Due 12/4/11

CS 427 Semester Project: Removing Duplicate Code in Constructors

CaimitoPapaya:
Austine Lakayil, Edie Liao, Nitesh Solanki, Vilius Zaikauskas

Table of Contents

Project Description

Project Goals

Often, constructors do little more than initializing class variables, and much of the code across constructors is repeated. What variation there is between multiple constructors of the same class is likely to be which particular variables the constructors initialize. Our refactoring removes initialization code that spans two or more constructors and relocates it to a master constructor, henceforth referred to as the "Megatron" constructor, that takes all parameters and initializes them with the associated code. Meanwhile, the existing code is replaced by a call to the "Megatron" constructor using the provided parameters and Java default values. In conclusion, this refactoring aims to reduce and unify repeated initialization code.

Implemented Refactorings

Constructor Parameter Disjoint Sets

If all the constructors contain unique parameters such that the intersection of all constructor parameter sets results in an empty set, then a "Megatron" constructor will be created that contains the union of all the constructor parameters, and their corresponding assignment statements.

Before

```
public class TestDisjointMegatronCreation_testMixedParamTypes {
    private int testVar1;
    private int testVar2;
    private double workingVar1;
    private String workingVar2;

    public TestDisjointMegatronCreation_testMixedParamTypes(int
inputVar1, int inputVar2) {
          this.testVar1 = inputVar1;
          this.testVar2 = inputVar2;
    }

    public TestDisjointMegatronCreation_testMixedParamTypes(double
workingInputVar1, String workingInputVar2) {
          this.workingVar1 = workingInputVar1;
          this.workingVar2 = workingInputVar2;
    }
}
```

After

```
public class TestDisjointMegatronCreation_testMixedParamTypes {
    private int testVar1;
    private int testVar2;
    private double workingVar1;
    private String workingVar2;
```

```
    public TestDisjointMegatronCreation_testMixedParamTypes(int
inputVar1, int inputVar2) {
        this(inputVar1, inputVar2, 0.0, null);
    }

    public TestDisjointMegatronCreation_testMixedParamTypes(double
workingInputVar1, String workingInputVar2) {
        this(0, 0, workingInputVar1, workingInputVar2);
    }

    public TestDisjointMegatronCreation_testMixedParamTypes(int
inputVar1, int inputVar2, double workingInputVar1, String
workingInputVar2) {
        this.testVar1 = inputVar1;
        this.testVar2 = inputVar2;
        this.workingVar1 = workingInputVar1;
        this.workingVar2 = workingInputVar2;
    }
}
```

Constructor Parameter Overlapping Sets

If constructor parameter sets overlap but are not a subset of one another, then a "Megatron" constructor will be created that contains the union of all the constructor parameters, and their corresponding assignment statements.

Before

```
public class  TestOverlap_MultipleOverlap {
    private int testVar1;
    private int testVar2;
    private double testVar3;
    private double testVar4;

    public TestOverlap_MultipleOverlap(int inputVar1, int
    inputVar2, double inputVar4){
        this.testVar1 = inputVar1;
        this.testVar2 = inputVar2;
        this.testVar4 = inputVar4;
    }
    public TestOverlap_MultipleOverlap(int inputVar2, double
    inputVar3, double inputVar4){
        this.testVar2 = inputVar2;
        this.testVar3 = inputVar3;
        this.testVar4 = inputVar4;
}
```

After

```
public class  TestOverlap_MultipleOverlap {
    private int testVar1;
    private int testVar2;
    private double testVar3;
```

```
    private double testVar4;

    public TestOverlap_MultipleOverlap(int inputVar1, int
    inputVar2, double inputVar4){
          this(inputVar1,inputVar2, 0.0, inputVar4);
    }

    public TestOverlap_MultipleOverlap(int inputVar2, double
    inputVar3, double inputVar4){
          this(0, inputVar2, inputVar3, inputVar4);
    }

    public TestOverlap_MultipleOverlap(int inputVar1, int
    inputVar2, double inputVar3, double inputVar4){
          this.testVar1 = inputVar1;
          this.testVar2 = inputVar2;
          this.testVar3 = inputVar3;
          this.testVar4 = inputVar4;
    }
}
```

Constructor Parameter Subsets

If a constructor parameter list contains the parameters of all the other constructors then this constructor is the "Megatron" constructor. The only change would be to create a constructor invocation in the non-Megatron constructors.

Before

```
public class TestSubset_OneOverlap {
    private int testVar1;
    private String testVar2;
    private double testVar3;
    private double testVar4;

    public TestOverlap_OneOverlap(int inputVar1, String inputVar2){
          this.testVar1 = inputVar1;
          this.testVar2 = inputVar2;
    }

    public TestOverlap_OneOverlap(int inputVar1, String inputVar2,
double inputVar3){
          this.testVar1 = inputVar1;
          this.testVar2 = inputVar3;
          this.testVar3 = inputVar3;
    }
}
```

After

```
public class TestSubset_OneOverlap {
    private int testVar1;
```

```
    private String testVar2;
    private double testVar3;
    private double testVar4;

    public TestOverlap_OneOverlap(int inputVar1, String inputVar2){
          this(inputVar1,inputVar2, 0.0, 0.0);
    }

    public TestOverlap_OneOverlap(int inputVar1, String inputVar2,
double inputVar3){
          this.testVar1 = inputVar1;
          this.testVar2 = inputVar3;
          this.testVar3 = inputVar3;
    }
}
```

Constructor Specific operations/assignments

If a constructor contains statements that do not use the input parameters, then these statements should be kept in the constructor but placed after the constructor invocation and not placed in the "Megatron" constructor.

Before

```
public class TestFunctionInsideConstructor {
    private int testVar1;
    private String testVar2;
    private double testVar3;
    private double testVar4;

    public int testFunc(){ return 0; }

    public TestFunctionInsideConstructor(int testVar1, String
testVar2) {
          this.testVar1 = testVar1;
          this.testVar2 = testVar2;
          testFunc();
    }

    public TestFunctionInsideConstructor(double testVar3, double
testVar4) {
          this.testVar3 = testVar3;
          this.testVar4 = testVar4;
    }
}
```

After

```
public class TestFunctionInsideConstructor {
    private int testVar1;
    private String testVar2;
    private double testVar3;
    private double testVar4;

    public int testFunc(){ return 0; }

    public TestFunctionInsideConstructor(int testVar1, String
testVar2) {
          this(testVar1, testVar2, 0.0, 0.0);
          testFunc();
    }

    public TestFunctionInsideConstructor(double testVar3, double
testVar4) {
          this(0, null, testVar3, testVar4);
    }

    public TestFunctionInsideConstructor(int testVar1, String
testVar2, double testVar3, double testVar4) {
          this.testVar1 = testVar1;
          this.testVar2 = testVar2;
          this.testVar3 = testVar3;
          this.testVar4 = testVar4;
    }
}
```

User Interface

The interface that was created placed a Refactoring Constructor option was added into the Refactor menu tab. This option automatically refactors the file selected in the Package Explorer.

Classes and Methods

Our most important method dealt with the refactoring. We had one method called refactor that was implemented in the RemoveDuplicationInConstructorsRefactoring class which was our main class, which uses the Refactoring API. The rest of the methods are simply helper methods to avoid stuffing everything into one method (such as union, intersection of lists). Furthermore, we have a visitor that finds the *TypeDeclaration* inside the *CompilationUnit* in which the refactoring would be done.

Future Plans

Project Future

This project has potential to become a very useful plug in. However, the project has long way to go to become fully functional. In the given time for the project the refactoring had to have limitations and handle a specific subset of all potential cases. These cases are stated below:

Refactoring Requirements

- Left hand side of all constructor assignment statements must be instance variables of the class.
- Left hand side of all constructor assignment statements must be Field_Declarations
- Right hand side of constructor assignment statements must be parameters or constants.
- Only lines in constructor that contain an input parameter on the right hand side of an assignment statement are moved to the "Megatron" constructor.
- Each line in the constructor must end in a semi-colon.
- No line of code in the constructor can span more than one line
- Class must contain <= 2 constructors before refactoring.

Personal Reflections

Austine Lakayil

More than anything, this assignment was a learning experience. We started out this project with a basic guideline of what to do, which I thought was a lot more robust and focused that it actually was. When we started this project, we were shown small examples in class and we had to find a way to implement the necessary ideas in a similar fashion. However, as the project went on, we received less and less help and the material we had to find/research/implement got harder and more difficult conceptually. While this experience is true to life, I felt that it was excessively hopeful of the teaching staff to expect us to finish within the amount of time they assumed it would take. XP programming was definitely a pleasure to use when we were on target, but I felt crunched on time and overexerted as soon as we were no longer meeting our goals. However, having said all of that, I'm glad I went through this experience because I feel like I learned a lot.

Edie Liao

Given our overall ability and the resources available to us as a class, I feel that a project of this proportion was a bit out of our reach. While we were taught basic principles of software engineering in class, it was difficult putting them into practice. Looking back, I really wish we could experience XP with a knowledgeable moderator, rather than having a group of clueless students blindly leading the blind. Having someone help us break down the steps of developing this project and guiding us through it, especially during the planning game, would have made this project much more manageable. I also would have preferred having a step-by-step tutorial and even starter code for the xml and UI portion of the project, seeing as xml was never covered in class and the project is so heavily dependent on it. In conclusion, I feel that the project would have taught us more if it were broken down into smaller chunks and we received more concrete, step-by-step

guidance during the time we worked on it. The one thing that I did feel was successful was writing tests before developing the code. Knowing what to code and attacking the problem with a top-down approach really makes sense to me, allowing us to understand and modularize the components of the program, distribute work within the group, and really shortening the amount of time we spent debugging. As the code became cleaner through refactoring, it became easier to understand, and easier for group members to catch up and pitch in.

Nitesh Solanki

This project surprised me in many ways. At first, I thought this project didn't have much to it, but the more I got into it the more I realized how much work this project entailed. For the most part coming up with solutions was not algorithmically difficult, but the syntax was time consuming. We ran into two major issues. The first was learning the AST. The second was learning how to add the UI. Overall, this experience taught me a lot about using the tools we have at hand, and learning how to understand technical documentation better.

Vilius Zaikauskas

I somewhat thought the project was a little bit dry, but maybe I'm still a "greenhorn" when it comes to experiencing what it's really like to be a Software Engineer. Often times I think of creating everything from scratch in hopes I could make something cool. It took a while to understand what the real nature of this project was and how to utilize all the given libraries to us. In fact, it took us a while to find out what kind of libraries existed, let alone utilize them. After finding them and 20 hours of playing around with them, everything made sense and the procedures of creating nodes or making deep copies of certain nodes became second nature. Although, I believe that the API could be improved upon due to some interfaces not being allowed to be implementable by clients. Then again, without knowing the lower level stuff, I don't know how much of an effort creators would have to put it. We worked long and hard, but we learned from this experience.

Appendix

Installation

- Check out illinois.edu.CaimitoPapaya
- Open illinois.edu.CaimitoPapaya and find the plugin.xml file
- Right-click on it and choose Run As.../Eclipse Application
- A new Eclipse window will open
- Open your project in the new instance of Eclipse

How to Run the Program

- Highlight the java file you wish to refactor in the package explorer
- Go to Refactor menu
- Click on Remove Duplicate Code In Constructors

How to Test the Program

- Check out illinois.edu.CaimitoPapaya.tests
- Go to illinois.edu.CaimitoPapaya.tests
- Go to src/edu.illinois.CaimitoPapaya.refactorings.tests
- Right-click RemoveCodeDuplicationInConstructorsTests.java
- Choose Run as.../JUnit Test