Java Annotations

INTRODUCTION

Java Annotations

• A way of "tagging" source code elements with extra information
• Widely used by Java frameworks:
  • Java compiler/Eclipse IDE
  • Spring Framework
  • JUnit 4
  • Enterprise JavaBeans

Java Annotations are a powerful and flexible feature of Java that allow adding metadata to Java code. However, managing annotations can be challenging, especially when dealing with large-scale codebases. This document introduces JQuery-Based Refactorings (JQR), a framework designed to simplify the process of managing Java annotations.

Solution: JQuery-Based Refactoring (JQR) Framework

• Use JQuery (a code searching tool with awareness of Java code structure) to find target code elements to manage and determine what annotation management tasks (a.k.a. "refactorings") are available for those elements.
• Use a Java API built around the Eclipse LTK Refactoring Framework to perform annotation changes.

Using the Framework: An Example

Applying @Override annotations to methods

1. Use JQuery to search for refactoring targets
2. Select refactoring to apply
3. Review proposed changes and apply refactoring

Extending the Framework

Reuse one of the provided refactorings (e.g., Add Annotation or Remove Annotation), or implement your own.

Validating the Framework

Experimental Approach

• Use JQR to perform various annotation management tasks on JHotDraw 7.1.1, a drawing application.
• Observe setup time and execution time for refactoring scenarios and difficulties encountered.

Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Java API</th>
<th>JQuery configuration</th>
<th>Query for targets</th>
<th>Execution time</th>
<th>Number of annotations processed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Add implementation to all methods in package org.jhotdraw.util not called from org.jhotdraw figuring</td>
<td>N/R</td>
<td>N/R</td>
<td>3m</td>
<td>28s</td>
<td>51</td>
</tr>
<tr>
<td>(2) Add implementation to all methods whenever applicable</td>
<td>N/R</td>
<td>N/R</td>
<td>N/R</td>
<td>24s</td>
<td>375</td>
</tr>
<tr>
<td>(3) Add implementations to all methods in package org.jhotdraw.gui</td>
<td>N/R</td>
<td>N/R</td>
<td>N/R</td>
<td>16s</td>
<td>7</td>
</tr>
<tr>
<td>(4) Double the value of &quot;timeout&quot; property on all setter methods in package org.jhotdraw.geom.desiredPathOn</td>
<td>30m</td>
<td>4m</td>
<td>2m</td>
<td>10s</td>
<td>7</td>
</tr>
</tbody>
</table>

Difficulties Observed

• Java implementation of new refactoring (scenario 4) required familiarity with complicated Eclipse APIs.
• Successfully used JQRs to perform complex annotation management tasks quickly.
• Configuration syntax makes it easy to reuse Java refactoring classes provided with framework in new situations.
• Java implementation of new refactorings is harder.

Conclusions

• Apply framework to general refactoring tasks, not just annotations.
• Develop a comprehensive library of annotation refactorings.
• Release framework as Eclipse plugin.

References


Acknowledgments

Thanks go to my supervisor, Kris De Volder, for his guidance and encouragement, and to Lloyd Markle and Rick Chen for technical assistance and feedback. This work was partially supported by an NSERC USRA and a Rick Sample Memorial Scholarship.