Code Transformation with

Spoon

Gérard Paligot

Code transformation

- Read, generate, analyse or transform.

Code transformation

- Read, generate, analyse or transform.
- Used more and more by libraries.

Code transformation

- Read, generate, analyse or transform.
- Used more and more by libraries.

```java
public void sayHello(String message) {
  if (message.isEmpty()) {
    System.out.println("Hello, World!");
  } else {
    System.out.println(message);
  }
}
```

Code transformation

- Read, generate, analyse or transform.
- Used more and more by libraries.

```java
@Log
public void sayHello(String message) {
  if (message.isEmpty()) {
    System.out.println("Hello, World!");
  } else {
    System.out.println(message);
  }
}
```

Code transformation

- Read, generate, analyse or transform.
- Used more and more by libraries.

```java
public void sayHello(@Log String message) {
  if (message.isEmpty()) {
    System.out.println("Hello, World!");
  } else {
    System.out.println(message);
  }
}
```

Code transformation

- Read, generate, analyse or transform.
- Used more and more by libraries.

```java
public void sayHello(@NotNull String message) {
  if (message.isEmpty()) {
    System.out.println("Hello, World!");
  } else {
    System.out.println(message);
  }
}
```

Library rescue

- APT
- JDT
- JavaParser
- JTransformer
- ASM
- pfff
- Others…

JDT

- Developed at Eclipse.

JDT

- Developed at Eclipse.
- Used in the project Eclipse.

JDT

- Developed at Eclipse.
- Used in the project Eclipse.
- Easy to create a Eclipse plugin with JDT.

JDT

- Developed at Eclipse.
- Used in the project Eclipse.
- Easy to create a Eclipse plugin with JDT.
- Read, generate, analyse and transform source code.

JDT

- Developed at Eclipse.
- Used in the project Eclipse.
- Easy to create a Eclipse plugin with JDT.
- Read, generate, analyse and transform source code.
- API and meta model hard to use and to understand.

APT

- Developed at SUN, later by Oracle.

APT

- Developed at SUN, later by Oracle.
- Adopted by a lot of open-source librairies.

APT

- Developed at SUN, later by Oracle.
- Adopted by a lot of open-source librairies.
- Accessible by the package javax.lang.model.* and javax.annotation.processing.

APT

- Developed at SUN, later by Oracle.
- Adopted by a lot of open-source librairies.
- Accessible by the package javax.lang.model.* and javax.annotation.processing.
- Well integrated in modernes build-management.

APT

- Developed at SUN, later by Oracle.
- Adopted by a lot of open-source librairies.
- Accessible by the package javax.lang.model.* and javax.annotation.processing.
- Well integrated in modernes build-management.
- Meta model limited and transformation not allowed.

```
@SupportedAnnotationTypes("fr.inria.NotNull")
public class TypeProcessor extends AbstractProcessor {
  @Override
  public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
    roundEnv.getElementsAnnotatedWith(NotNull.class).stream() //
            .map(e -> "annotation found on " + e.getSimpleName()) //
            .forEach(System.out::println);

    for (Element element : roundEnv.getElementsAnnotatedWith(NotNull.class)) {
      // Editing not possible!
    }
    return true;
  }
}
```

```java
@SupportedAnnotationTypes("fr.inria.NotNull")
public class TypeProcessor extends AbstractProcessor {
  @Override
  public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
    roundEnv.getElementsAnnotatedWith(NotNull.class).stream() //
            .map(e -> "annotation found on " + e.getSimpleName()) //
            .forEach(System.out::println);

    for (Element element : roundEnv.getElementsAnnotatedWith(NotNull.class)) {
      // Editing not possible!
    }
    return true;
  }
}
```

```java
@SupportedAnnotationTypes("fr.inria.NotNull")
public class TypeProcessor extends AbstractProcessor {
  @Override
  public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
    roundEnv.getElementsAnnotatedWith(NotNull.class).stream() //
            .map(e -> "annotation found on " + e.getSimpleName()) //
            .forEach(System.out::println);

    for (Element element : roundEnv.getElementsAnnotatedWith(NotNull.class)) {
      // Editing not possible!
    }
    return true;
  }
}
```

```java
@SupportedAnnotationTypes("fr.inria.NotNull")
public class TypeProcessor extends AbstractProcessor {
  @Override
  public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
    roundEnv.getElementsAnnotatedWith(NotNull.class).stream() //
            .map(e -> "annotation found on " + e.getSimpleName()) //
            .forEach(System.out::println);

    for (Element element : roundEnv.getElementsAnnotatedWith(NotNull.class)) {
      // Editing not possible!
    }
    return true;
  }
}
```

```java
@SupportedAnnotationTypes("fr.inria.NotNull")
public class TypeProcessor extends AbstractProcessor {
  @Override
  public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
    roundEnv.getElementsAnnotatedWith(NotNull.class).stream() //
            .map(e -> "annotation found on " + e.getSimpleName()) //
            .forEach(System.out::println);

    for (Element element : roundEnv.getElementsAnnotatedWith(NotNull.class)) {
      // Editing not possible!
    }
    return true;
  }
}
```

```java
@SupportedAnnotationTypes("fr.inria.NotNull")
public class TypeProcessor extends AbstractProcessor {
  @Override
  public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
    roundEnv.getElementsAnnotatedWith(NotNull.class).stream() //
            .map(e -> "annotation found on " + e.getSimpleName()) //
            .forEach(System.out::println);

    for (Element element : roundEnv.getElementsAnnotatedWith(NotNull.class)) {
      // Editing not possible!
    }
    return true;
  }
}
```

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <executions>
    <execution>
      <id>default-compile</id>
      <goals>
        <goal>compile</goal>
      </goals>
      <configuration>
        <compilerArgument>-proc:none</compilerArgument>
      </configuration>
    </execution>
  </executions>
</plugin>
```

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <executions>
    <execution>
      <id>default-compile</id>
      <goals>
        <goal>compile</goal>
      </goals>
      <configuration>
        <compilerArgument>-proc:none</compilerArgument>
      </configuration>
    </execution>
  </executions>
</plugin>
```

```xml
<dependency>
  <groupId>fr.inria</groupId>
  <artifactId>apt</artifactId>
  <version>${project.parent.version}</version>
  <optional>true</optional>
</dependency>
```

Spoon

- Developed at INRIA by Renauld Pawlak, Nicolas Petitprez and Carlos Noguera.

Spoon

- Developed at INRIA by Renauld Pawlak, Nicolas Petitprez and Carlos Noguera.
- Used by Spirals team at INRIA Lille, by Diversify at INRIA Rennes and by the world!

Spoon

- Developed at INRIA by Renauld Pawlak, Nicolas Petitprez and Carlos Noguera.
- Used by Spirals team at INRIA Lille, by Diversify at INRIA Rennes and by the world!
- Designed for developers to be use easily.

Spoon

- Developed at INRIA by Renauld Pawlak, Nicolas Petitprez and Carlos Noguera.

- Used by Spirals team at INRIA Lille, by Diversify at INRIA Rennes and by the world!

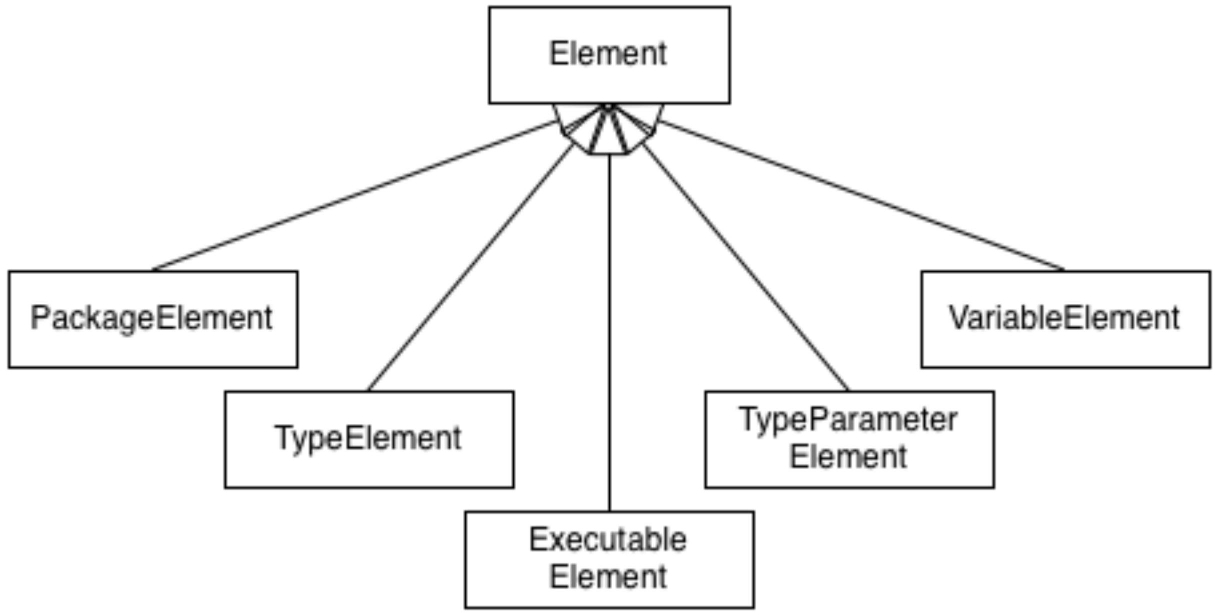
- Designed for developers to be use easily.

- Source to source code transformation.

Spoon

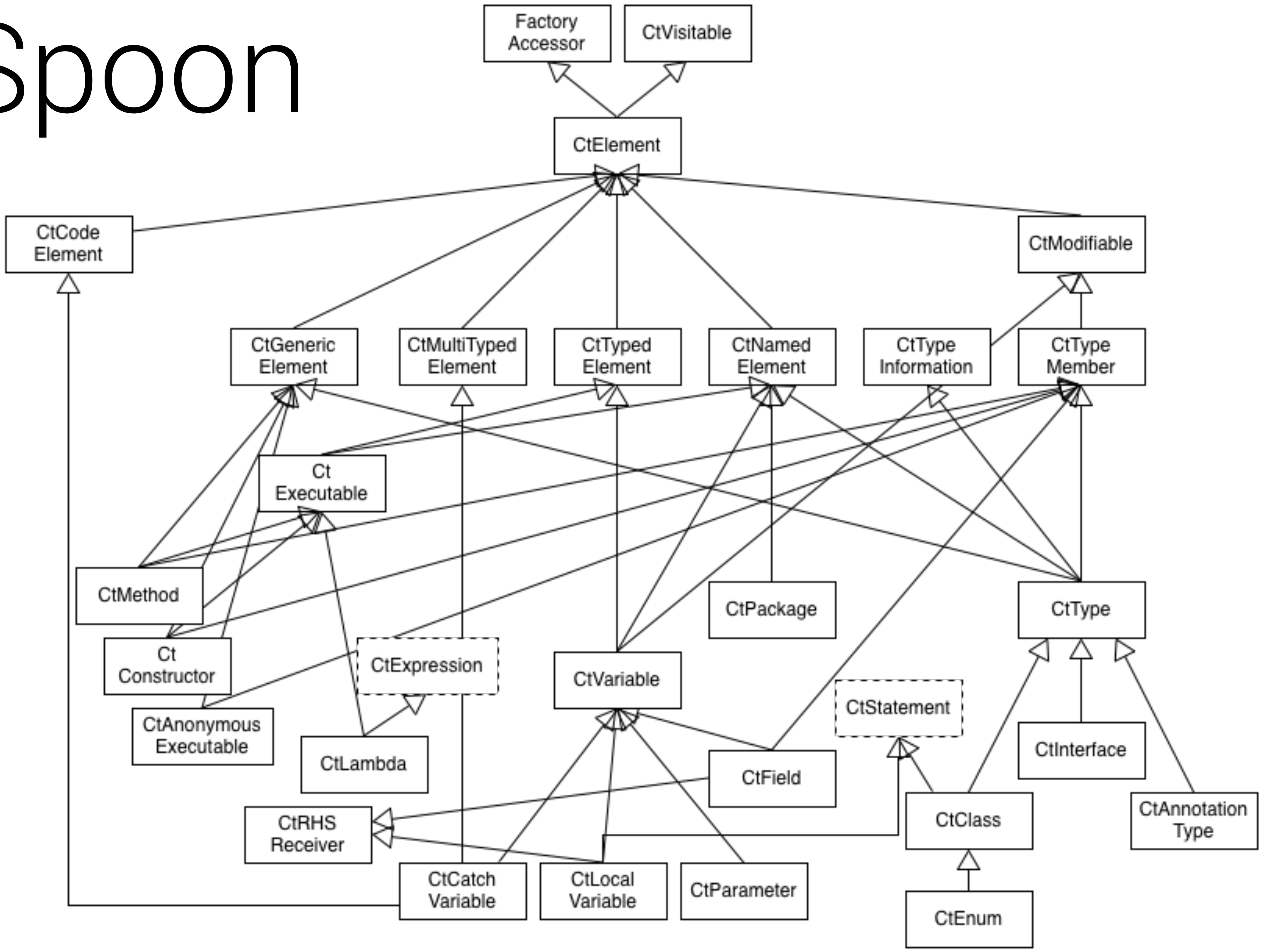
- Developed at INRIA by Renauld Pawlak, Nicolas Petitprez and Carlos Noguera.
- Used by Spirals team at INRIA Lille, by Diversify at INRIA Rennes and by the world!
- Designed for developers to be use easily.
- Source to source code transformation.
- Complete and fine-grained Java meta model.

APT



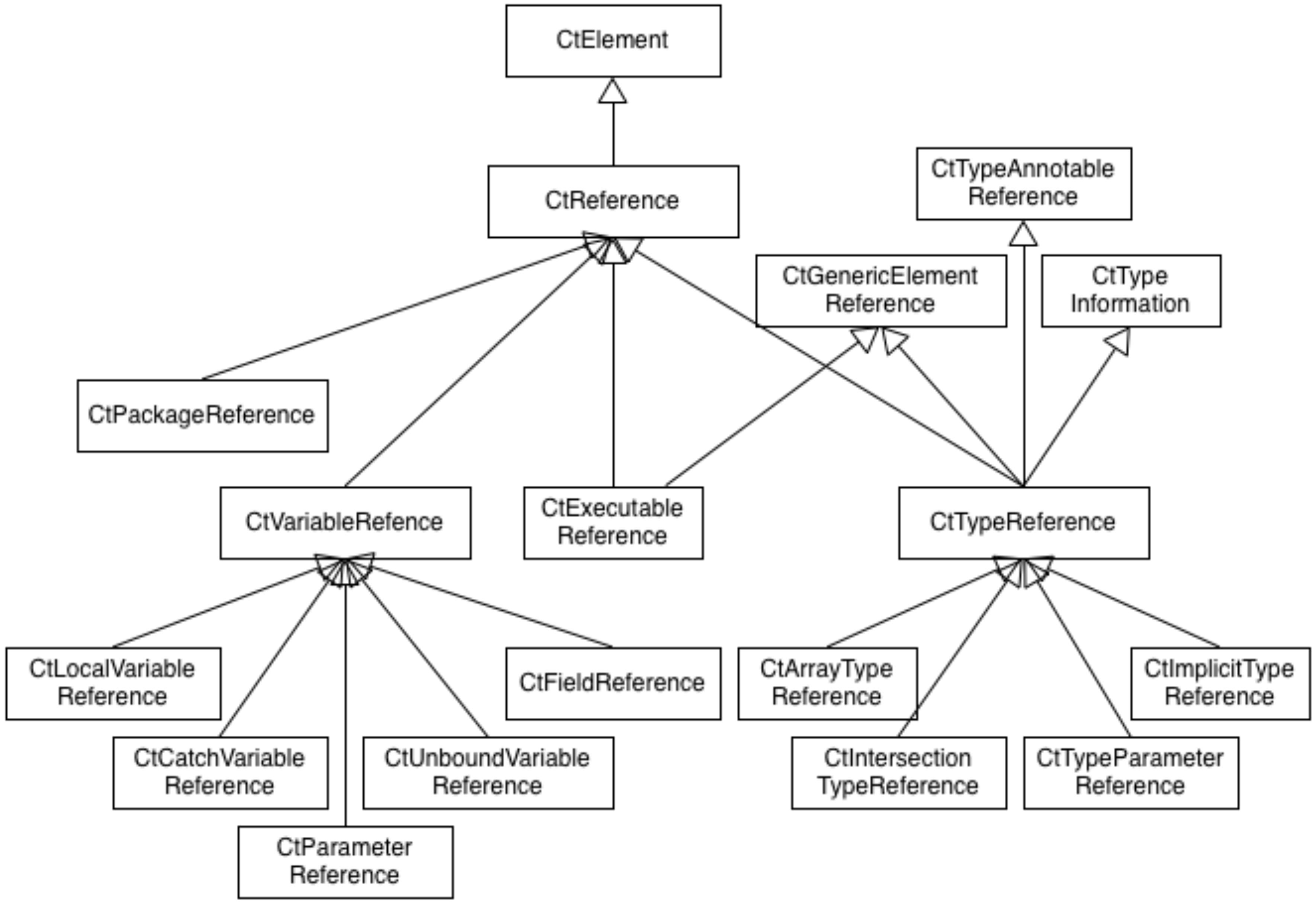
Element
PackageElement
TypeElement
Executable Element
TypeParameter Element
VariableElement

Spoon



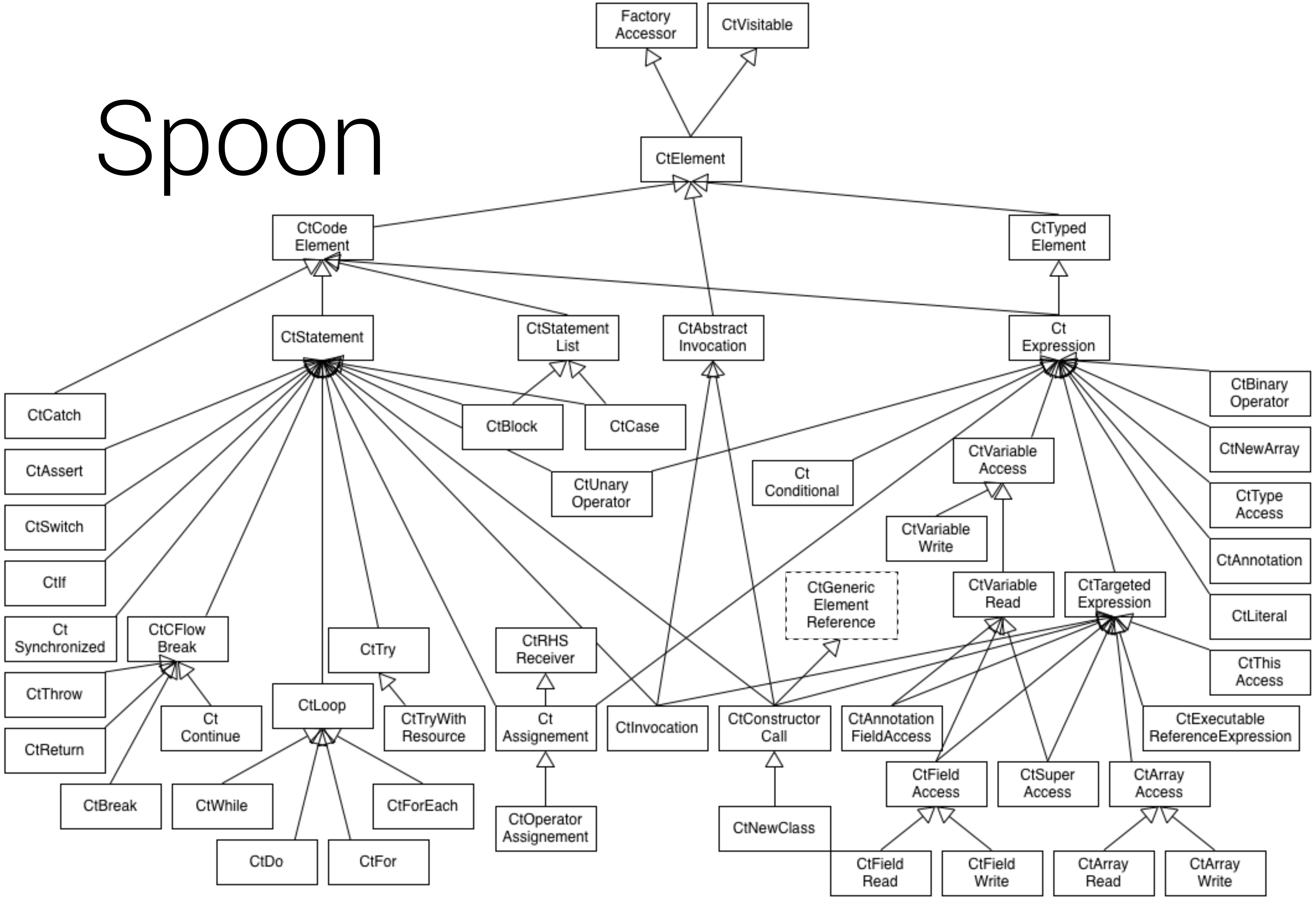
http://spoon.gforge.inria.fr/structural_elements.html

Spoon



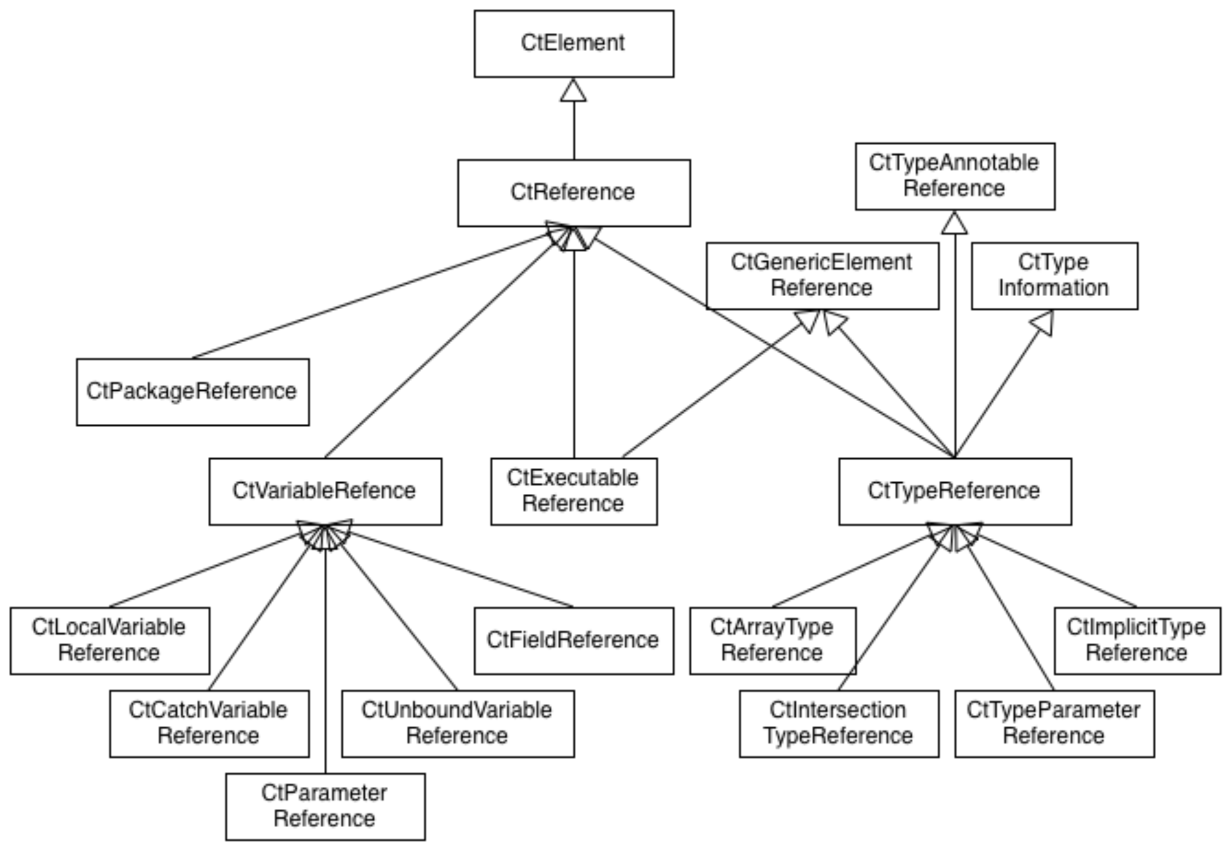
http://spoon.gforge.inria.fr/references.html

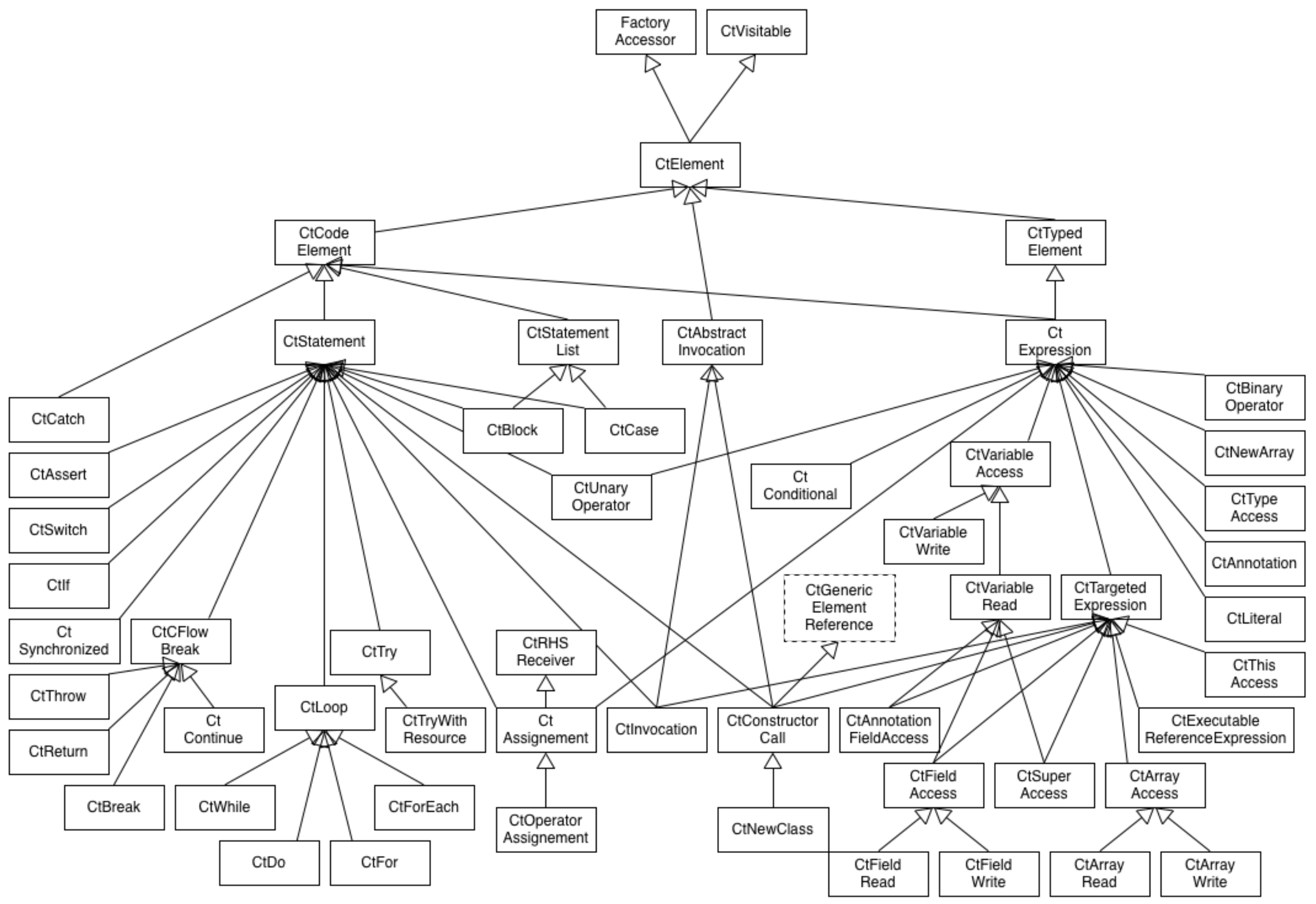
Spoon

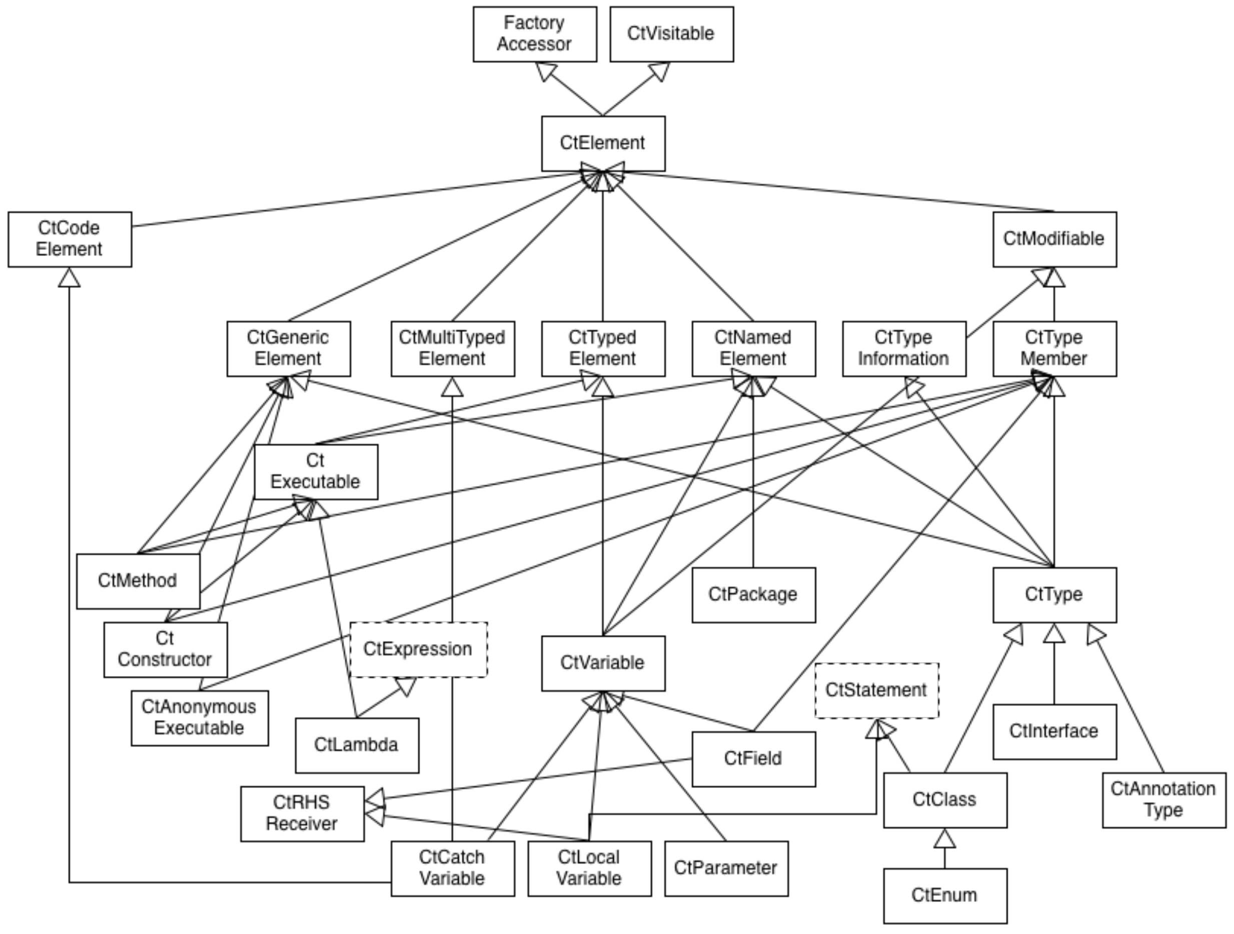


http://spoon.gforge.inria.fr/code_elements.html

Spoon


CtElement
CtReference
CtTypeAnnotable Reference
CtGenericElement Reference
CtType Information
CtPackageReference
CtVariableRefence
CtExecutable Reference
CtTypeReference
CtLocalVariable Reference
CtFieldReference
CtArrayType Reference
CtImplicitType Reference
CtCatchVariable Reference
CtUnboundVariable Reference
CtIntersection TypeReference
CtTypeParameter Reference
CtParameter Reference


Factory Accessor
CtVisitable
CtElement
CtCode Element
CtTyped Element
CtStatement
CtStatement List
CtAbstract Invocation
Ct Expression
CtBinary Operator
CtCatch
CtBlock
CtCase
CtNewArray
CtAssert
CtUnary Operator
Ct Conditional
CtVariable Access
CtType Access
CtSwitch
CtVariable Write
CtAnnotation
CtIf
CtGeneric Element Reference
CtVariable Read
CtTargeted Expression
CtLiteral
Ct Synchronized
CtCFlow Break
CtTry
CtRHS Receiver
CtThis Access
CtThrow
Ct Continue
CtLoop
CtTryWith Resource
Ct Assignement
CtInvocation
CtConstructor Call
CtAnnotation FieldAccess
CtExecutable ReferenceExpression
CtReturn
CtBreak
CtWhile
CtForEach
CtOperator Assignement
CtNewClass
CtField Access
CtSuper Access
CtArray Access
CtDo
CtFor
CtField Read
CtField Write
CtArray Read
CtArray Write


Factory Accessor
CtVisitable
CtElement
CtCode Element
CtModifiable
CtGeneric Element
CtMultiTyped Element
CtTyped Element
CtNamed Element
CtType Information
CtType Member
Ct Executable
CtMethod
CtPackage
CtType
Ct Constructor
CtExpression
CtVariable
CtStatement
CtAnonymous Executable
CtLambda
CtInterface
CtField
CtRHS Receiver
CtClass
CtAnnotation Type
CtCatch Variable
CtLocal Variable
CtParameter
CtEnum

How it works?

Input

AST

Processing

Output

Features

- Factory
- Query
- Processor
- Template

Processor

A program analysis is a combination of query and analysis code. In Spoon, this conceptual pair is reified in a processor. A processor is a class that focuses on the analysis of one kind of program elements.

http://spoon.gforge.inria.fr/processor.html

Processor

A program analysis is a combination of query and analysis code. In Spoon, this conceptual pair is reified in a processor. A processor is a class that focuses on the analysis of one kind of program elements.

```
public class TypeSpoonProcessor extends AbstractAnnotationProcessor<NotNull, CtParameter<?>> {
  @Override public void process(NotNull annotation, CtParameter<?> element) {

  }
}
```

http://spoon.gforge.inria.fr/processor.html

Processor

A program analysis is a combination of query and analysis code. In Spoon, this conceptual pair is reified in a processor. A processor is a class that focuses on the analysis of one kind of program elements.

```
public class TypeSpoonProcessor extends AbstractAnnotationProcessor<NotNull, CtParameter<?>> {
  @Override public void process(NotNull annotation, CtParameter<?> element) {

  }
}
```

http://spoon.gforge.inria.fr/processor.html

Processor

A program analysis is a combination of query and analysis code. In Spoon, this conceptual pair is reified in a processor. A processor is a class that focuses on the analysis of one kind of program elements.

```
public class TypeSpoonProcessor extends AbstractAnnotationProcessor<NotNull, CtParameter<?>> {
  @Override public void process(NotNull annotation, CtParameter<?> element) {

  }
}
```

http://spoon.gforge.inria.fr/processor.html

Processor

A program analysis is a combination of query and analysis code. In Spoon, this conceptual pair is reified in a processor. A processor is a class that focuses on the analysis of one kind of program elements.

```
public class TypeSpoonProcessor extends AbstractAnnotationProcessor<NotNull, CtParameter<?>> {
  @Override public void process(NotNull annotation, CtParameter<?> element) {
    System.out.println("annotation found on " + element.getSimpleName());
  }
}
```

http://spoon.gforge.inria.fr/processor.html

Processor

A program analysis is a combination of query and analysis code. In Spoon, this conceptual pair is reified in a processor. A processor is a class that focuses on the analysis of one kind of program elements.

```
public class TypeSpoonProcessor extends AbstractAnnotationProcessor<NotNull, CtParameter<?>> {
  @Override public void process(NotNull annotation, CtParameter<?> element) {
    System.out.println("annotation found on " + element.getSimpleName());
  }
}
```

http://spoon.gforge.inria.fr/processor.html

Factory

Create new elements, fill their data and add them in an existing AST. The factory is the entry point to do that and each factory has them own responsibility. i.e., CoreFactory creates empty nodes and CodeFactory creates a node ready to be printed.

http://spoon.gforge.inria.fr/factories.html

Factory

Create new elements, fill their data and add them in an existing AST. The factory is the entry point to do that and each factory has them own responsibility. i.e., CoreFactory creates empty nodes and CodeFactory creates a node ready to be printed.

```
if (message == "null")
    java.lang.System.out.println("message is null");
```

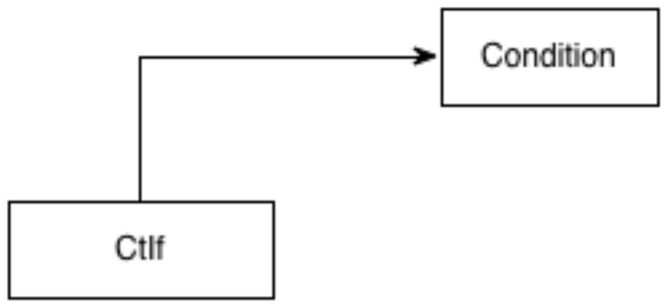
http://spoon.gforge.inria.fr/factories.html

```java
if (message == "null")
    java.lang.System.out.println("message is null");
```

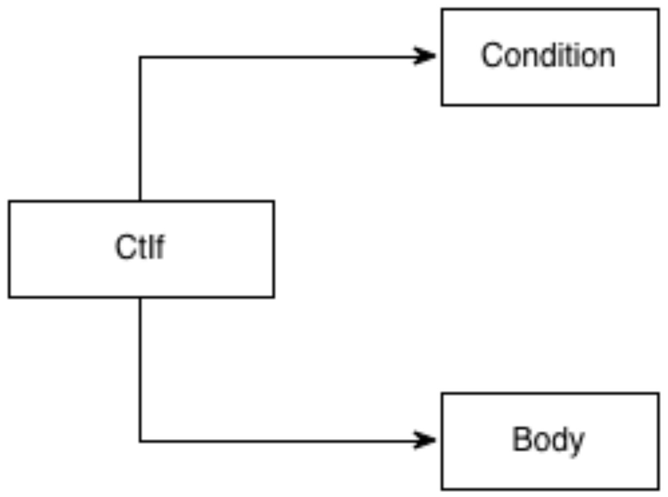
```
if (message == "null")
    java.lang.System.out.println("message is null");
```

Ctlf

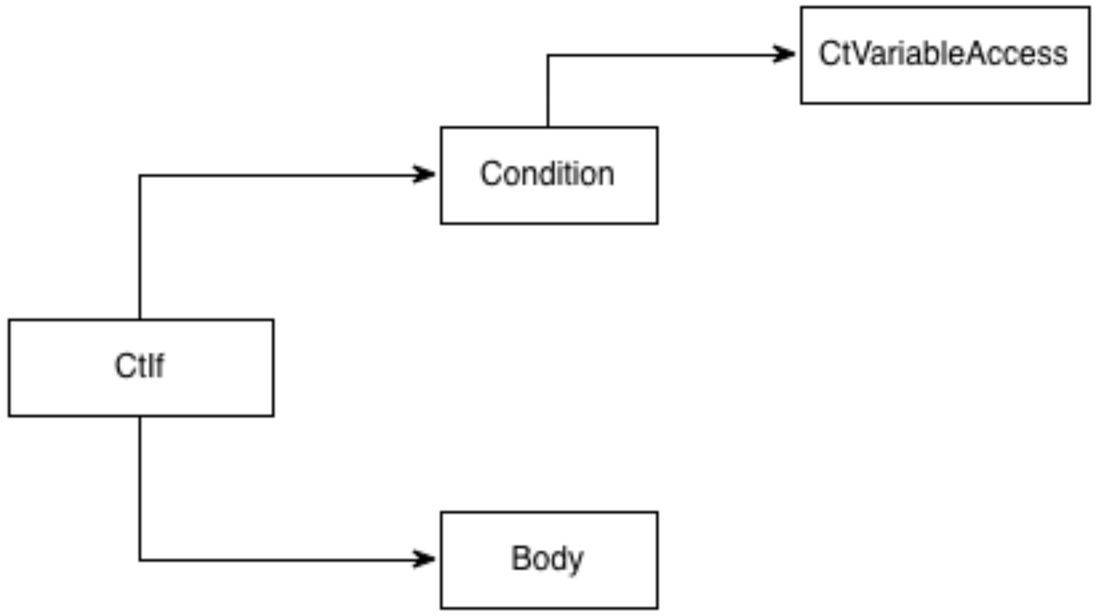
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



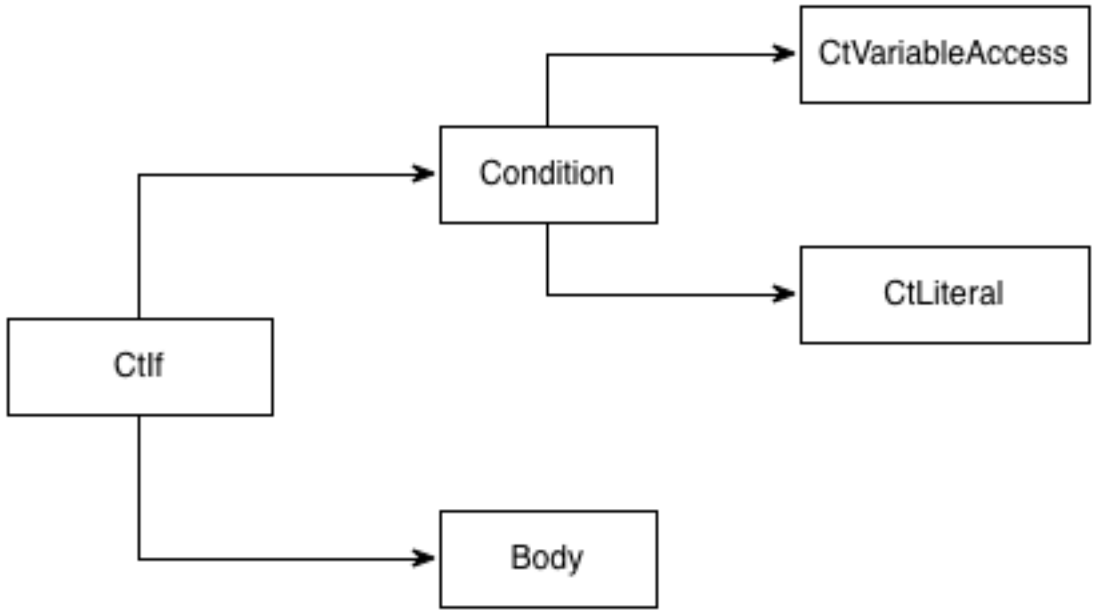
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



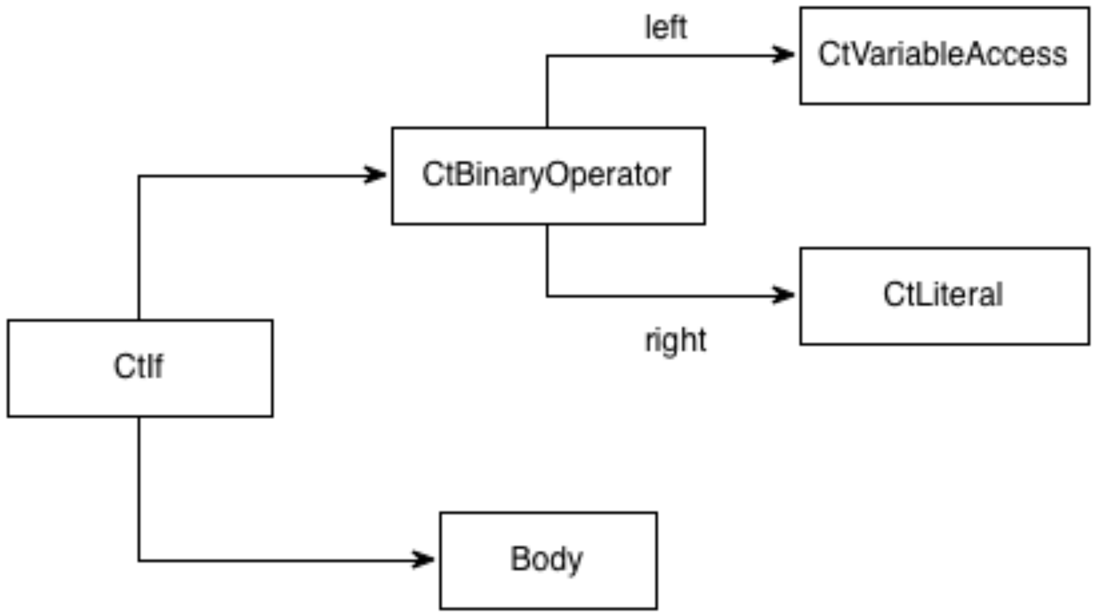
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



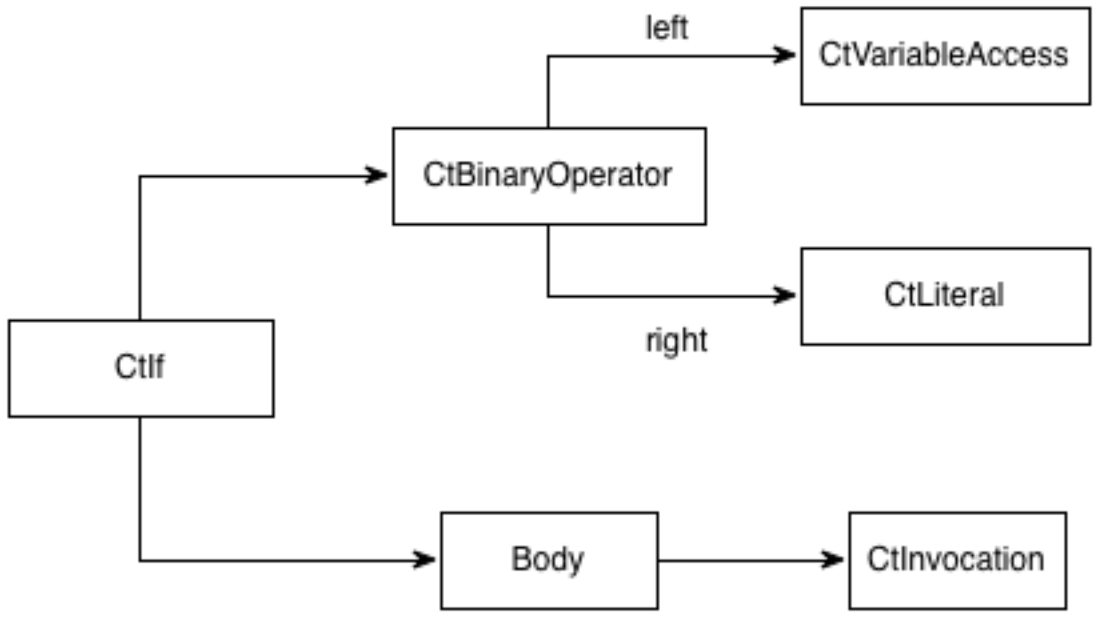
```java
if (message == "null")
    java.lang.System.out.println("message is null");
```



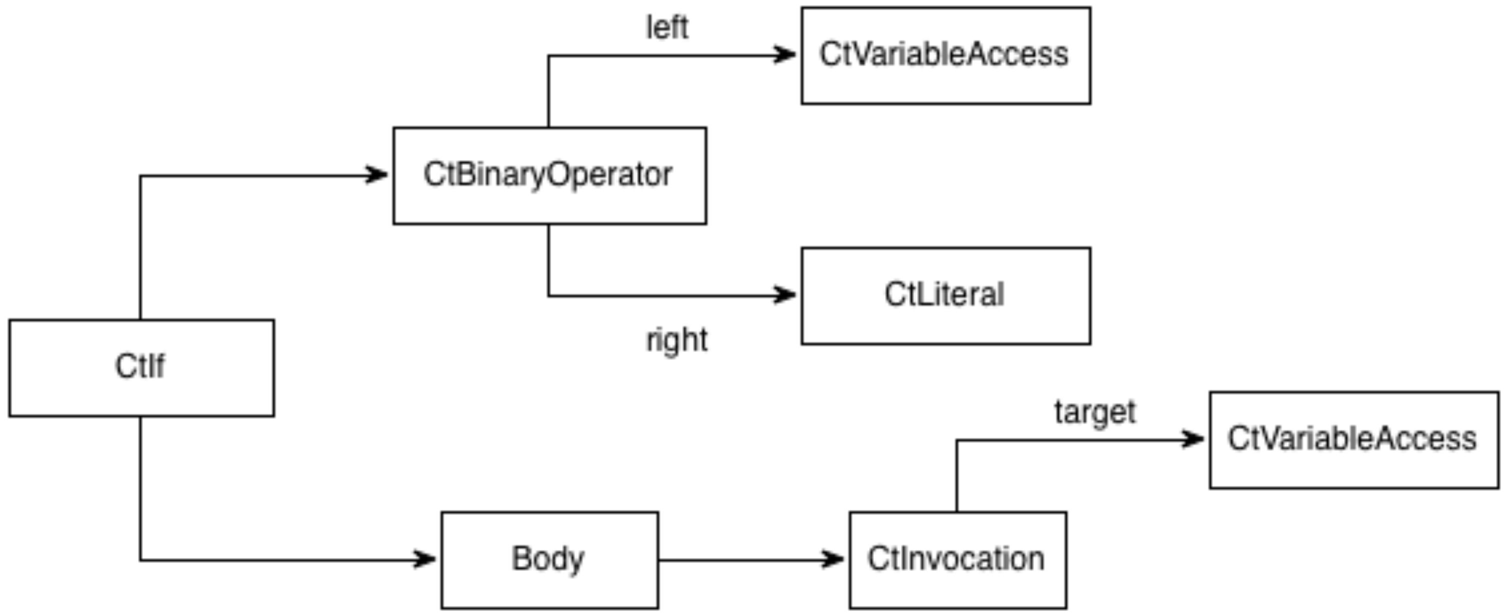
```java
if (message == "null")
    java.lang.System.out.println("message is null");
```



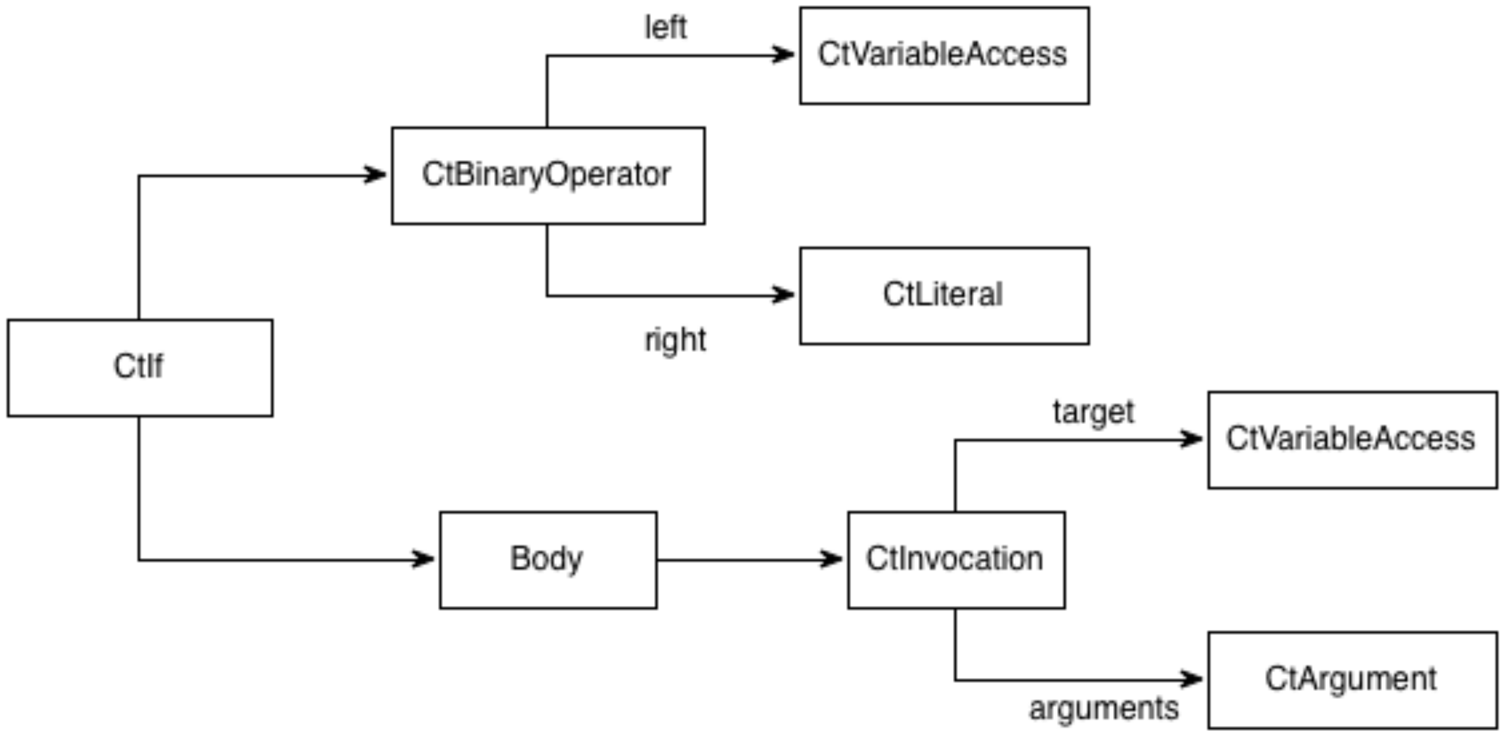
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



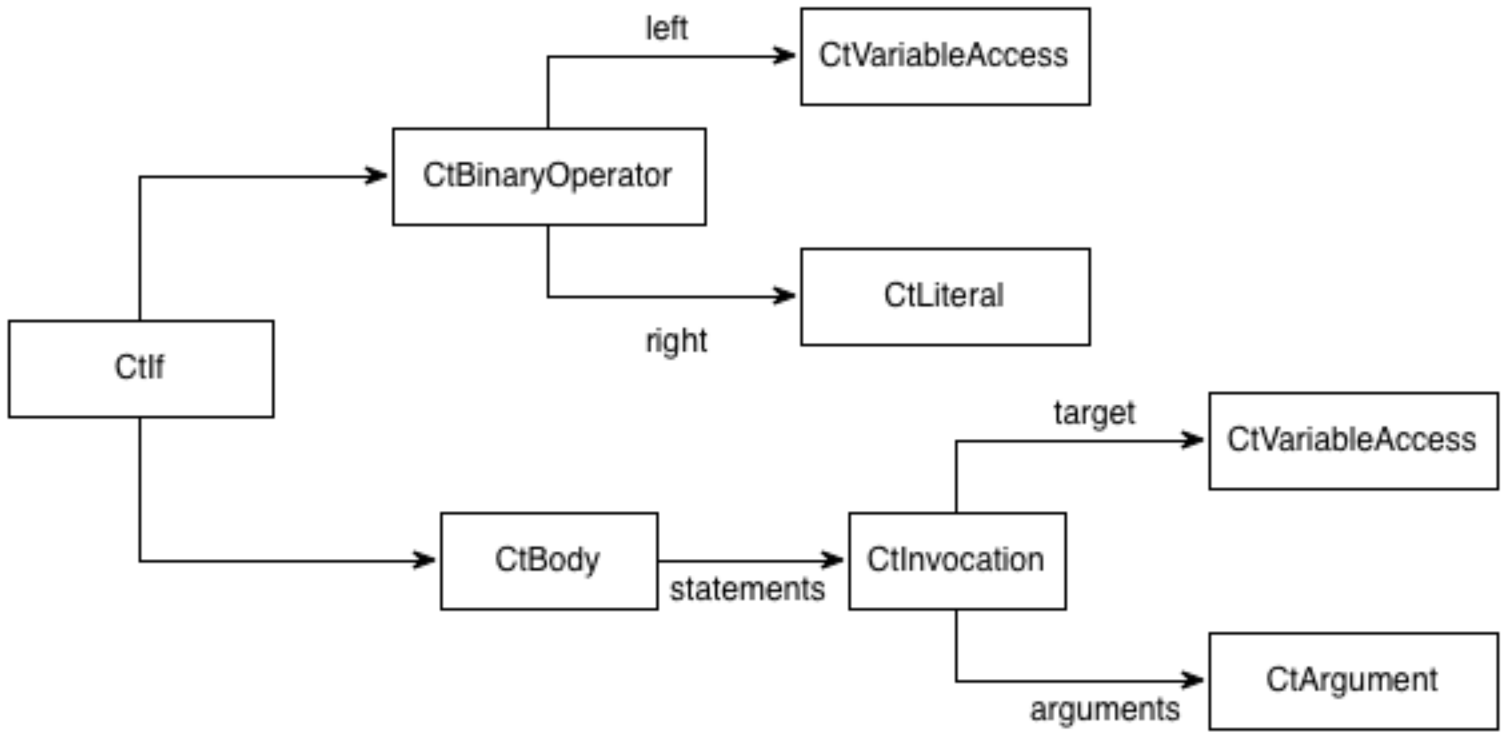
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



```
if (message == "null")
    java.lang.System.out.println("message is null");
```



```java
if (message == "null")
    java.lang.System.out.println("message is null");
```



```
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

```java
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

```java
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

```
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

```java
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

```
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

```java
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

Template

Spoon provides developers a way of writing code transformations: code templates. Those templates are statically type-checked, in order to ensure statically that the generated code will be correct.

http://spoon.gforge.inria.fr/template_definition.html

```java
public class NotNullTemplate extends StatementTemplate {
  @Parameter CtVariableAccess<?> _access_;

  public NotNullTemplate(CtVariableAccess<?> _access_) {
    this._access_ = _access_;
  }

  @Override public void statement() throws Throwable {
    if (_access_.S() == null) {
      System.out.println("Parameter is null");
    }
  }
}
```

```
public class NotNullTemplate extends StatementTemplate {
  @Parameter CtVariableAccess<?> _access_;

  public NotNullTemplate(CtVariableAccess<?> _access_) {
    this._access_ = _access_;
  }

  @Override public void statement() throws Throwable {
    if (_access_.S() == null) {
      System.out.println("Parameter is null");
    }
  }
}
```

```java
public class NotNullTemplate extends StatementTemplate {
  @Parameter CtVariableAccess<?> _access_;

  public NotNullTemplate(CtVariableAccess<?> _access_) {
    this._access_ = _access_;
  }

  @Override public void statement() throws Throwable {
    if (_access_.S() == null) {
      System.out.println("Parameter is null");
    }
  }
}
```

```
public class NotNullTemplate extends StatementTemplate {
  @Parameter CtVariableAccess<?> _access_;

  public NotNullTemplate(CtVariableAccess<?> _access_) {
    this._access_ = _access_;
  }

  @Override public void statement() throws Throwable {
    if (_access_.S() == null) {
      System.out.println("Parameter is null");
    }
  }
}
```

```java
public class NotNullTemplate extends StatementTemplate {
  @Parameter CtVariableAccess<?> _access_;

  public NotNullTemplate(CtVariableAccess<?> _access_) {
    this._access_ = _access_;
  }

  @Override public void statement() throws Throwable {
    if (_access_.S() == null) {
      System.out.println("Parameter is null");
    }
  }
}
```

```java
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  final CtVariableAccess<?> parameter = //
      getFactory().Code().createVariableRead(element.getReference(), false);
  final CtLiteral<?> nullAccess = getFactory().Code().createLiteral("null");
  final CtBinaryOperator<Boolean> binaryOperator = //
      getFactory().Code().createBinaryOperator(parameter, nullAccess, BinaryOperatorKind.EQ);

  final CtCodeSnippetStatement snippet = getFactory().Code().createCodeSnippetStatement(
      "System.out.println(\"" + element.getSimpleName() + " is null\")");

  // Creates condition.
  final CtIf anIf = getFactory().Core().createIf();
  anIf.setCondition(binaryOperator);
  anIf.setThenStatement(snippet.compile());

  // Factory
  final CtExecutable ctExecutable = element.getParent(CtExecutable.class);
  ctExecutable.getBody().insertBegin(anIf);
}
```

```
@Override public void process(NotNull annotation, CtParameter<?> element) {
  System.out.println("annotation found on " + element.getSimpleName());

  new NotNullTemplate(parameter).apply(element.getParent(CtType.class));

}
```

Query

Make a complexe query on a given AST, based on the notion of filter, done in plain Java, in the spirit of an embedded DSL and in one single line of code in the normal cases. i.e., TypeFilter is used to return all elements of a certain type.

```
Query.getElements(factory, new TypeFilter<>(CtParameter.class));

Query.getElements(element, new TypeFilter<>(CtParameter.class));

    element.getElements(new TypeFilter<>(CtParameter.class));
```

http://spoon.gforge.inria.fr/filter.html

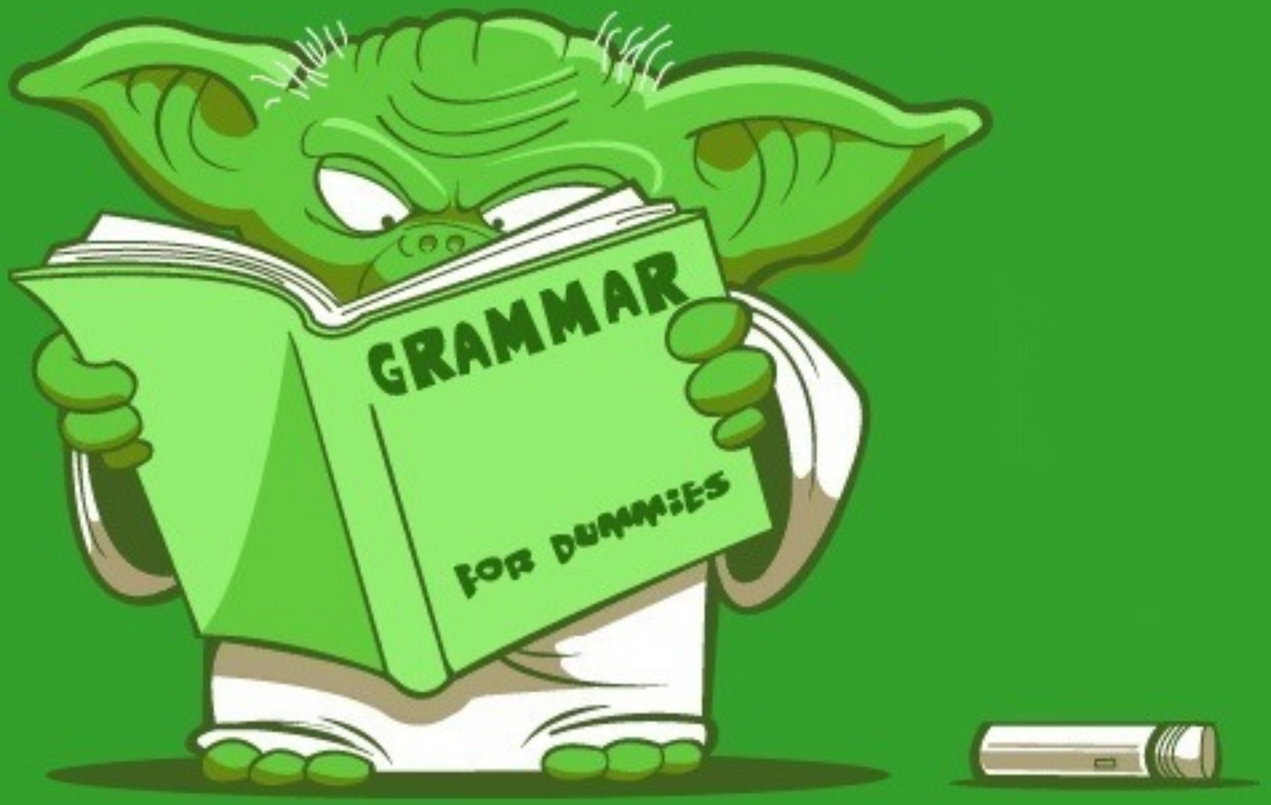
Usage

Dependency

```xml
<dependency>
  <groupId>fr.inria.gforge.spoon</groupId>
  <artifactId>spoon-core</artifactId>
  <version>5.1.1</version>
</dependency>
```

API

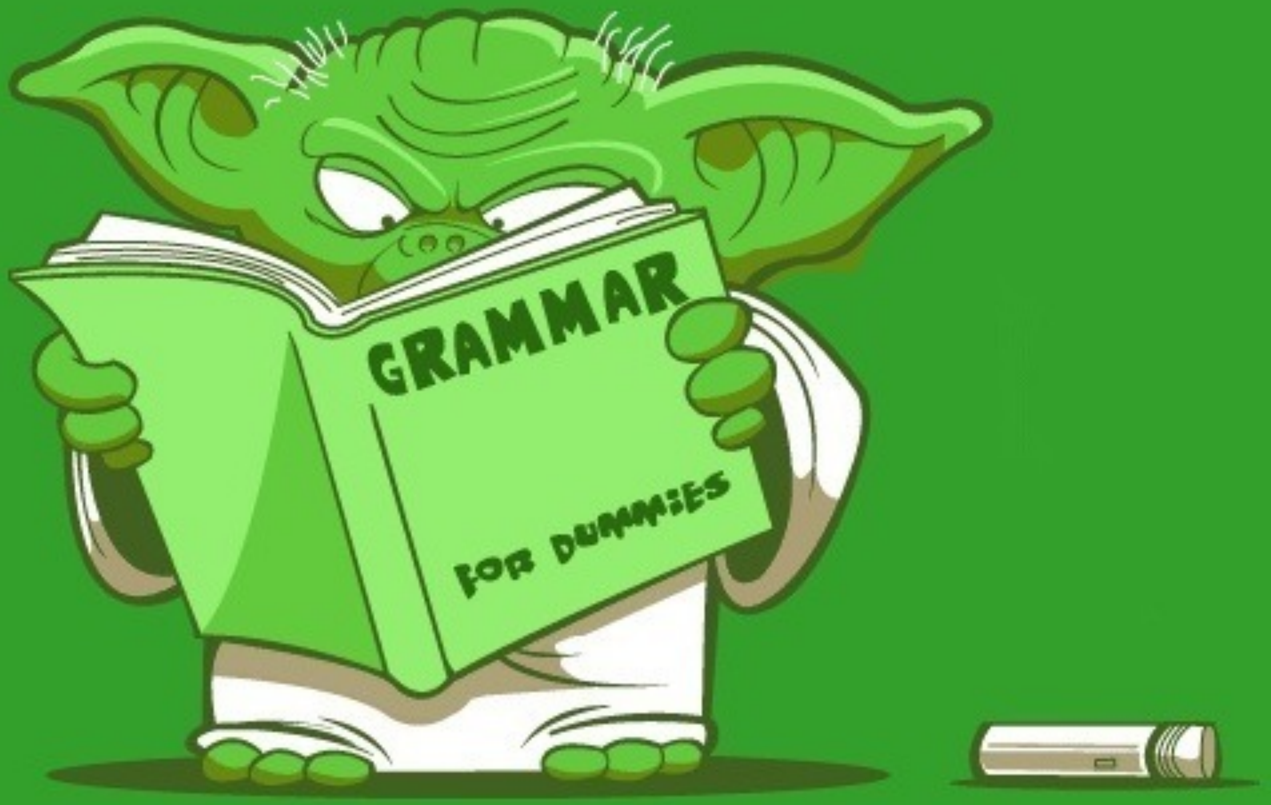
```java
final SpoonAPI spoon = new Launcher();
spoon.addInputResource("/src/main/java/");
spoon.setSourceOutputDirectory("/target/");
spoon.addProcessor(new AwesomeProcessor());
spoon.run();

// Here, make complex query from the
factory.
```



Usage

Maven Plugin

```
<plugin>
  <groupId>fr.inria.gforge.spoon</groupId>
  <artifactId>
    spoon-maven-plugin
  </artifactId>
  <version>2.2</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <processors>
      <processor>
        fr.inria.AwesomeProcessor
      </processor>
    </processors>
  </configuration>
</plugin>
```



Usage

Gradle Plugin

```
buildscript {
  repositories {
    mavenCentral()
    mavenLocal()
  }
  dependencies {
    classpath ‘fr.inria.gforge.spoon:spoon-gradle-plugin:
1.0-SNAPSHOT’
  }
}

apply plugin: "java"
apply plugin: "spoon"

spoon {
 processors = [‘fr.inria.AwesomeProcessor’]
}
```

Thank you!

Gérard Paligot

Practical work

#1

For each interface, count how many classes implement it.

```java
public interface Delicious {
  void prepare(Prepare prepare, int time);

  void make(Cook cook, int time);
}
```

#2

Inserts a probe in all entry methods to display its name and its declaring type.

Before

```java
@Override
public void prepare() {

}
```

After

```java
@Override
public void prepare() {
  System.out.println("enter: Tacos - prepare");
}
```

#2 expert

Inserts a probe in all **exit** methods to display its name and its declaring type.

#3

Adds a probe (with the invocation of a method named isNotNull) to check all parameters not primitives of a method.

Before

```
@Override
public void prepare(Prepare prepare) {

}
```

After

```
@Override
public void prepare(Prepare prepare) {
  fr.inria.tp3.Check.isNotNull(
    prepare, "Parameter %s can't be null.", "prepare");
}
```

#3 expert

Uses an annotation, named @NotNull, to check the parameter annotated.

Before

```java
@Override
public void prepare(@NotNull Prepare prepare) {

}
```

After

```java
@Override
public void prepare(Prepare prepare) {
  fr.inria.tp3.Check.isNotNull(
    prepare, "Parameter %s can't be null.", "prepare");
}
```

#4

Create the annotation @Bound to check the minimum and maximum of a method parameter.

Before

```
@Override
public void prepare(@Bound(min = 0, max = 10) int time) {

}
```

After

```
@Override
public void prepare(int time) {
  if (time > 10.0)
    throw new RuntimeException(
      "out of max bound (" + time + " > 10.0");

  if (time < 0.0)
    throw new RuntimeException(
      "out of min bound (" + time + " < 0.0");
}
```

#5

Create the annotation @RetryOnFailure to relaunch execution of a method if an exception is thrown.

Think to use templates. ;)

```java
@java.lang.Override
public void prepare() {
  int attempt = 0;
  java.lang.Throwable lastTh = null;
  while ((attempt++) < 3) {
    try {
      // Body of the original code here.
    } catch (java.lang.Throwable ex) {
      lastTh = ex;
      if (false) {
        ex.printStackTrace();
      }
      try {
        java.lang.Thread.sleep(50L);
      } catch (java.lang.InterruptedException ignored) {
      }
    }
  }
  if (lastTh != null) {
    throw new java.lang.RuntimeException(lastTh);
  }
}
```

- TP 1: For each interface, count how many classes implement it.
- TP 2: Inserts a probe in all entry methods to display its name and its declaring type.
- TP 2 (expert): Handles exit of methods.
- TP 3: Adds a probe (with the invocation of a method named checkNotNull) to check all parameters not primitives of a method.
- TP 3 (expert): Handles it with an annotation named @NotNull.
- TP 4: Create the annotation @Bound to check the minimum and maximum of a method parameter.
- TP 5 (expert): Create the annotation @RetryOnFailure to relaunch execution of a method if an exception is thrown.