Spoon: Open Source Library to Analyze, Rewrite, Transform, Transpile Java Source Code - A getting started

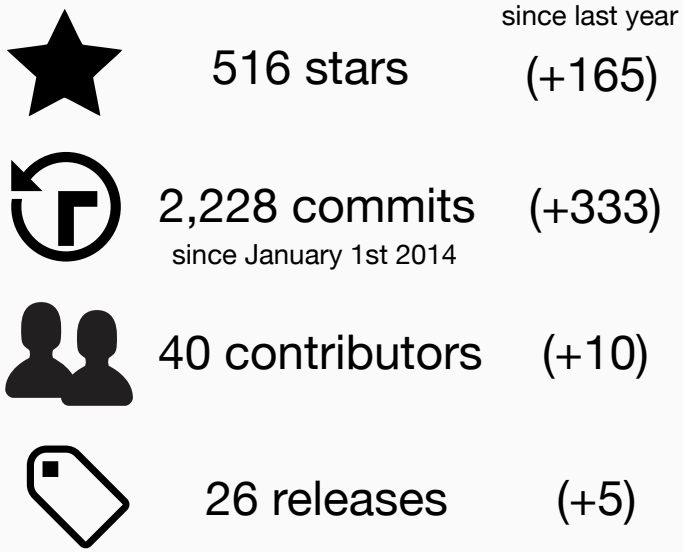
Simon Urli
simon.urli@inria.fr
https://github.com/INRIA/spoon

7th, June 2018

OW2Con'18

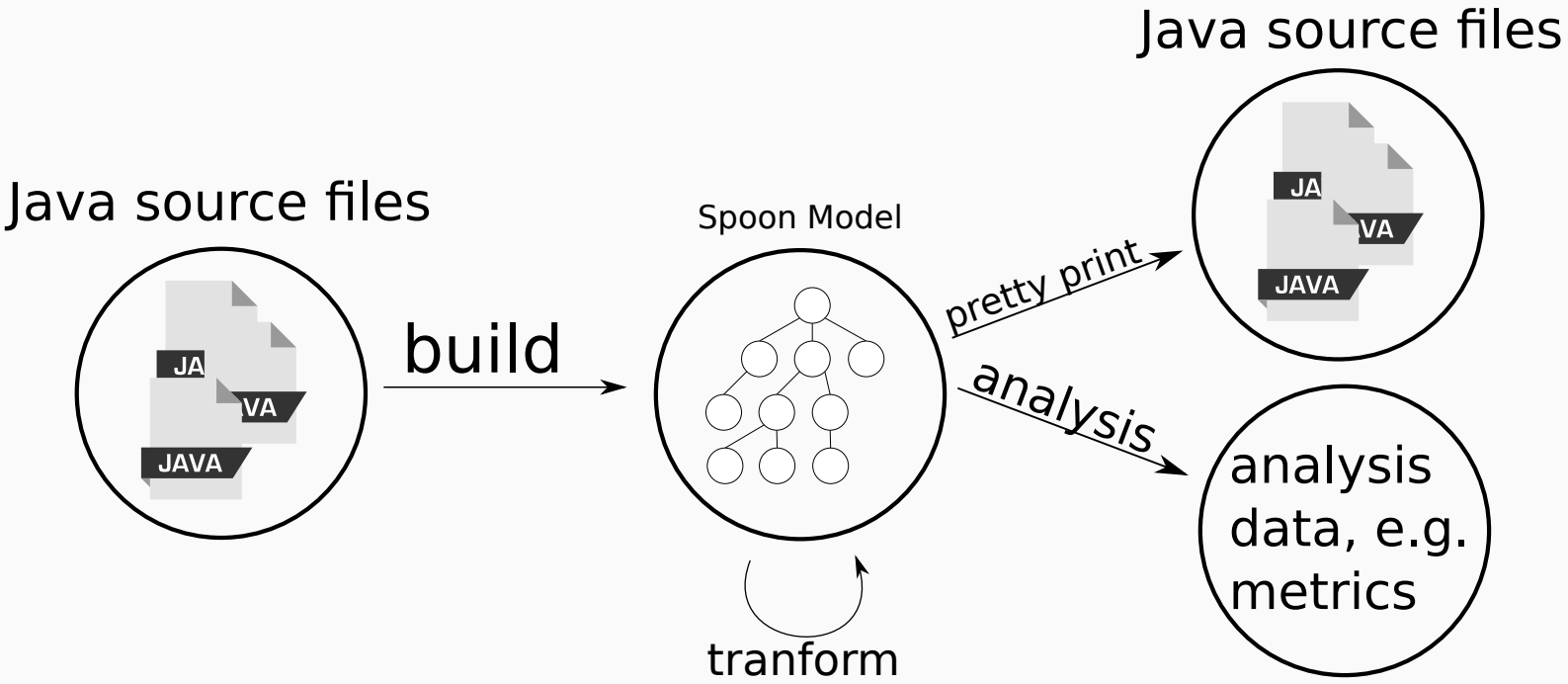
History & Community

- 2005: project creation at INRIA Lille
- 2014: Spoon on Github
- 2016: Spoon at OW2
- 2017: Community Award at OW2Con



Spoon in a nutshell

A library to write your own analysis and transformation in java.



Usage examples:

- test improvement
- transpilation, e.g. Java to Javascript
- detection of bad smells
- automatic refactoring

Spoon Overview

Spoon standard process

1. Build a model of your project
2. Query and analyze interesting parts
3. Transform what needs to be transformed
4. Output a transformed source code

Building a Spoon model

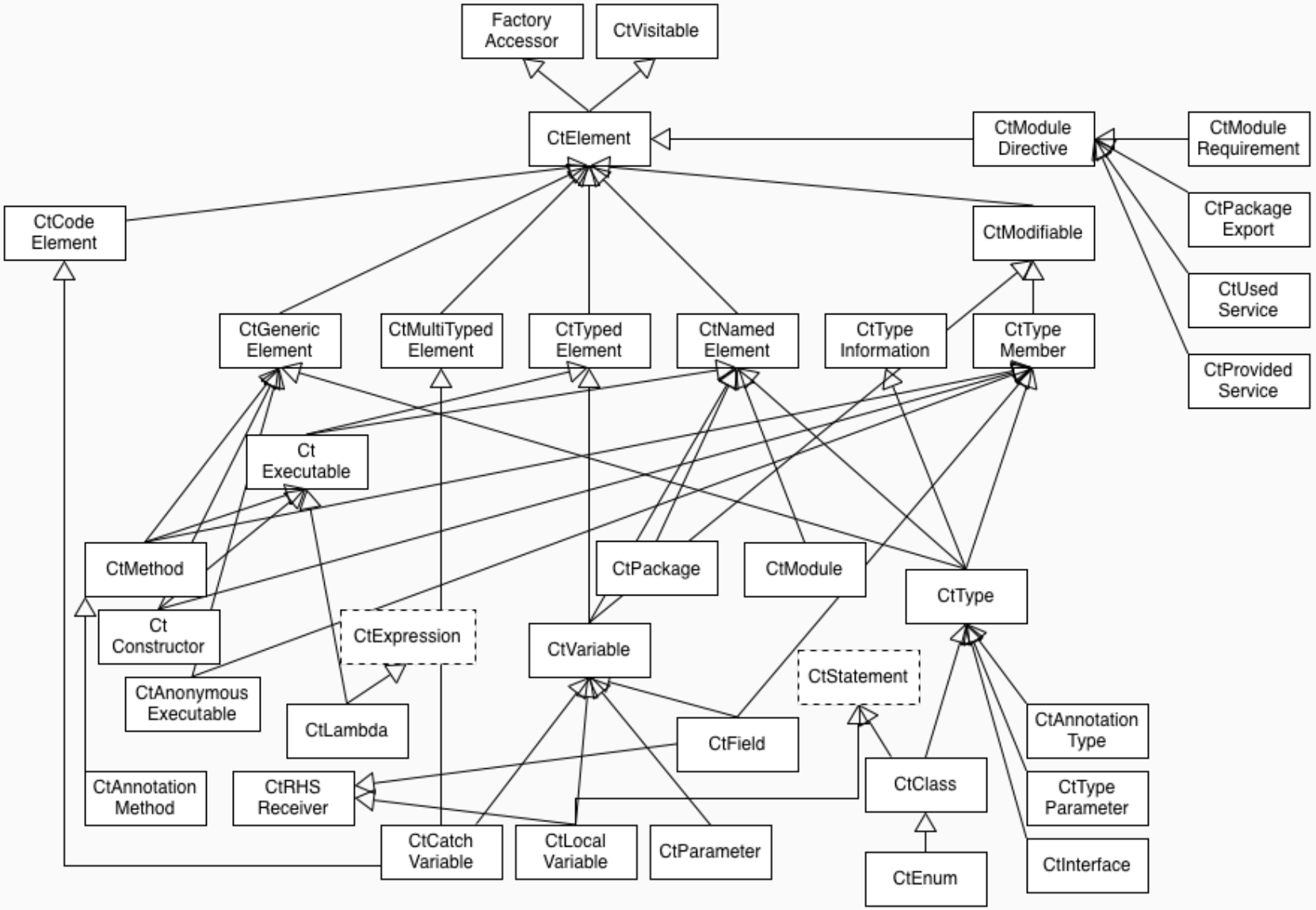
Inputs:

- Source code (i.e. java files or snippets)
- Many options like classpath, Java version, ...

Output:

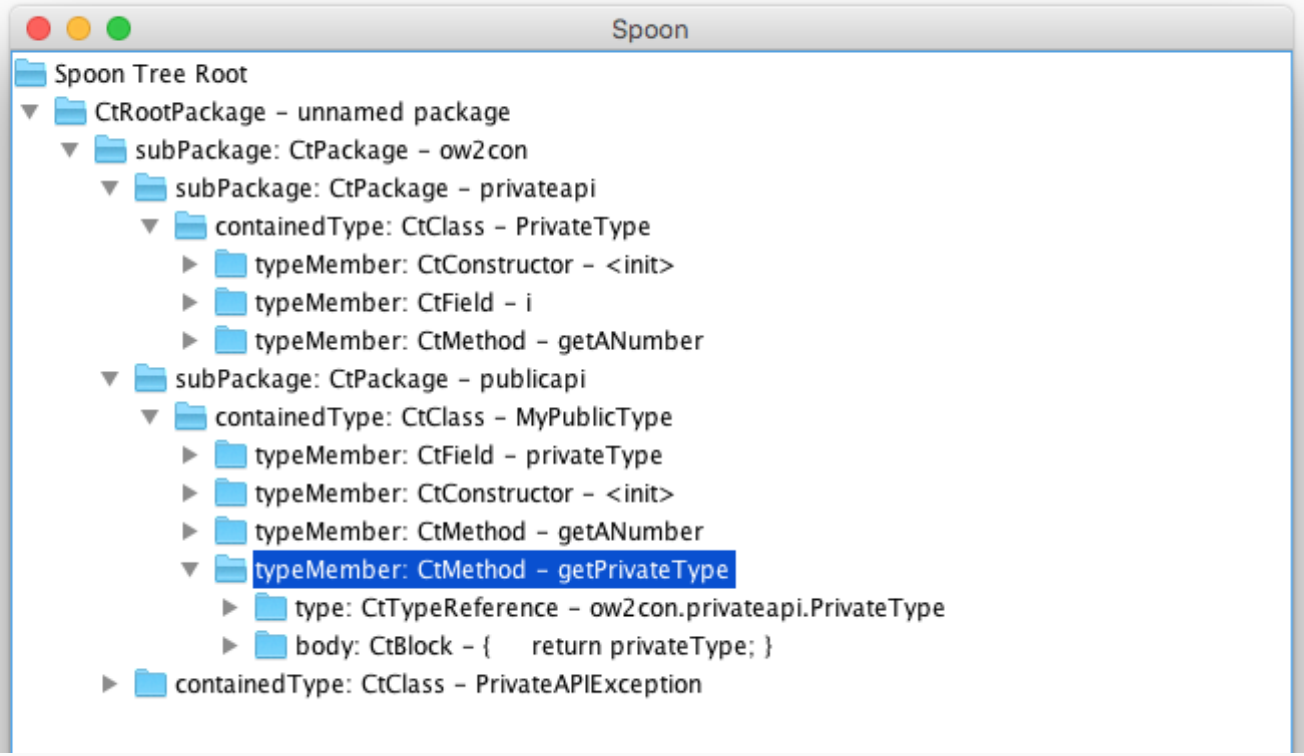
- Spoon model

Spoon AST Metamodel (excerpt)



Spoon model example

```
193-51-236-54:~ urli$ java -jar spoon-core-6.2.0-jar-with-dependencies.jar --gui -i Github/ow2condemo/src/main/java/
```



Query and analyze model

Different ways of doing it:

- Using Query API (filter-chaining API)
- Using XPath-like API

Transform model

Transforming model involve performing basic CRUD operations on the nodes of the model.

Transform model

Transforming model involve performing basic CRUD operations on the nodes of the model.

- Create a node

```
Factory factory = launcher.getFactory();
CtClass aClass = factory.createClass( qualifiedName: "my.org.MyClass");
```

Transform model

Transforming model involve performing basic CRUD operations on the nodes of the model.

- Create a node
- Update a node

```
aClass.setSimpleName("myNewName");
CtMethod myMethod = factory.createMethod();
aClass.addMethod(myMethod);
```

Transform model

Transforming model involve performing basic CRUD operations on the nodes of the model.

- Create a node
- Update a node
- Delete a node

```
aClass.removeMethod(myMethod);
```

Output the model

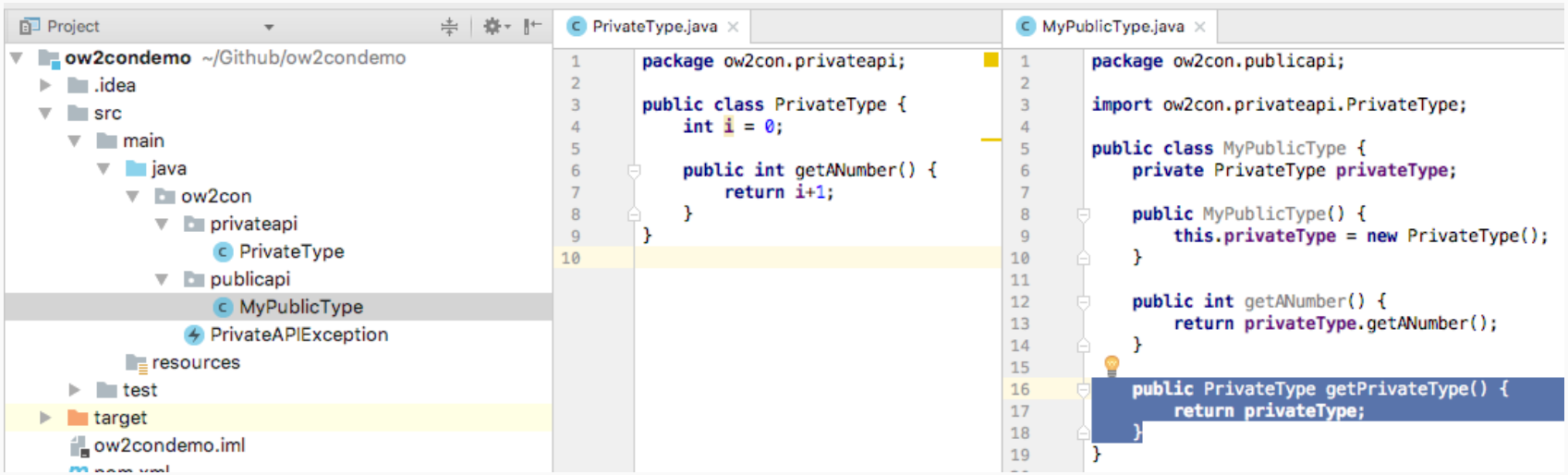
Spoon provides a default Java pretty-printer:

- output the model using standard Java code-style
- create files hierarchy based on the compilation units or types

Quick scenario

Scenario

You want to enforce that all types from the **private API** are not returned from the **public API**.



Building the model

Building a Spoon model from Maven

From a Maven Project:

```
MavenLauncher launcher = new MavenLauncher(
          mavenProject: "path/to/my/project",
          MavenLauncher.SOURCE_TYPE.APP_SOURCE);

CtModel model = launcher.buildModel();
```

Automatically get the libraries from Maven dependencies.

Building a Spoon model from sources

From input source of a project:

```
Launcher launcher = new Launcher();
launcher.addInputResource( path: "/path/to/sources");
launcher.getEnvironment().setNoClasspath(true);
launcher.getEnvironment().setSourceClasspath(
        "lib1.jar:lib2.jar".split( regex: ":")
);
launcher.getEnvironment().setComplianceLevel(7);
CtModel model = launcher.buildModel();
```

Be careful with the classpath of your project. Don't hesitate to use `noclasspath` mode. (default in next version of Spoon)

Query and analyze model: scenario

Goal:

1. retrieve all types from the public API package

```
List<CtMethod> methodList = model.
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "ow2con")).
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "publicapi")).
```

Query and analyze model: scenario

Goal:

1. retrieve all types from the public API package
2. retrieve all methods from those types

```
List<CtMethod> methodList = model.
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "ow2con")).
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "publicapi")).
        filterChildren(new TypeFilter<CtMethod>(CtMethod.class)).
```

Query and analyze model: scenario

Goal:

1. retrieve all types from the public API package
2. retrieve all methods from those types
3. retrieve methods that are public

```
List<CtMethod> methodList = model.
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "ow2con")).
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "publicapi")).
        filterChildren(new TypeFilter<CtMethod>(CtMethod.class)).
        filterChildren(new Filter<CtMethod>() {
            @Override
            public boolean matches(CtMethod element) {
                boolean isPublic = element.isPublic();
```

Query and analyze model: scenario

Goal:

1. retrieve all types from the public API package
2. retrieve all methods from those types
3. retrieve methods that are public
4. and returns a type from the private API package

```
List<CtMethod> methodList = model.
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "ow2con")).
        filterChildren(new NamedElementFilter<CtPackage>(CtPackage.class,  name: "publicapi")).
        filterChildren(new TypeFilter<CtMethod>(CtMethod.class)).
        filterChildren(new Filter<CtMethod>() {
            @Override
            public boolean matches(CtMethod element) {
                boolean isPublic = element.isPublic();
                CtTypeReference returnType = element.getType();
                String privateApiPackage = "ow2con.privateapi";
                boolean isTypeFromPrivateApi = returnType.getQualifiedName().contains(privateApiPackage);
                return isPublic && isTypeFromPrivateApi;
            }
        }).list();
```

Transform model: scenario - Plan

Goal: prevent using the detected methods

How to do it?

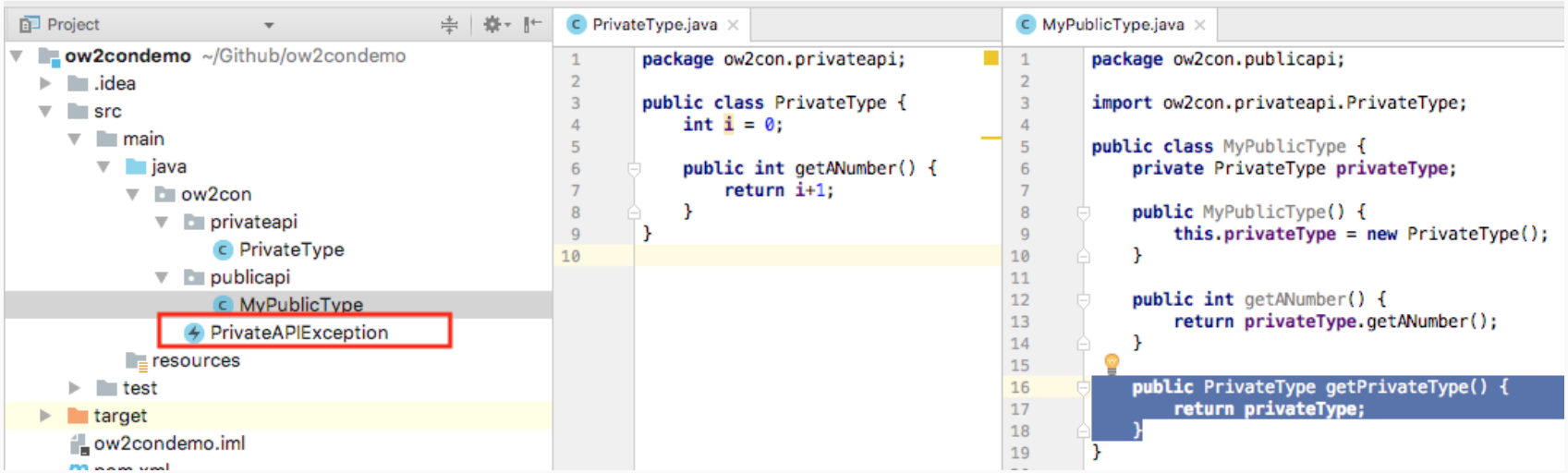
- Comments all statements
- Throw a dedicated RuntimeException
- Add an explanatory comment

Remember: it's only an educational example :-)

Transform model: scenario - Which exception?

What exception should be throw?

We actually already have one!



Transform model: hands on!

Goal: Get the exception class.

```
Factory factory = launcher.getFactory();
CtClass<? extends Throwable> exceptionClass = factory.createClass( qualifiedName: "ow2con.PrivateAPIException");
```

Transform model: hands on!

Goal: Create an instance of the exception class.

```
Factory factory = launcher.getFactory();
CtClass<? extends Throwable> exceptionClass = factory.createClass( qualifiedName: "ow2con.PrivateAPIException");
CtConstructorCall<? extends Throwable> exceptionInstance = factory.createConstructorCall(exceptionClass.getReference());
```

Transform model: hands on!

Goal: Iterate over the queried methods.

```
Factory factory = launcher.getFactory();
CtClass<? extends Throwable> exceptionClass = factory.createClass( qualifiedName: "ow2con.PrivateAPIException");
CtConstructorCall<? extends Throwable> exceptionInstance = factory.createConstructorCall(exceptionClass.getReference());

for (CtMethod method : methodList) {
    CtBlock methodBody = method.getBody();
    List<CtComment> bodyComments = new ArrayList<>();
```

Transform model: hands on!

Goal: Create comments for all statements.

```
Factory factory = launcher.getFactory();
CtClass<? extends Throwable> exceptionClass = factory.createClass( qualifiedName: "ow2con.PrivateAPIException");
CtConstructorCall<? extends Throwable> exceptionInstance = factory.createConstructorCall(exceptionClass.getReference());

for (CtMethod method : methodList) {
    CtBlock methodBody = method.getBody();
    List<CtComment> bodyComments = new ArrayList<>();

    ArrayList<CtStatement> ctStatements = new ArrayList<>(methodBody.getStatements());

    for (CtStatement ctStatement : ctStatements) {
        String statement = ctStatement.toString();
        CtComment statementAsComment = factory.createInlineComment(statement);
        bodyComments.add(statementAsComment);
        methodBody.removeStatement(ctStatement);
    }
```

Transform model: hands on!

Goal: Create a new **throw** statement with the call to the new instance of the exception.

```
Factory factory = launcher.getFactory();
CtClass<? extends Throwable> exceptionClass = factory.createClass( qualifiedName: "ow2con.PrivateAPIException");
CtConstructorCall<? extends Throwable> exceptionInstance = factory.createConstructorCall(exceptionClass.getReference());

for (CtMethod method : methodList) {
    CtBlock methodBody = method.getBody();
    List<CtComment> bodyComments = new ArrayList<>();

    ArrayList<CtStatement> ctStatements = new ArrayList<>(methodBody.getStatements());

    for (CtStatement ctStatement : ctStatements) {
        String statement = ctStatement.toString();
        CtComment statementAsComment = factory.createInlineComment(statement);
        bodyComments.add(statementAsComment);
        methodBody.removeStatement(ctStatement);
    }

    CtThrow throwMyException = factory.createThrow();
    CtConstructorCall<? extends Throwable> constructorCall = exceptionInstance.clone();
    throwMyException.setThrownExpression(constructorCall);
    methodBody.addStatement(throwMyException);
```

Transform model: hands on!

Goal: Add a final explanatory comment.

```
Factory factory = launcher.getFactory();
CtClass<? extends Throwable> exceptionClass = factory.createClass( qualifiedName: "ow2con.PrivateAPIException");
CtConstructorCall<? extends Throwable> exceptionInstance = factory.createConstructorCall(exceptionClass.getReference());

for (CtMethod method : methodList) {
    CtBlock methodBody = method.getBody();
    List<CtComment> bodyComments = new ArrayList<>();

    ArrayList<CtStatement> ctStatements = new ArrayList<>(methodBody.getStatements());

    for (CtStatement ctStatement : ctStatements) {
        String statement = ctStatement.toString();
        CtComment statementAsComment = factory.createInlineComment(statement);
        bodyComments.add(statementAsComment);
        methodBody.removeStatement(ctStatement);
    }

    CtThrow throwMyException = factory.createThrow();
    CtConstructorCall<? extends Throwable> constructorCall = exceptionInstance.clone();
    throwMyException.setThrownExpression(constructorCall);
    methodBody.addStatement(throwMyException);

    bodyComments.add(
            factory.createInlineComment(
            content: "FIXME: The private API type should never be return in a public API."
            )
    );

    for (CtComment bodyComment : bodyComments) {
        throwMyException.addComment(bodyComment);
    }
}
```

Let's output the model

```
launcher.prettyprint();
```

Output the model: scenario

What about the imports and comments?

```
package ow2con.publicapi;


public class MyPublicType {
    private ow2con.privateapi.PrivateType privateType;

    public MyPublicType() {
        this.privateType = new ow2con.privateapi.PrivateType();
    }

    public int getANumber() {
        return privateType.getANumber();
    }

    public ow2con.privateapi.PrivateType getPrivateType() {
        throw new ow2con.PrivateAPIException();
    }

    public ow2con.publicapi.subpack.TypePublic getTypePublic() {
        return new ow2con.publicapi.subpack.TypePublic();
```

Manage imports and comments

```
Environment environment = launcher.getEnvironment();
environment.setCommentEnabled(true);
environment.setAutoImports(true);
launcher.prettyprint();
```

Output the model with imports and comments

```java
package ow2con.publicapi;


import ow2con.PrivateAPIException;
import ow2con.privateapi.PrivateType;
import ow2con.publicapi.subpack.TypePublic;


public class MyPublicType {
    private PrivateType privateType;

    public MyPublicType() {
        this.privateType = new PrivateType();
    }

    public int getANumber() {
        return privateType.getANumber();
    }

    public PrivateType getPrivateType() {
        // return privateType
        // FIXME: The private API type should never be return in a public API.
        throw new PrivateAPIException();
    }

    public TypePublic getTypePublic() {
        return new TypePublic();
    }
```

It looks tedious? Use Processors!

Processors are a way to avoid calling all those steps:

- A processor is created for a specific kind of node in Spoon Model
- A process method is called each time the node type is encountered in the model
- We can then check properties and/or transform the node or the model itself

Let's try it!

Processor for our scenario - Structure

```
public class APICheckingProcessor extends AbstractProcessor<CtMethod> {
    private static final String PACKAGE_PUBLIC_API = "ow2con.publicapi";
    private static final String PACKAGE_PRIVATE_API = "ow2con.privateapi";
    private static final String EXCEPTION_FQN = "ow2con.PrivateAPIException";

    @Override
    public boolean isToBeProcessed(CtMethod method) {...}

    @Override
    public void process(CtMethod method) {...}
}
```

Processor for our scenario - isToBeProcessed

Method contains code to **query the model**

```
@Override
public boolean isToBeProcessed(CtMethod method) {
    // get the CtClass the method is attached to
    CtClass parentClass = method.getParent(CtClass.class);

    // check that the class belongs to the public API
    if (parentClass.getQualifiedName().contains(PACKAGE_PUBLIC_API)) {
        // check that the method is public
        if (method.isPublic()) {
            // check that the return type belongs to the private API
            CtTypeReference returnType = method.getType();
            return returnType.getQualifiedName().contains(PACKAGE_PRIVATE_API);
        }
    }
    return false;
}
```

Processor for our scenario - process

Method contains code to **transform the model**

```
@Override
public void process(CtMethod method) {
    final Factory factory = method.getFactory();

    CtBlock methodBody = method.getBody();

    List<CtComment> bodyComments = new ArrayList<>();

    ArrayList<CtStatement> ctStatements = new ArrayList<>(methodBody.getStatements());
    for (CtStatement ctStatement : ctStatements) {
        String statement = ctStatement.toString();
        bodyComments.add(factory.createInlineComment(statement));
        methodBody.removeStatement(ctStatement);
    }

    CtClass<? extends Throwable> myExceptionClass = factory.Class().get(EXCEPTION_FQN);
    CtConstructorCall<? extends Throwable> myNewException = factory.createConstructorCall(myExceptionClass.getReference());

    CtThrow throwMyException = factory.createThrow();
    throwMyException.setThrownExpression(myNewException);
    methodBody.addStatement(throwMyException);

    bodyComments.add(factory.createInlineComment( content: "FIXME: The private API type should never be return in a public API."));

    for (CtComment bodyComment : bodyComments) {
        throwMyException.addComment(bodyComment);
    }
}
```

Processor usage from Java API

```
public static void main(String[] args) {
    String projectPath = ".";

    MavenLauncher launcher = new MavenLauncher(projectPath, MavenLauncher.SOURCE_TYPE.APP_SOURCE);

    Environment environment = launcher.getEnvironment();
    environment.setCommentEnabled(true);
    environment.setAutoImports(true);

    launcher.addProcessor(new APICheckingProcessor());
    launcher.run();
}
```

Processor usage from CLI

```
Kwak-Spirals:ow2con2018-demo urli$ java -cp spoon-core-6.2.0-jar-with-dependencies.jar:demospoon/target/demo-spoo
n-1.0-SNAPSHOT.jar spoon.Launcher --with-imports --enable-comments -i demoproject/src/main/java -o /tmp/ow2demo -
p ow2con.processor.APICheckingProcessor
Kwak-Spirals:ow2con2018-demo urli$ cat /tmp/ow2demo/ow2con/publicapi/MyPublicType.java
package ow2con.publicapi;


import ow2con.PrivateAPIException;
import ow2con.privateapi.PrivateType;
import ow2con.publicapi.subpack.TypePublic;


public class MyPublicType {
    private PrivateType privateType;

    public MyPublicType() {
        this.privateType = new PrivateType();
    }

    public int getANumber() {
        return privateType.getANumber();
    }

    public PrivateType getPrivateType() {
        // return privateType
        // FIXME: The private API type should never be return in a public API.
        throw new PrivateAPIException();
    }

    public TypePublic getTypePublic() {
        return new TypePublic();
    }
}
```

Processor usage from Maven Plugin

```xml
<build>
    <plugins>
        <plugin>
            <groupId>fr.inria.gforge.spoon</groupId>
            <artifactId>spoon-maven-plugin</artifactId>
            <version>3.0</version>
            <dependencies>
                <dependency>
                    <groupId>ow2.demo.2018</groupId>
                    <artifactId>demo-spoon</artifactId>
                    <version>1.0-SNAPSHOT</version>
                </dependency>
            </dependencies>
            <executions>
                <execution>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>generate</goal>
                    </goals>
                </execution>
            </executions>
            <configuration>
                <enableComments>true</enableComments>
                <withImports>true</withImports>
                <processors>
                    <processor>
                        ow2con.processor.APICheckingProcessor
                    </processor>
                </processors>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Conclusion

Spoon is a multi-tool for Java Projects

This presentation was only about **a basic usage** of Spoon.

Projects are using Spoon for code analysis, program repair, code transpiling, or test amplification.

Use it in your own projects: for architecture checking, code refactoring, test enforcing, ...

New features are incoming to help you there!

We have an incredible community! Come help us improving Spoon API and get ready for Java 11 ;-)