Source code analysis and transformation

Martin Monperrus

version of Oct 21, 2014

Latest version at http://www.monperrus.net/martin/lecture-slides-source-code-analysis-and-transformation.pdf

Slide contents

- Concepts:
  - Abstract Syntax Tree
  - Source code analysis
  - Lint tools
  - Source code transformation
- Tools
  - Spoon: a Java library for analyzing and transforming Java source code

An Example

```
try {

  // some code

} catch (Exception e) {

   e.printStackTrace();

}
```

Bad practice, should be either:

- throw new RuntimeException(e);
- Log.log(e)

Analysis

An Example

```
try {
  // some code
} catch (Exception e) {
  e.printStackTrace();
}
```

Text search (Grep):
grep "e.printStackTrace"?

What if:
th.printStackTrace(); ?

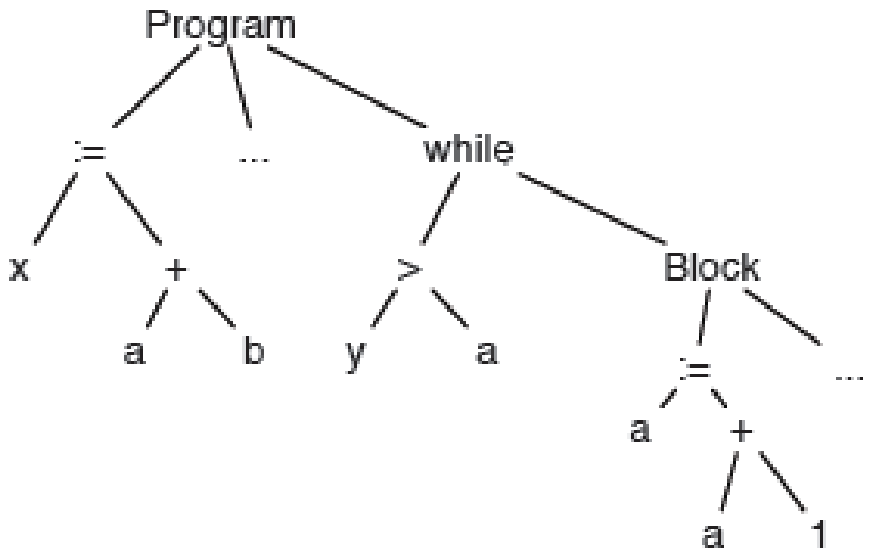
How to check empty catch blocks?

Text search: False positives, false negatives

Astract Syntax Tree

- An Abstract Syntax Tree is a data structure representing source code.

```
x := a + b;
y := a * b;
while (y > a) {
   a := a + 1;
   x := a + b
}
```



Libraries exist to compute and manipulate ASTs.

Astract Syntax Tree

- There can be many different ASTs for he same programming language
  - Use enum instead of classes
  - Groups nodes together or not
- An AST is designed for a given task (compilation, interpretation, code query, metrics, etc.)

There are many tools for extracting and analyzing ASTs:

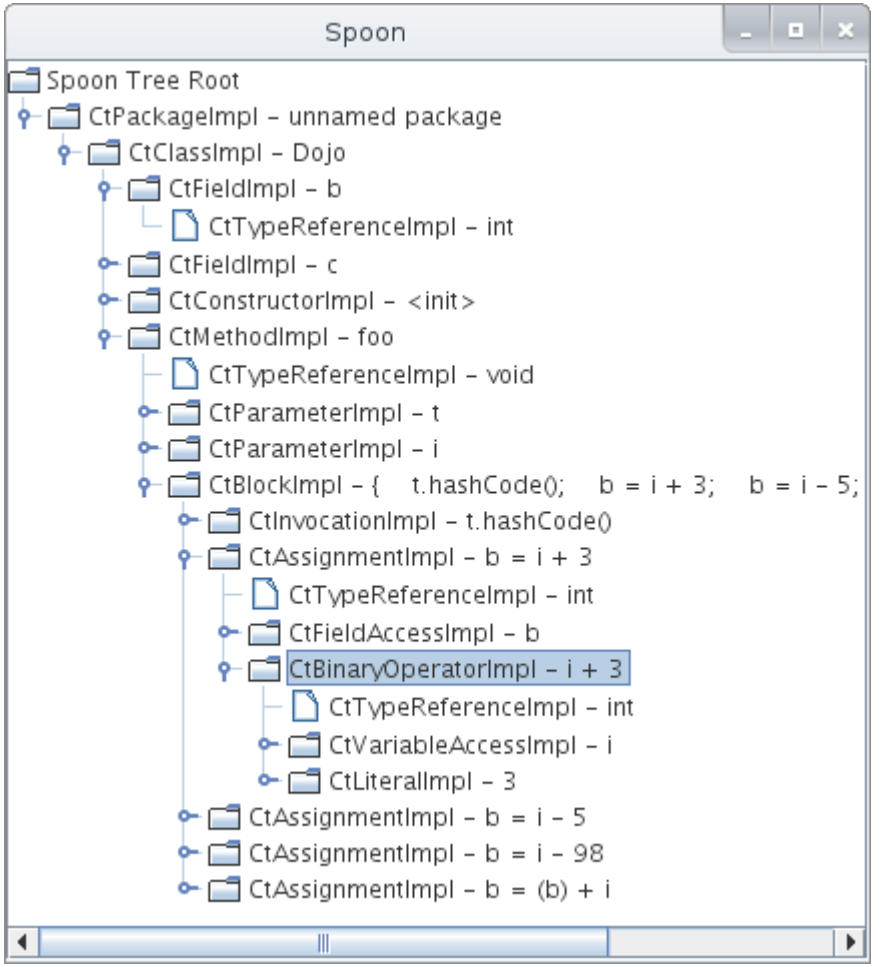
- Spoon (Inria)
- SrcML (Java/C++ to XML)
- JDT
- AST (Python library)
- Clang (frontend of the compiler)

Example: Spoon

Spoon provides Java ASTs (and a GUI to navigate through them)

java spoon.Launcher -g -i srcDir

```
public class Dojo {
  public int b;
  public double c;
  void foo(Object t, int i) {
     t.hashCode();
     b=i+3;
     b=i-5;
     b=i-98;
     b=b+i;
  }
}
```



Example: SrcML

SrcML provides XML-based ASTs that can be analyzed with any DOM technology

```
<unit>
<comment type="line">// copy the input to the output</comment>
<while>while
  <condition>(<expr><name><name>std</name>::<name>cin</name></name>
&gt;&gt; <name>n</name></expr>)</condition>
  <expr_stmt><expr><name><name>std</name>::<name>cout</name></name>
&lt;&lt; <name>n</name> &lt;&lt; '\n'</expr>;</expr_stmt></while>
</unit>
```

- Pro: toString is valid code
- Pro: Can be loaded in any XML ready library
- Con: no complete AST (very fine grain expressions are not handled)

AST Read versus Write

- An AST can be accessed in read and write mode.
- Static analysis is done in read mode
- Source code transformation is done in write mode
  - methods: insertBegin, insertEnd, insertBefore, insertAfter, replace

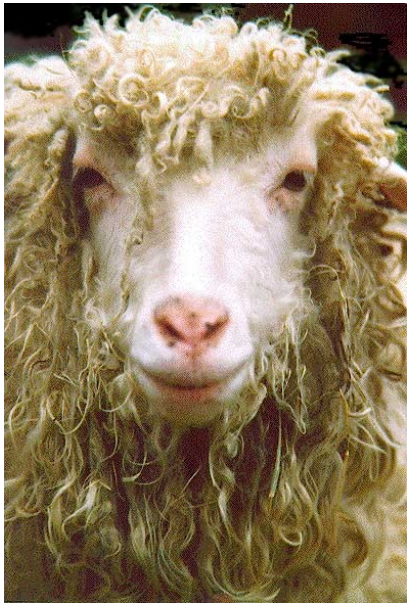
Source Code Analysis

AST analysis is useful is a number of different use cases:

- Lint tools / Bad smells
  - Lint, FindBugs, PMD, Checkstyle, JLint
- Verification of contracts
  - e.g. no explicit exception reaches the main
- Visualization

Origins of Lint

- Lint first appeared in the seventh version (V7) of the UNIX operating system in 1979.
- Lint flagged some suspicious and non-portable constructs in C language source code.
  - use before definition
  - unreachable code
  - ignored return values
  - etc.



Basics of AST Analysis: navigation and query

```
// navigation
class.getFields()
field.getDefaultExpression()
if.getThenBranch();
method.getBody();
...
```

```
// queries
list1 = methodBody.getElements(new
TypeFilter(CtAssignment.class));

// collecting all deprecated classes
list2 = rootPackage.getElements(new
AnnotationFilter(Deprecated.class));

// creating a custom filter to select all public fields
list3 = rootPackage.getElements(
  new AbstractFilter<CtField>(CtField.class) {
    public boolean matches(CtField field) {
      return field.getModifiers.contains(ModifierKind.PUBLIC);
    }});
```

Spoon Analysis #1: Detecting empty catch blocks

```
public class CatchProcessor extends
AbstractProcessor<CtCatch> {

public void process(CtCatch element) {
  if (element.getBody().getStatements().size() == 0) {
    getFactory().getEnvironment().report(this,
Severity.WARNING,
     element, "empty catch clause");
  }
}

}
```

$ java -cp spoon.jar spoon.Launcher -i sourceFolder -p CatchProcessor

Spoon Analysis #2: Detecting public fields

```
public class PublicFieldProcessorWarning extends AbstractProcessor<CtField<?>>{

    @Override

    public void process(CtField<?> arg0) {

        if (arg0.hasModifier(ModifierKind.PUBLIC)) {

            getEnvironment().report(this, Severity.WARNING, arg0,
                                    "Found a public field");

        }

    }
}
```

Source Code Transformation

A source code transformation tool provides you with an API.

Scope	Name	Description
generic	replace(element)	replaces an element by another one.
generic	insertBefore(element)	inserts the current element ("this") before another element in a code block.
generic	insertAfter(element)	inserts the current element ("this") after another element in a code block.
block	insertBegin(element)	adds an element at the begin of a code block.
block	insertEnd(element)	appends an element at the end of a code block.
throw	setThrownExpression(expr)	sets the expression returning a throwable object.
assignment	setAssignment(expression)	sets the expression to be assigned in a variable.
if	setElseStatement(stmt)	sets the "else" statement of an if/then/else.

Excerpt of Spoon's transformation API

Spoon Transformation #1: adding not-null checks

```
public class NotNullCheckAdderProcessor extends
         AbstractProcessor<CtParameter<?>> {

  @Override
  public boolean isToBeProcessed(CtParameter<?> element) {
    return !element.getType().isPrimitive();// only for objects
  }

  public void process(CtParameter<?> element) {
    // we declare a new snippet of code to be inserted
     CtCodeSnippetStatement snippet = getFactory().Core().createCodeSnippetStatement();

    // this snippet contains an if check
    snippet.setValue("if(" + element.getSimpleName() + " == null "
          + ") throw new IllegalArgumentException(
                        \"[Spoon inserted check] null passed as parameter\");");

    // we insert the snippet at the beginning of the method boby
    element.getParent(CtMethod.class).getBody().insertBegin(snippet);
  }

}
```

Spoon Transformation #2 (driven by annotations)

```
public @interface Bound {
  double min();
}

public void openUserSpacePort(@Bound(min = 1025) int a) {
  // code to open a port
}
```

```
public class Bound2Processor extends
AbstractAnnotationProcessor<Bound, CtParameter<?>> {

public void process(Bound annotation, CtParameter<?> element) {
  // we declare a new snippet of code to be inserted
  CtCodeSnippetStatement snippet = getFactory().Core()
    .createCodeSnippetStatement();

  // this snippet contains an if check
  snippet.setValue("if("
    + element.getSimpleName() + " < " + annotation.min() + ")"
    +" throw new RuntimeException(\"[Spoon check] Bound violation\");");

  // we insert the snippet at the beginning of the method boby
  element.getParent(CtMethod.class).getBody().insertBegin(snippet);
} // end process

}
```

Summary

- Source code analysis and transformation is powerful for automating parts of the development process
- Is sometimes more powerful at the abstract syntax tree level
- Spoon is Java library for analyzing and transforming Java source code