


UFR IEEA
Formations en
Informatique de
Lille 1

# Conception Orientée Objet

Romain Rouvoy
Licence mention Informatique
Université Lille 1

# Menu du  jour

1. Coder c’est douter
2. De la génération
3. À l’analyse

# Mauvaise conception ?

Où est le problème ?

# Derrière le clavier !

# Coder, c’est douter ?

# Pourquoi ne pas générer ?

- Beaucoup d'erreurs proviennent de la routine
  - Coder des parties répétitives
- Par exemple
  - Générer des getters/setters

```
StringBuilder sourceBuilder = new StringBuilder();
sourceBuilder.append("package fil.coo;\n\n");
sourceBuilder.append("import java.util.*;\n\n");
sourceBuilder.append("public class MyBean {");

for (DBField dbField:getFields) {
  // bean attribute
        sourceBuilder.append("\tprivate ")
                .append(dbField.getType)
                .append(toFieldName(dbField.getName()))
                .append(" = null;\n");

  // setter method
        sourceBuilder.append("\tpublic void ")
                .append(toSetterName(dbField.getName()))
                .append("(");
                .append(dbField.getType)
                .append(toFieldName(dbField.getName()))
                .append(") {\n")
                .append("\t\tthis.")
                .append(toFieldName(dbField.getName()))
                .append(" = ")
                .append(toFieldName(dbField.getName()))
                .append(";\n\t}");

   // getter method...
sourceBuilder.append("\t}\n}\n");
```

```
import fil.coo;import java.util.*;
public class MyBean {
#foreach( $field in getFields())
	private $field.type $field.name = null;

	public void $field.setter()($field.type $field.name) {
		this.$field.name = $field.name ;
	}

	public void #field.getter()() {
		return this.$field.name ;
	}
#end
}
```

# Template-based generation

- Forces
  - Très «visuel»
  - Facile à aborder
  - Peut générer tout (et n'importe quoi)
- Faiblesses
  - Pas de typage
  - Difficulté de réutilisation (donc copier/coller)
  - Rapidement complexe (pré-*processing*)

```
$petList.size() Pets on Sale!
 We are proud to offer these fine pets
 at these amazing prices.  This month only,
 choose from:
 #foreach( $pet in $petList )
   $pet.name for only $pet.price
 #end
  Call Today!
```

# Template-based generation

- Nombreux outils
  - Apache Velocity
  - Freemarker
  - Eclipse JET
  - …
- Très utilisée
  - Web *frameworks* (génération HTML)
  - Persistence *frameworks* (génération POJO)
  - Maven archetypes
  - …

# API-based generation

- Forces
  - Typage fort (correct par construction)
  - Réutilisation aisée (comme tout programme)

- Faiblesses
  - Potentiellement verbeux
  - Pas très visuel

```java
package com.example.helloworld;

public final class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

```
MethodSpec main = MethodSpec.methodBuilder("main")
    .addModifiers(Modifier.PUBLIC, Modifier.STATIC)
    .returns(void.class)
    .addParameter(String[].class, "args")
    .addStatement("$T.out.println($S)", System.class, "Hello, World!")
    .build();

TypeSpec helloWorld = TypeSpec.classBuilder("HelloWorld")
    .addModifiers(Modifier.PUBLIC, Modifier.FINAL)
    .addMethod(main)
    .build();

JavaFile javaFile = JavaFile.builder("fil.coo", helloWorld)
    .build();

javaFile.writeTo(System.out);
```

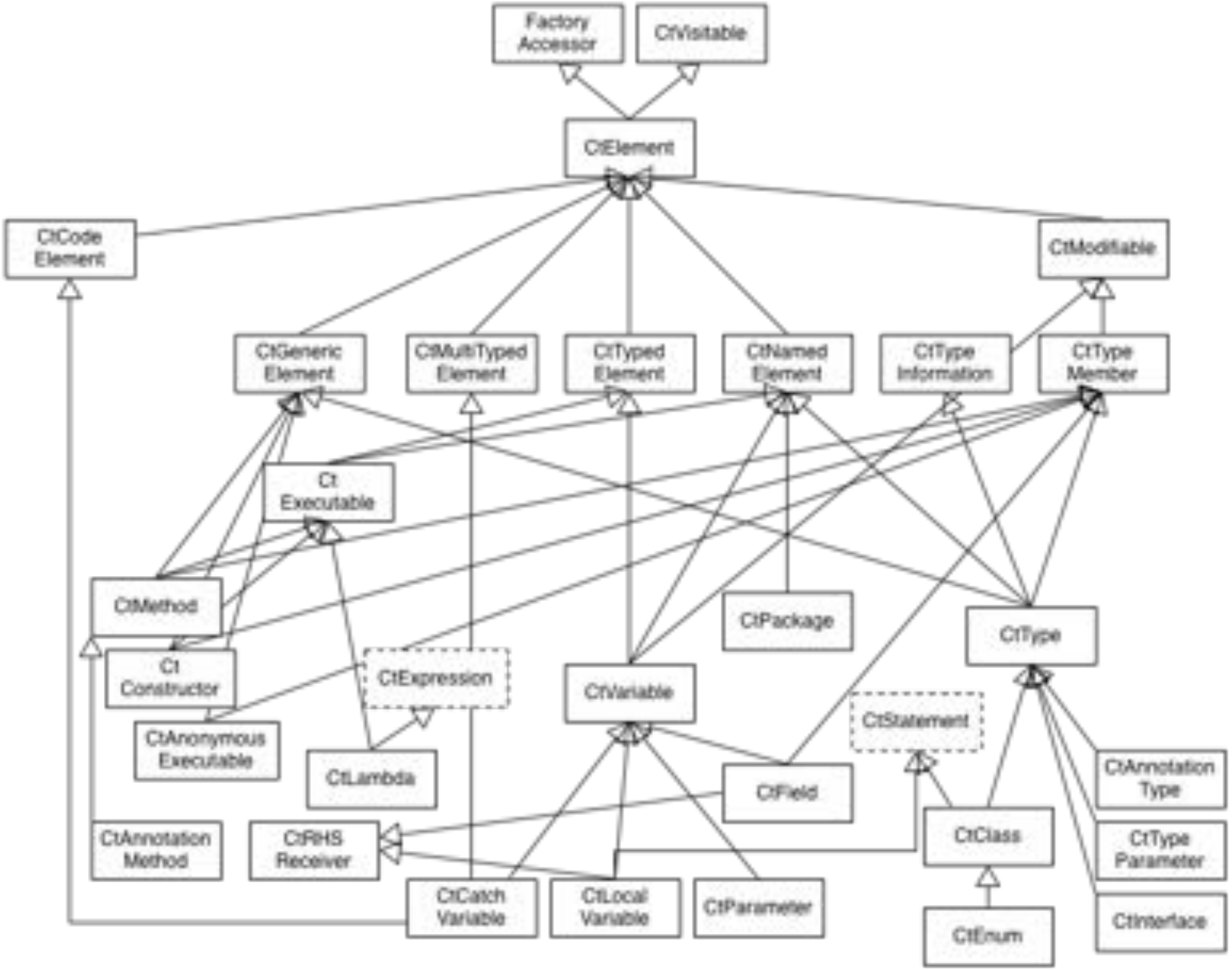
# Et la qualité du code dans tout ça ?

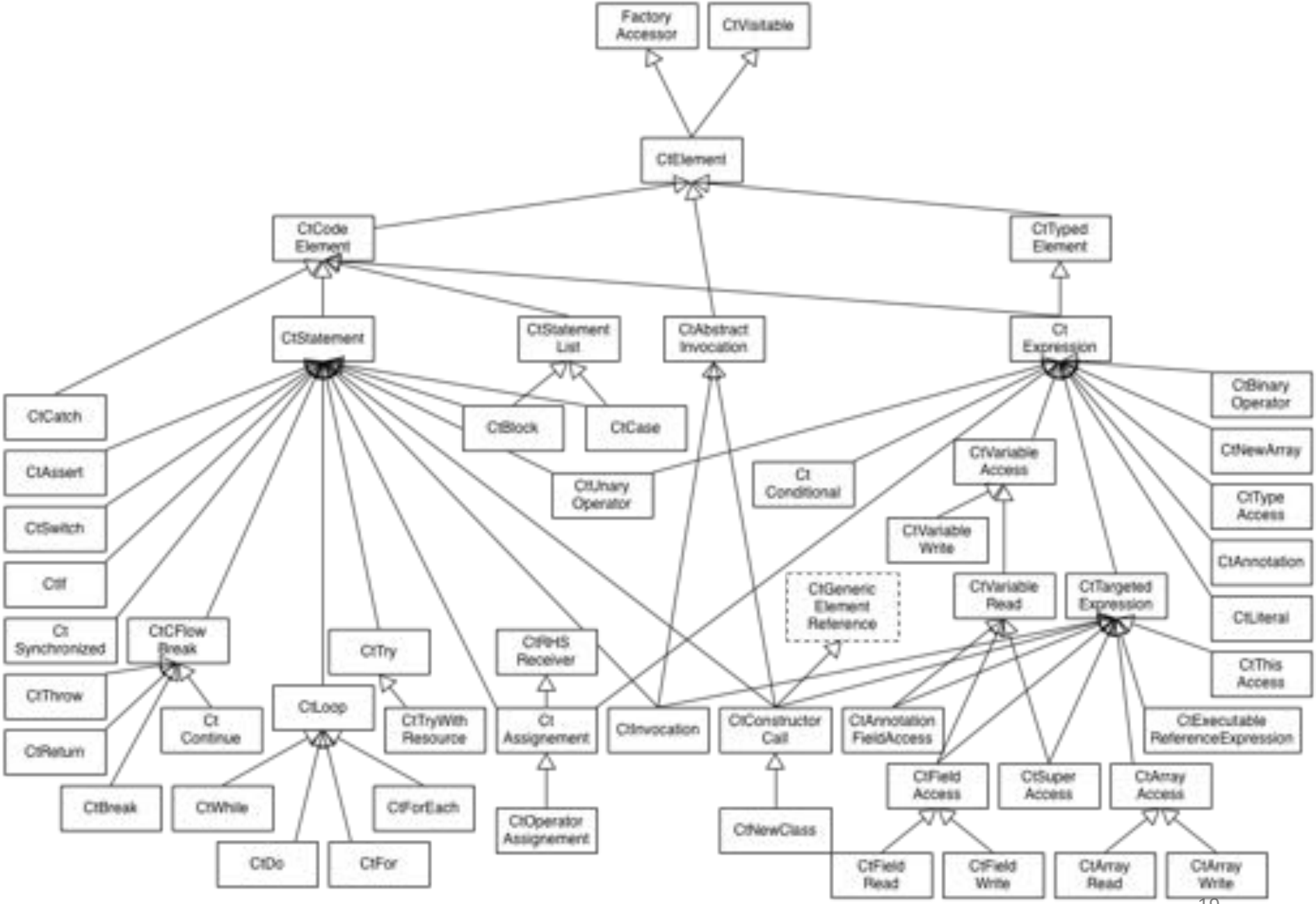
```java
package fr.inria.gforge.spoon.processors;

import org.apache.log4j.Level;
import spoon.processing.AbstractProcessor;
import spoon.reflect.code.CtCatch;

/**
 * Reports warnings when empty catch blocks are found.
 */
public class CatchProcessor extends AbstractProcessor<CtCatch> {
    public void process(CtCatch element) {
        if (element.getBody().getStatements().size() == 0) {
            getFactory().getEnvironment().report(this,
                Level.WARN, element, "empty catch clause");
        }
    }
}
```

# Spoon

- Projet OSS développé par Inria depuis 2005
- Couvre la génération, l'analyse et la transformation

- Charge le code en un modèle
  - Votre code devient une donnée manipulable
  - Possibilité de lire/modifier/ajouter des éléments
  - Correct par construction


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


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

```
<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.gforge.spoon.processors.CatchProcessor</processor>
    </processors>
  </configuration>
  <!-- To be sure that you use the latest version of Spoon, specify it as
dependency. -->
  <dependencies>
    <dependency>
      <groupId>fr.inria.gforge.spoon</groupId>
      <artifactId>spoon-core</artifactId>
     <version>5.4.0</version>
    </dependency>
  </dependencies>
</plugin>
```

# Factory

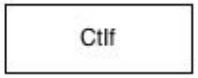
Create new elements, fill their data and add them in an existing AST. The factory is the entry point to do that and each factory has them own responsibility. i.e., CoreFactory creates empty nodes and CodeFactory creates a node ready to be printed.

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

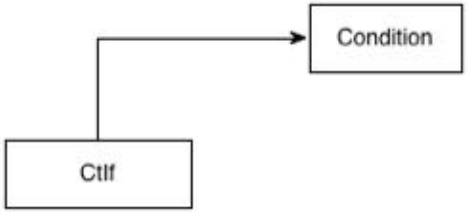
http://spoon.gforge.inria.fr/factories.html

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

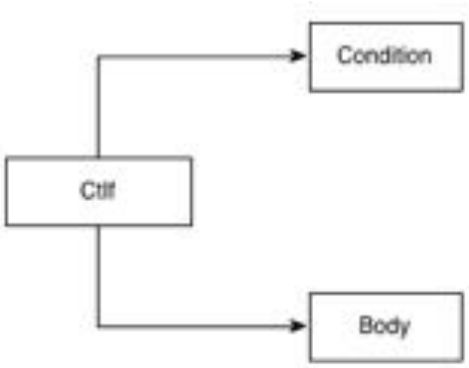
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



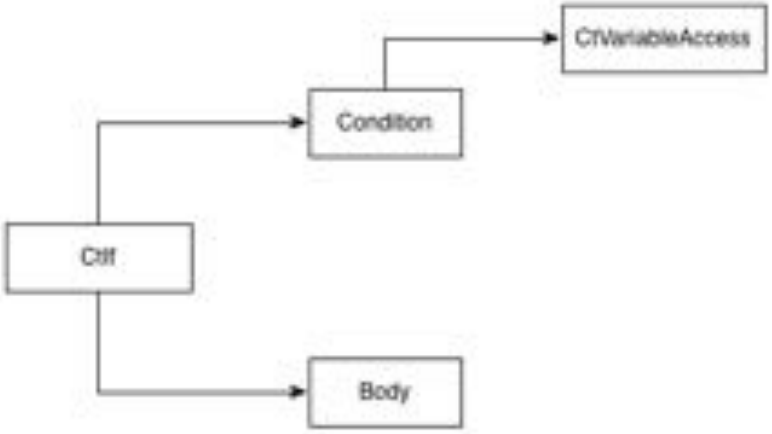
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



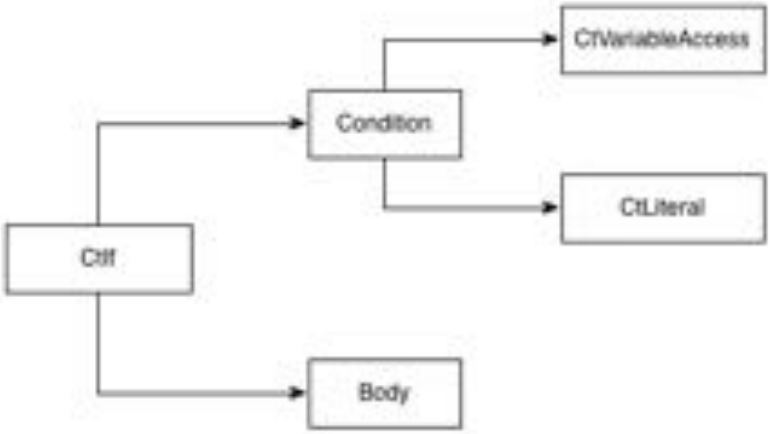
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



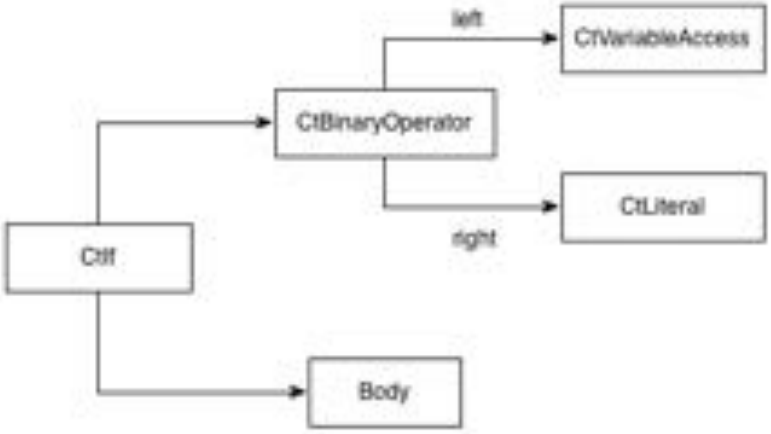
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



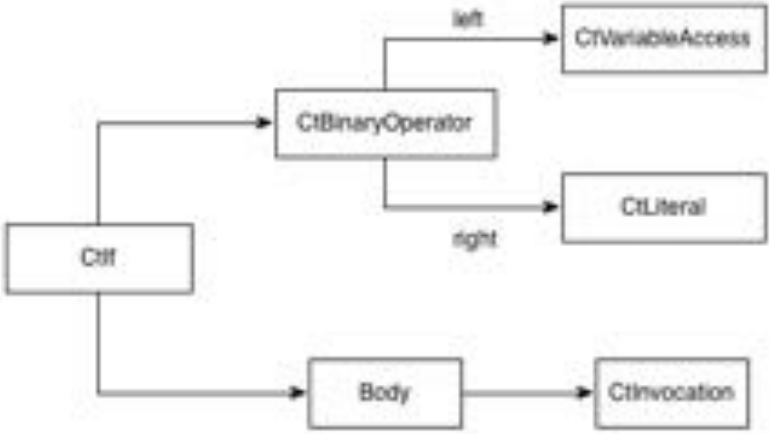
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



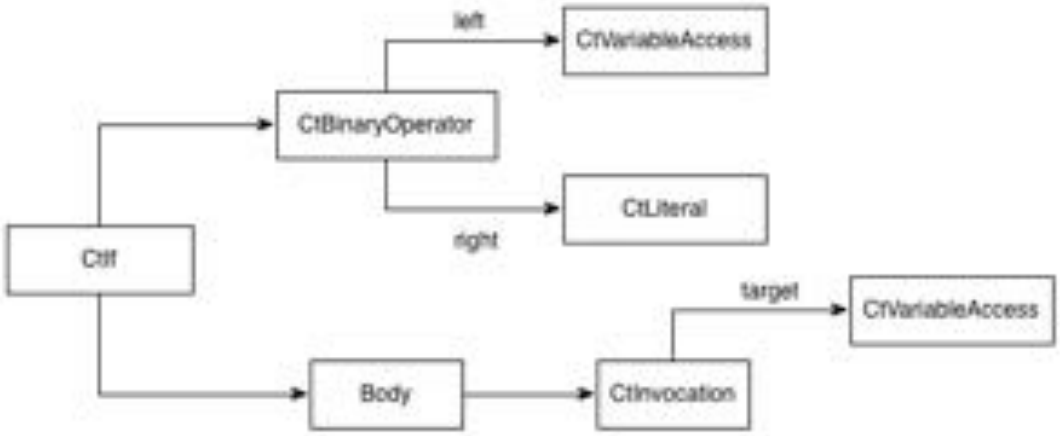
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



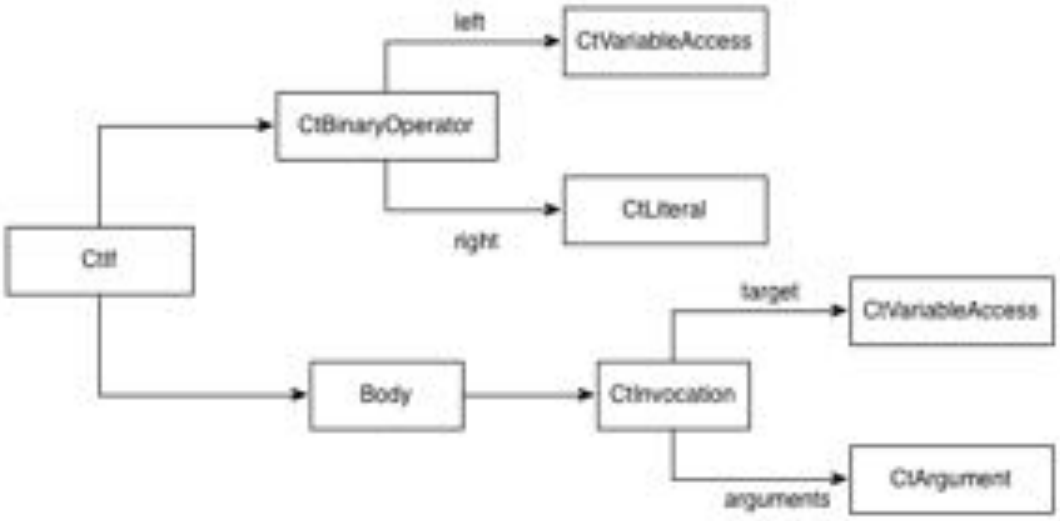
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



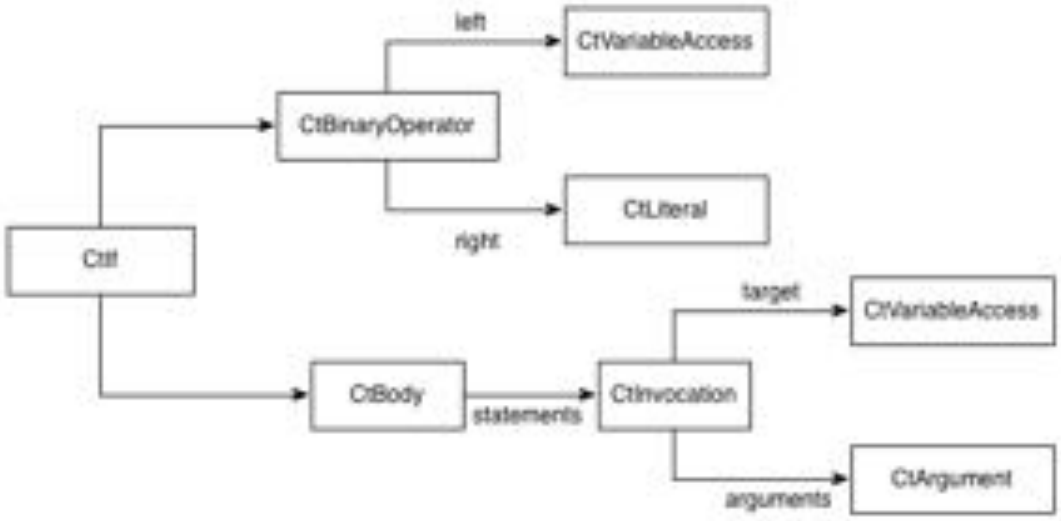
```
if (message == "null")
    java.lang.System.out.println("message is null");
```



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



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



```
@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);
}
```

```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);
}
```

```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);
}
```

# Exemples d'utilisation de Spoon

- Program Analysis
  - The **CatchProcessor** detects *empty catch blocks*
  - The **ReferenceProcessor** detects *circular references* between packages
  - This **Factory** example detects *wrong uses of the factory* pattern
- Program Transformation
  - The **NotNullProcessor** adds a *not-null check* for all method parameters
  - The **MutationProcessor** randomly *mutates some parts of the abstract syntax tree* for mutation testing
- Annotation Processing
  - The **N-ton** example introduces a N-ton design pattern (extension of singleton but for N instances) into a target class. It inserts static fields, methods, and initializer code into constructors
  - The **Database** access example shows how to use annotation processing to add persistence into a POJO
  - The **Visitor** example implements a visitor pattern by automatically introducing an accept method in a visited type hierarchy
  - The **Field Access** example implements a refactoring that introduces setters and getters for the fields annotated with the Access annotation and that replaces all the direct accesses to these fields by calls to its new getters and setters

# Et quoi d'autre ?

```
void noTreeSetInSpoon() throws Exception {
    // we don't use TreeSet, because they implicitly depend on Comparable
    SpoonAPI spoon = new Launcher();
    spoon.addInputResource("src/main/java/");
    spoon.buildModel();

    assertEquals(0, spoon.getFactory().Package().getRootPackage()
                          .getElements(new AbstractFilter<CtConstructorCall>() {
       @Override
        public boolean matches(CtConstructorCall element) {
            return element.getType().getActualClass().equals(TreeSet.class);
       };
    }).size());
}
```

# Comment s’assurer qu’un générateur est correct ?

# Par le test !

```
final SpoonAPI spoon = new Launcher();
spoon.addInputResource("path/of/my/file/Foo.java");
spoon.run();

final CtType<Foo> type = spoon.getFactory().Type().get(Foo.class);
assertThat(type.getField("i"))
        .withProcessor(new MyProcessor()).isEqualTo("public int j;");
```

# En résumé

- Génération de code pour limiter les erreurs «bêtes» (copier/coller)
- Analyse de code pour détecter les erreurs
- Transformation de code pour automatiser la production (lien Analyse => Génération)

# C’était le dernier épisode…

- Ce que vous avez vu
  - Quelques principes clés / méthodologies **pour vous aider**
  - Notions d’antipatrons **à éviter** et de métriques de code **à comprendre**
  - Les apports de la génération et analyse de code
  - Le tout supporté par un outillage (compatible Maven/Eclipse)
  - Le tout valable pour tous vos développements (POJO, Android, Web, Go, Python…)
- Ce qu’on attend de vous
  - **Approfondir chacun des points évoqués en cours**
  - Pratiquer, pratiquer, pratiquer ! (en TP, en projet, en stage, en OSS, etc.)
- Est-ce que c’est au programme du partiel ?
  - Bien évidemment !