Prequestionnaire

Programming Experience

1. How long have you been programming?

◯ Less than 1 year

◯ 1 - 2 years

◯ 2 - 5 years

◯ 5 - 10 years

◯ More than 10 years

2. How would you describe your level of familiarity with the following programming languages?

	Unfamiliar	Somewhat familiar	Familiar	Very familiar	Expert
Java					
C#					
Objective C					
Visual Basic					
C					
C++					
Python					
Ruby					
Javascript					
PHP					
Perl					

IDE Familiarity

3. Which of the following Integrated Development Environments (IDEs) are you familiar with? Check all that apply.

☐ Apple Xcode

☐ Eclipse

☐ JetBrains IntelliJ

☐ Microsoft Visual Studio

☐ Oracle NetBeans

☐ None - I don't use an IDE

☐ Other IDEs (Please specify)

4. How would you describe your level of familiarity with the Eclipse IDE?

◯ Unfamiliar

◯ Somewhat familiar

◯ Familiar

◯ Very familiar

◯ Expert

Refactoring Practice

5. How would you describe your level of familiarity with each of the following?

	Unfamiliar	Somewhat familiar	Familiar	Very familiar	Expert
The Eclipse Refactoring Tool					
The Extract Superclass Refactoring in Eclipse					

6. Consider the situations in which you need to refactor your code. What portion of the time, in these situations, do you use a refactoring tool?

◯ Never

◯ Rarely

◯ Some of the time

◯ Most of the time

◯ Nearly every time

7. Consider the situations in which you can refactor your code using the Extract Superclass refactoring of your IDE. What portion of the time, in these situations, do you use the refactoring tool?

◯ Never

◯ Rarely

◯ Some of the time

◯ Most of the time

◯ Nearly every time

Procedures

1 HTML Parser

We would like to ask you to refactor an HTML parser to remove some of its code duplications. The parser generates objects that represent HTML tags and pieces of text. Consider the following piece of HTML:

```html
<html>
  <head>Login Form</head>
  <body>
    <a href="home"><img src="home-icon.png"></a>
    <form action="register" method="post">
      <p>Username: <input type="text" name="username"/></p>
      <p>Password: <input type="password" name="password"/></p>
    </form>
  </body>
</html>
```

The parser would generate instances of classes like the following for the elements of the input HTML code.

- `HTMLLinkTag` (for the `<a href="...">` tag)
- `HTMLFormTag` (for the `<form ...>` tag)
- `HTMLImageTag` (for the `<img src="...">` tag)

2 Removing Code Duplication

Since the link tag `<a href="...">` contains an image tag `<img src="...">`, the parser makes the corresponding `HTMLImageTag` object a child of the corresponding `HTMLLinkTag` object. Similar to `HTMLLinkTag`, `HTMLFormTag` is a child container. Each of `HTMLLinkTag` and `HTMLFormTag` has a `Vector` field for storing its children. Both classes provide a method called `toPlainTextString()` that generates the non-HTML formatted text of the tag's children.

Figure 1 shows the current code structure of the HTML parser. There is some code duplication in the `Vector` field and method `toPlainTextString()` of `HTMLLinkTag` and `HTMLFormTag`. Although the `Vector` fields in `HTMLLinkTag` and `HTMLFormTag` have different names, they both contain the children of the tag. The methods `toPlainTextString()` in the two classes are similar. Their main difference is in how they retrieve the tag's children. We ask you to move the `Vector` fields and `toPlainTextString()` methods into a common superclass called `CompositeHTMLTag`. The code snippet in Figure 2 shows the structure of HTML parser after this refactoring.

```
package org.htmlparser.tags;
...
public class HTMLLinkTag extends HTMLTag ...
  private java.util.Vector nodeVector;
  ...
  public Enumeration linkData() {
    return nodeVector.elements();
  }
  ...
  public String toPlainTextString() {
    StringBuffer sb = new StringBuffer();
    HTMLNode node;
    for (Enumeration e = linkData(); e.hasMoreElements();) {
      node = (HTMLNode) e.nextElement();
      sb.append(node.toPlainTextString());
    }
    return sb.toString();
  }
  ...
```

```
package org.htmlparser.tags;
...
public class HTMLFormTag extends HTMLTag ...
  protected Vector allNodesVector;
  ...
  public Vector getAllNodesVector() {
    return allNodesVector;
  }
  ...
  public String toPlainTextString() {
    StringBuffer stringRepresentation = new StringBuffer();
    HTMLNode node;
    for (Enumeration e = getAllNodesVector().elements(); e.hasMoreElements();) {
      node = (HTMLNode) e.nextElement();
      stringRepresentation.append(node.toPlainTextString());
    }
    return stringRepresentation.toString();
  }
  ...
```

Figure 1: The code structure of the HTML parser before the refactoring

```
package org.htmlparser.tags;
...
public class CompositeHTMLTag extends HTMLTag ...
  protected Vector children;
  public CompositeHTMLTag(...) {
    super(...);
  }
  public String toPlainTextString() {
    StringBuffer stringRepresentation = new StringBuffer();
    HTMLNode node;
    for (Enumeration e = children(); e.hasMoreElements();) {
      node = (HTMLNode) e.nextElement();
      stringRepresentation.append(node.toPlainTextString());
    }
    return stringRepresentation.toString();
  }
  protected Enumeration children() {
    return children.elements();
  }
  ...
```

```
package org.htmlparser.tags;
...
public class HTMLLinkTag extends CompositeHTMLTag ...
  public Enumeration linkData() {
    return children.elements();
  }
  ...
```

```
package org.htmlparser.tags;
...
public class HTMLFormTag extends CompositeHTMLTag ...
  public Vector getAllNodesVector() {
    return children;
  }
  ...
```

Figure 2: The code structure of the HTML parser after the refactoring

3 Introductory Tasks

Before refactoring the code, we ask you to do a few tasks to explore the code and familiarize yourself with it.

1. Please ensure that the unit tests packaged in `org.htmlparser.tests.AllTests` pass. To run these unit tests, open the class `AllTests`. Then, select `Run` → `Run As` → `JUnit Test` from the main menu.

2. Open the two classes that you are going to refactor later: `HTMLLinkTag` and `HTMLFormTag`.

3. Check the hierarchies of both classes `HTMLLinkTag` and `HTMLFormTag`. To see the hierarchy of a class, move the curor inside the class name, then right-click and select `Quick Type Hierarchy` from the context menu.

4. Locate the declarations of the two fields `HTMLLinkTag.nodeVector` and `HTMLFormTag.allNodesVector` and see how many times they are referenced. To find the references of a name, move the cursor inside the name, then right-click and select `References` → `Workspace` from the context menu.

5. Locate the declarations of the methods called `toPlainTextString()` in classes `HTMLLinkTag` and `HTMLFormTag`. Method `HTMLLinkTag.toPlainTextString()` calls method `linkData()`. Find the declaration of `linkData()`. To go to the declaration of a method, move the cursor inside the method name, and press `F3`. Similarly, method `HTMLFormTag.toPlainTextString()` calls method `getAllNodesVector()`. Find the declaration of `getAllNodesVector()`.

4 Refactoring Tasks

We ask you to refactor HTML parser from the structure shown in Figure 1 to the structure illustrated in Figure 2. Moreover, we ask you to perform the same refactoring using two sets of automated refactoring tools: *wizard-based* and *stepwise* refactorings.

While refactoring the code using each set of tools, please try to leverage the automated tools in that set as much as possible and avoid manual refactorings.

Please ensure the following after your refactoring:

- The unit tests packaged in `org.htmlparser.tests.AllTests` should pass.

- The new class you created, i.e. `CompositeHTMLTag` should be referenced only in two other classes: `HTMLLinkTag` and `HTMLFormTag`.

Stepwise Refactoring

The following are some of the refactorings that you can use to refactor the HTML parser in small steps:

- Create a new super class for '?' in '?'
- Extract to method
- Move '?' to super type '?'
- Rename in workspace

The question marks (?) in the above refactoring names depend on the elements that you invoke the refactorings on.

Select a piece of code that you would like to refactor and press the shortcut key `CTRL+1` to get a list of refactorings that are applicable to your selection. Then, double-click on your desired refactoring to perform it.

If you know the shortcut key for "Rename in workspace", you may use the shortcut key to invoke it directly.

You can use the "Create New Superclass" refactoring in the "Quick Refactor" menu to create a new superclass for one or more classes. The "Quick Refactor" menu is also available in the context menu. You can right-click on one or more classes and go to the "Quick Refactor" submenu to invoke the "Create New Superclass" refactoring.

Wizard-based Refactoring

The following are some of the wizard-based refactorings that you may use to refactor the HTML parser:

- Extract Method
- Extract Superclass
- Pull Up
- Rename

There are several ways to invoke a wizard-based refactoring. First, you can go to the `Refactor` menu and select one of the available refactorings. Second, you can select a code element in the editor, the package explorer view, or the outline view, and open the refactoring context menu. You can open the context menu by either right-clicking and going to the `Refactor` submenu or using the shortcut key `Shift-Alt-T`. Third, you can invoke some of the refactorings by their shortcut keys listed in the `Refactor` menu.

Postquestionnaire

1. Which style of automated refactoring is easier to use?

◯ Wizard-based

◯ Stepwise

◯ The two styles are similar in this regard.

2. Which style of automated refactoring is easier to learn?

◯ Wizard-based

◯ Stepwise

◯ The two styles are similar in this regard.

3. Which style of automated refactoring gives you more control over the refactoring?

◯ Wizard-based

◯ Stepwise

◯ The two styles are similar in this regard.

4. In which style of automated refactoring do you feel more confident about the correctness of the refactoring?

◯ Wizard-based

◯ Stepwise

◯ The two styles are similar in this regard.

5. Consider how often you perform a change that can be automated using one of the two styles of refactoring. Which style of automated refactoring supports the kind of transformations that are more likely to occur in your code?

◯ Wizard-based

◯ Stepwise

◯ The two styles are similar in this regard.

6. Which style of automated refactoring are you more satisfied with?

◯ Wizard-based

◯ Stepwise

◯ The two styles are similar in this regard.

7. Suppose that the two user interfaces of the Extract Superclass refactoring are available in your IDE. Consider the situations in which you need to perform an Extract Superclass refactoring similar to the one you just performed. What portion of the time, in these situations, do you think you would use each of the following ways of performing such a refactoring?

	Never	Rarely	Some of the time	Most of the time	Nearly every time
Wizard-based refactoring					
Stepwise refactoring					
Manual refactoring					