RefaFlex:
Safer Refactorings for Reflective Java Programs

Andreas Thies

Eric Bodden
package a;

public class C {
    public int i = 42;

    public static void main(String[] args) throws Exception {
        Class<?> c = Class.forName("a.C");
        java.lang.reflect.Field f = c.getField("i");
        Object myC = c.newInstance();
        System.out.println(f.get(myC));
    }
}
“In general, every interesting property of a program is undecidable”

(Don Roberts on refactoring Smalltalk programs)
Smalltalk Refactoring Browser

- executed before and after method invocations
- to analyze dynamic method lookup
- run tests and dynamically rewrite code at runtime
RefaFlex

Method Wrappers

monitor calls to the reflection API during test run execution

User Interface

check preconditions and compute code transformations

Offline Refactoring
Refactoring Reflective Code

```java
package a;

class Super {
    public int j = 23;
}

class C extends Super {
    public int i = 42;
}

public class Reflection {
    public static void main(String[] args) throws Exception {
        Class<?> c = Class.forName("a.C");
        Field f = c.getField("j");
        Object myC = c.newInstance();
        System.out.println(f.get(myC));
    }
}
```
Refactoring Reflective Code

```java
package a;

class Super {
    public int j = 23;
}

public class C extends Super {
    public int i = 42;
}

public class Reflection {
    public static void main(String[] args) throws Exception {
        Class<?> c = Class.forName("a.C");
        Field f = c.getField("j");  // FieldNotFoundException
        Object myC = c.newInstance();
        System.out.println((f.get(myC)));  // IllegalAccessException
    }
}
```
Refactoring Reflective Code

```java
package a;

class Super {
    public int j = 23;
}

class C extends Super {
    public int i = 42;
}

class Reflection {
    public static void main(String[] args) throws Exception {
        Class<?> c = Class.forName("a.C");
        Field f = c.getField("j");
        Object myC = c.newInstance();
        System.out.println(f.get(myC));
    }
}
```
Constraint Based Refactoring

Each potential change in the program is represented by a constraint variable

```java
public class C {
    int i = 42;
}
```
Constraint Based Refactoring

Preservation of program behavior is guaranteed by constraint generation rules

\[(\text{Class} \# \text{getField}, \text{class, field})\]

\[
\text{accessibility( field) = public}
\]
Constraint Based Refactoring

Reading a field’s value requires the field to be accessible

\[ \text{hostType}(\text{Field}\#\text{get}, \text{class}, \text{field}) \neq \text{hostType}(\text{field}) \Rightarrow \text{accessibility}(\text{field}) > \text{private} \]

\[ \text{hostPackage}(\text{Field}\#\text{get}, \text{class}, \text{field}) \neq \text{hostPackage}(\text{field}) \Rightarrow \text{accessibility}(\text{field}) = \text{public} \]
Constraint Based Refactoring

Referencing a field requires the field to be named as the string used for access

\[
\text{stringArg}\left( \text{Class}\#\text{getField}, \text{class}, \text{field} \right) = \text{identifier}( \text{field} )
\]
Rewriting Reflective Invocations

```java
Field f = A.class.getField(args[0]);

String theField = args[0];
/
if (theField.equals("i"))
    theField = "j";
Field f = A.class.getField(theField);
```

- accesses field `i`
- shall be renamed to `j`
Constraint Solving

• Once all constraints are generated, they are passed to a constraint solver

• If there exists a solution, the refactoring may be performed

• Each solution directly indicates the necessary code transformations to perform
Implementation

- **TamiFlex Framework**
- **Eclipse Refactoring Participants**
  - Rename {Field, Method, Package, Type}
  - Change Method Signature
  - Move Type
- **Refacola Framework**
  - 9 types of constraint variables
  - 21 constraint generation Rules
Limitations

The correctness of the performed refactorings directly correlates with the given test’s coverage.

RefaFlex guarantees that existing test cases still pass after the refactoring.
Limitations

While RefaFlex considers all string arguments to reflective calls, it cannot, in general, rewrite those arguments.

RefaFlex: Rename conflicts with `Class#forName` from `PluginLoader.class`
Evaluation

21,524 refactoring runs on three open source projects known to use reflection
Evaluation

21,524 refactoring runs on three open source projects known to use reflection

- Renamed by Eclipse
- ... and RefaFlex
- ... even under possible rewriting

- Rename Field (1950)
- Rename Type (1233)
- Rename Package (56)
- Move Type (741)

- Rename Method (8772)
- Change Signature (8772)
Our implementation and all evaluation data is available at

http://www.feu.de/ps/prjs/rf/

![Bar charts showing refactoring statistics](chart.png)