RefaFlex: Safer Refactorings for Reflective Java Programs
Outline

- Problem
- RefaFlex
- Constraint based refactoring
- Example
- Evaluation
- Conclusion
- Discussion
Problem

What is Reflection?
Problem

What is Reflection?

“…reflection is the ability of a computer program to examine and modify the structure and behavior of an object at runtime”
What is Reflection?

“...reflection is the ability of a computer program to examine and modify the structure and behavior of an object at runtime”
Problem

What is Reflection?

```java
1. public class C {
   2.     public String i = ... 
   3. }

4. public class Reflection {
   5.     public void m() throws Exception {
   6.         Class c = Class.forName("C");
   7.         Field f = c.getField("i");
   8.     }
   9. }
```
Problem

What is Reflection?

What if we change the name of the i variable using refactoring tools?
Problem

What is Reflection?

What if we change the name of the `i` variable using refactoring tools?

Turns out modern refactoring tools can’t always spot the effect on reflection.
Problem: NoSuchFieldException!

This results is “non-refactoring transformation”.

The change intended on preserving the program’s behavior, but didn’t.
RefaFlex

“Novel and more defensive approach towards the refactoring of reflective (Java) programs”
RefaFlex

“Novel and more defensive approach towards the refactoring of reflective (Java) programs”

Built upon previous works: TamiFlex and RefaCola
“Novel and more defensive approach towards the refactoring of reflective (Java) programs”

Built upon previous works: TamiFlex and RefaCola

RefaFlex!
RefaFlex

TamiFlex
- will log reflective calls.

RefaCola
- creates constraints for refactoring
- solves them
RefaFlex

TamiFlex
- will log reflective calls.

RefaCola
- creates constraints for refactoring
- solves them

Goal: Tell the developer when refactoring produces non-refactoring transformation
Before we go on…

What is Constraint Based Refactoring?
Before we go on…

What is Constraint Based Refactoring?

- Refactoring tools use patterns to generate constraints that are necessary to maintain program semantics.
Before we go on...

What is Constraint Based Refactoring?

- Refactoring tools use patterns to generate constraints that are necessary to maintain program semantics.
- All constraints make up a constraint system.
Before we go on…

What is Constraint Based Refactoring?

- Refactoring tools use patterns to generate constraints that are necessary to maintain program semantics.
- All constraints make up a constraint system.
- Each solution to the constraint system is a valid refactoring.
Before we go on...

What is Constraint Based Refactoring?

● Refactoring tools use patterns to generate constraints that are necessary to maintain program semantics.
● All constraints make up a constraint system.
● Each solution to the constraint system is a valid refactoring.

So what do we need to add in order to be able to refactor reflections?
Constraint Based Refactoring

Constraint rules for correct handling of reflections!
Constraint Based Refactoring

Format:

Program Query: Preconditions from runtime analysis.
Constraint: Constraint generation pattern
Constraint Based Refactoring

Format:

Program Query:
- Preconditions from runtime analysis.

Constraint:
- Constraint generation pattern

### Variables:
- `identifier(d) ∈ String`: the identifier of `d`
- `accessibility(a) ∈ Acc`: the access modifier of `a`
- `package(t) ∈ P`: the declaring package of `t`
- `type(m) ∈ T`: the declaring type of `m`
- `parameters(m) ∈ T^m`: `m`'s `n_m` formal parameter types
- `stringArg(r) ∈ String`: the string passed or received by `r`
- `isAccessible(r) ∈ boolean`: `r`'s `isAccessible()` value
- `package(r) ∈ P`: the package in which `r` resides
- `type(r) ∈ T`: the type in which `r` resides

### Domains:
- `P`: the set of declared packages
- `T`: the set of declared types
- `M`: the set of decl. methods, fields and constructors
- `D = P ∪ T ∪ M`
- `A = T ∪ M`
- `Acc = {private, package, protected, public}`
- `R`: the set of calls to the reflection API
Constraint Based Refactoring

Example:

\[
\text{Constraint Rule:}
\]
If the program accesses some field, then during refactoring, we need to make sure that the field is public.
Constraint Based Refactoring

Example:

(program query

Constraint Rule:
If the program accesses some field, then during refactoring, we need to make sure that the field is public.
Constraint Based Refactoring

Example: program query

Constraint Rule:
If the program accesses some field, then during refactoring, we need to make sure that the field is public.
Constraint Based Refactoring

Example:

```
(Class#getField, class, field)
accessibility(field) = public
```

Constraint Rule:
If the program accesses some field, then during refactoring, we need to make sure that the field is public.

getField can only work for public fields under Java Reflections!
Constraint Based Refactoring

Example:

```
(Class#getField, class, field)
accessibility(field) = public
```

Constraint Rule:
If the program accesses some field, then during refactoring, we need to make sure that the field is public.

This is the key idea!!!
Constraint Based Refactoring

Naming Rule

\[
\frac{\text{holdsName}(r, d)}{\text{stringArg}(r) = \text{identifier}(d)}
\]

Scoping Rule

\[
\frac{\text{Class}\#\text{get}^*(r, m)}{\text{receiver}(r) \leq T \text{type}(m)}
\]

Accessibility Rule

\[
\frac{\text{Class}\#\text{get}^*(r, m)}{\text{accessibility}(m) = \text{public}}
\]

Hiding Rule

\[
\frac{\text{Class}\#\text{get}^*(r, m), \text{member}(m')}{\text{identifier}(m) \neq \text{identifier}(m') \lor \text{type}(m) < T \text{type}(m') \lor \text{type}(m) < T \text{receiver}(r) \lor \text{accessibility}(m') < A \text{public} \lor \text{parameters}(m) \neq \text{parameters}(m')}
\]
RefaFlex

TamiFlex
- will log reflective calls.

RefaCola
- creates constraints for refactoring
- solves them: either can’t solve or returns a set of changed variables.
```java
package a;

import java.lang.reflect.Method;

public class MyClass {

    void m() {
        System.out.println("method invocation");
    }

    void n() {
        m();
    }

    public static void main(String[] args) throws Exception {
        String methodName = args.length == 0 ? "m" : args[0];
        Class<MyClass> cls = MyClass.class;
        Method m = cls.getDeclaredMethod(methodName);
        m.invoke(cls.newInstance());
    }

    @Test
    public void test() throws Exception{
        main(new String[] {"m");
    }
```
RefaFlex

TamiFlex
Try to change the method name
Try to change the method name
RefaFlex

RefaCola

Log

\[
\frac{\text{holdsName}(r, d)}{\text{stringArg}(r) = \text{identifier}(d)}
\]

Variables:
- \text{id}(d) \in \text{String}
- \text{accessibility}(a) \in \text{Acc}
- \text{package}(t) \in P
- \text{type}(m) \in T
- \text{parameters}(m) \in T^m
- \text{stringArg}(r) \in \text{String}
- \text{isAccessible}(r) \in \text{boolean}
- \text{package}(r) \in P
- \text{type}(r) \in T

with \ d \in D, \ t \in T, \ m \in M, \ a \in A \ and \ r \in R

Domains:
- \ P \quad \text{the set of declared packages}
- \ T \quad \text{the set of declared types}
- \ M \quad \text{the set of decl. methods, fields and constructors}
- \ D = P \cup T \cup M
- \ A = T \cup M
- \ Acc \ {\text{private, package, protected, public}}
- \ R \text{ the set of calls to the reflection API}
RefaFlex

RefaCola

RefaCola Solver

\[
\frac{\text{holdsName}(r, d)}{\text{stringArg}(r) = \text{identifier}(d)}
\]

Find the solution.
Rename conflicts with Class#getDeclaredMethod(String,Class<?>) in MyClass.java:18 binding to m.

MyClass.java

```java
String methodName = args.length == 0 ? "m" : args[0];
Class<MyClass> cls = MyClass.class;
Method m = cls.getDeclaredMethod(methodName);
m.invoke(cls.newInstance());
```

@Test
public void test() throws Exception{
    main(new String[]{"m"});
}
RefaFlex can also change it, if advanced feature is turned on.
Evaluation

Evaluation conducted using 3 open source projects: *Play, Joda, JCC* and the following metric:

- Necessity
- Flexibility (Advanced Features)
- Correctness
- Efficiency
Evaluation

Experiment:

- Apply: RENAME [FIELD, METHOD, TYPE, PACKAGE], MOVE TYPE, CHANGE METHOD SIGNATURE to all 3 projects where relevant.
- Do 3 times for cases: No RefaFlex, RefaFlex, advanced feature RefaFlex
Evaluation

Result:

<table>
<thead>
<tr>
<th>Program</th>
<th>RENAME FIELD (total)</th>
<th>RENAME METHOD (total)</th>
<th>RENAME TYPE (total)</th>
<th>RENAME Pkg (total)</th>
<th>MOVE TYPE (total)</th>
<th>CHANGE SIGNATURE (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E  R  R+</td>
<td>E  R  R+</td>
<td>E  R  R+</td>
<td>E  R  R+</td>
<td>E  R  R+</td>
<td>E  R  R+</td>
</tr>
<tr>
<td>Play [8]</td>
<td>1014 8 15 14</td>
<td>2875 457 485 482</td>
<td>521 0 11 11</td>
<td>43 0 9 9</td>
<td>280 0 11 11</td>
<td>2875 878 907 907</td>
</tr>
<tr>
<td>Joda [2]</td>
<td>58 0 0 0</td>
<td>236 28 109 107</td>
<td>67 0 2 2</td>
<td>6 0 1 1</td>
<td>36 0 2 2</td>
<td>236 75 163 163</td>
</tr>
<tr>
<td>JCC [7]</td>
<td>878 34 36 34</td>
<td>5661 1358 2193 2023</td>
<td>645 5 5 5</td>
<td>7 0 0 0</td>
<td>425 5 6 6</td>
<td>5661 2525 2776 2776</td>
</tr>
<tr>
<td>Σ</td>
<td>1950 42 51 48</td>
<td>8772 1843 2787 2612</td>
<td>1233 5 18 18</td>
<td>56 0 10 10</td>
<td>741 5 19 19</td>
<td>8772 3478 3846 3846</td>
</tr>
</tbody>
</table>

total = number of refactorings, E = rejected by Eclipse, R = rejected by REFAFLex, R+ = rejected even under possible rewriting
Evaluation

Necessity:

Applied 21524 refactorings. Eclipse refactoring failed 1358 of them. a little over 6%
Evaluation

Flexibility:

178 cases where we are able to successfully rewrite reflection.

<table>
<thead>
<tr>
<th>Program</th>
<th>Rename Field</th>
<th>Rename Method</th>
<th>Rename Type</th>
<th>Rename Pkg</th>
<th>Move Type</th>
<th>Change Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>total E R R+</td>
<td>total E R R+</td>
<td>total E R R+</td>
<td>total E R R+</td>
<td>total E R R+</td>
<td>total E R R+</td>
</tr>
<tr>
<td>Play [8]</td>
<td>1014 8 15 14</td>
<td>2875 457 485 482</td>
<td>521 0 11 11</td>
<td>43 0 9 9</td>
<td>280 0 11 11</td>
<td>2875 878 907 907</td>
</tr>
<tr>
<td>Joda [2]</td>
<td>58 0 0 0</td>
<td>236 28 109 107</td>
<td>67 0 2 2</td>
<td>6 0 1 1</td>
<td>36 0 2 2</td>
<td>236 75 163 163</td>
</tr>
<tr>
<td>JCC [7]</td>
<td>878 34 36 34</td>
<td>5661 1358 2193 2023</td>
<td>645 5 5 5</td>
<td>7 0 0 0</td>
<td>425 5 6 6</td>
<td>5661 2525 2776 2776</td>
</tr>
<tr>
<td>Σ</td>
<td>1950 42 51 48</td>
<td>8772 1843 2787 2612</td>
<td>1233 5 18 18</td>
<td>56 0 10 10</td>
<td>741 5 19 19</td>
<td>8772 3478 3846 3846</td>
</tr>
</tbody>
</table>

*total = number of refactorings, E = rejected by Eclipse, R = rejected by REFAFLEX, R+ = rejected even under possible rewriting*
Evaluation

Correctness:

“Guarantee” that test cases will pass after refactoring.
Evaluation

Efficiency:

- Instrumentation by TamiFlex occurs within the Java runtime library, so it is really fast.
- Program that made over 684,000 calls to reflection in a second, had overhead of 160%.
Conclusion

- A translation of uses of Java’s reflection API to equivalent RefaCola constraint rules.
- A mechanism preventing non-refactorings caused by reflective accesses to classes and members.
- A constraint-based resolution mechanism to make refactorings valid that would otherwise have to be rejected.
- A full open-source implementation, along with an evaluation that shows that their approach successfully works on real world programs.
Discussion

What are the possible downsides?
Discussion

What are the possible downsides?

- Dynamically analyzed: correct only for recorded reflective cases.
- Only extends 6 refactoring tools.
- Can’t reliably determine whether or not the refactoring is actually behavior preserving.
Discussion

Can this idea be applied to other languages?
Discussion

Can this idea be applied to other languages?

- As long as the language is using semantics that are consistent and we can use them to create constraint rules.
Discussion

- Eclipse has 30 refactoring tools, but only 6 can be extended. Would using the other tools actually improve RefaFlex?
- Is there a way to avoid limitations? Improve?
- How often do developers use refactoring tools, as opposed to manual refactoring?
- How can we improve refactoring in general?