstructorsRefactoringTests.java 9012
RemoveDuplicateCodeInConstructorsRefactoringTestSetup.java 8257
org.eclipse.jdt.ui.tests.refactoring.infra 6561
doc 8975
META-INF 6640
resources 9012
testresources 6402
build.properties 6402
README.md 8252
MP3 4574 [https://subversion.ews.illinois.edu/svn/fa11-cs427/Cassabanana, Trunk
TestRefactoring
RemoveDuplicateCodeInConst

```
package edu.illinois

import java.text.Mes

/**
 * Test suite for th
 *
 * @author jon
 */
public class RemoveD

    // Handle to tes
    private static R

    // Error message
    private static f
            "This re
    private static f
            "This re
    private static f
            "This re
    private static f
    private static f
    private static f
    private static f
    private static f
    private static f
    private static f
```

Figure 7: File containing Remove Duplicate Code In Constructors Refactoring tests.


Java - edu.illinois.CanistelCassabanana.tests/src/edu/illinois/canistelCassabanana/tests/RemoveDuplicateCodeInConstructorsRefactoringTests.java - Eclipse
File Edit Refactor Source Navigate Search
Package Explorer
JUnit
edu.illinois.canistelCassabanana 9007
edu.illinois.CanistelCassabanana.tests 9012
JRE System Library [JavaSE-1.6]
Plug-in Dependencies
src 9012
edu.illinois.canistelCassabanana.tests
Activator.java 6668
RemoveDuplicateCodeInConstruc
RemoveDuplicateCodeInConstruc
org.eclipse.jdt.ui.tests.refactoring.infra
doc 8975
META-INF 6640
resources 9012
testresources 6402
build.properties 6402
README.md 8252
MP3 4574 [https://subversion.ews.illinois.edu
TestRefactoring
New
Open F3
Open With
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
References
Declarations
Refresh F5
Assign Working Sets...
Recreate Test Suite...
Run As
Debug As
Profile As
Validate
1 Run on Server Alt+Shift+X, R
2 JUnit Plug-in Test Alt+Shift+X, P
3 JUnit Test Alt+Shift+X, T
Run Configurations...
nConstructorsRefactoringTests.java

```
linois.canistelCassabanana.tests;

xt.MessageFormat;


for the refactoring



emoveDuplicateCodeInConstructorsRefactoringTe

to test the refactoring
atic RemoveDuplicateCodeInConstructorsRefacto

essages for refactoring
atic final String NO_DUPLICATE_CODE_MESSAGE =
his refactoring must be performed on containi
atic final String EXPECTED_ONE_CONSTRUCTOR_ME
his refactoring must be performed on containi
atic final String EXPECTED_NO_CONSTRUCTORS_ME
his refactoring must be performed on containi
atic final String EXPECTED_NULL_CONSTRUCTORS_
atic final String EXPECTED_CHECKING_PRECONDIT
atic final String EXPECTED_COMPILE_ERRORS_MES
atic final String UNABLE_TO_REFACTOR = "Prec
atic final String HELPER_FUNCTION_BAD_NAME_ME
atic final String HELPER_FUNCTION_EXISTING_NA
atic final String ASSIGNMENT_DEPENDENCY = "Th
```

Figure 8: Run JUnit Plug-in Tests.