OPENVALUE

BETTER SOFTWARE, FASTER.

OPENREWRITE WORKSHOP

Automated refactoring made easy

PREREQUISITES FOR THIS WORKSHOP

- the workshop repository
- An IDE, preferably IntelliJ IDEA
- Java 11, 17 and 21
  - sdkman on mac or linux
  - azul zulu or any other openjdk distribution on windows

sdkman

https://sdkman.io/

java

https://www.azul.com/downloads?package=jdk#zulu

workshop repository

https://github.com/OpenValue-D/Openrewrite_Workshop

WHAT'S YOUR NAME AGAIN?

Sebastian Konieczek

- working as software engineer since 17 years
- talking Java and Kotlin
- likes giving workshops and talks
- occasionally wakeboarding

- 19 years of experience
- likes to talk about observability
- DevEx as a hobby
- trainer for cloud infrastructures

Sascha Selzer

WHAT ARE YOUR EXPECTATIONS?

WHAT YOU CAN EXPECT

- what is open rewrite and how does it work
- how can I integrate openrewrite into my project
- how do I configure openrewrite
- how can I create custom openrewrite refactorings
- how do I test my custom openrewrite refactorings

WHAT IS OPENREWRITE?

A SHORT INTRODUCTION

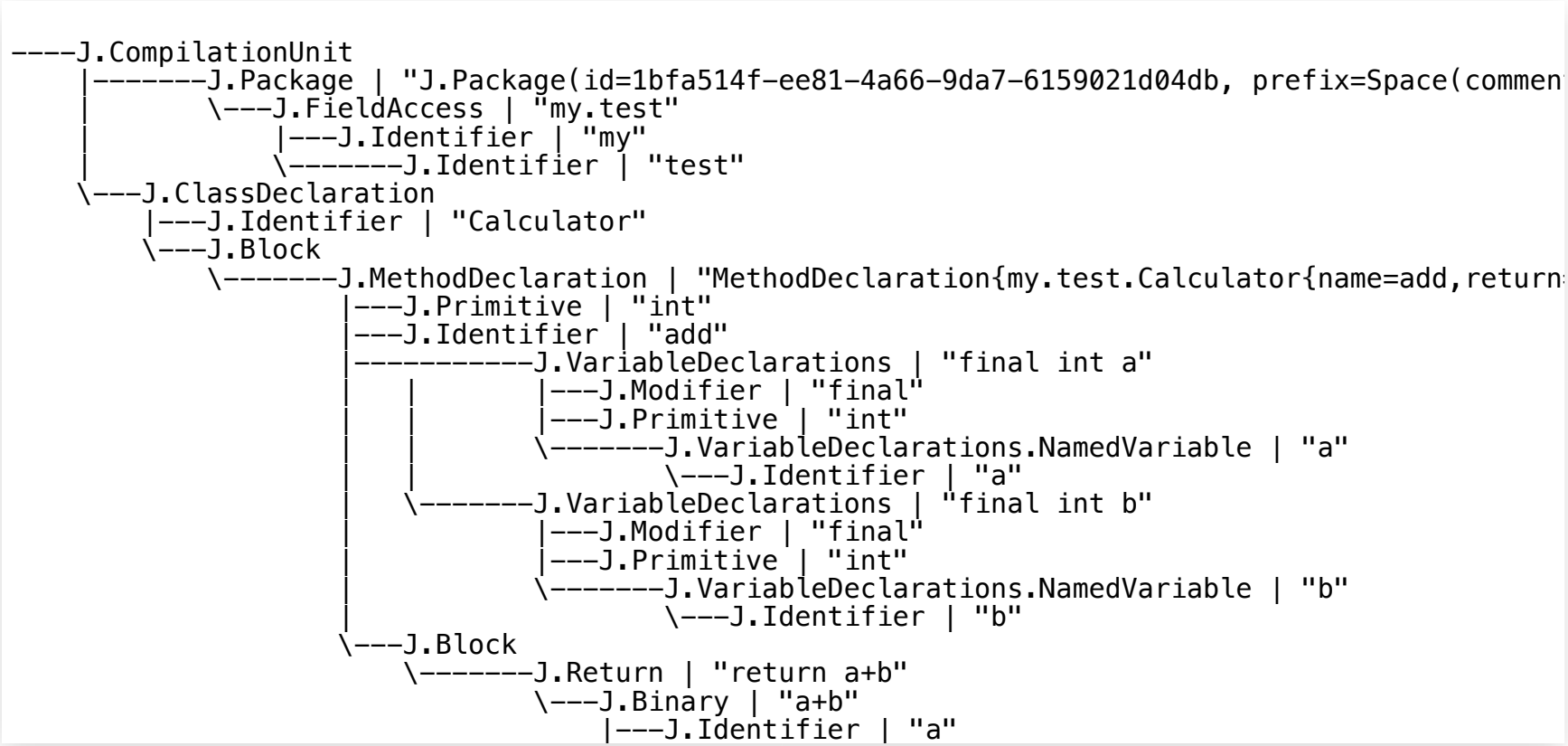
- created and maintained by moderne
- framework and ecosystem for automated code refactoring at scale
- prepackaged, open-source refactoring recipes*
- moderne platform/cli
  - free tier for open source projects

HOW DOES OPENREWRITE WORK?

- builds lossless semantic tree - LST
  - java
  - yml
  - xml
  - json
  - ...
- iterates recursively over the LST applying transformations to the LST
- stops when no more changes are applied to the LST
- writes transformed LST back to the source code files

```java
package my.test;

class Calculator
{
   int add(final int a, final int b)
   {
      return a+b;
   }
}
```


----J.CompilationUnit
|-------J.Package | "J.Package(id=1bfa514f-ee81-4a66-9da7-6159021d04db, prefix=Space(commen
| \---J.FieldAccess | "my.test"
| |---J.Identifier | "my"
| \-------J.Identifier | "test"
\---J.ClassDeclaration
|---J.Identifier | "Calculator"
\---J.Block
\-------J.MethodDeclaration | "MethodDeclaration{my.test.Calculator{name=add,return
|---J.Primitive | "int"
|---J.Identifier | "add"
|-----------J.VariableDeclarations | "final int a"
| | |---J.Modifier | "final"
| | |---J.Primitive | "int"
| | \-------J.VariableDeclarations.NamedVariable | "a"
| | \---J.Identifier | "a"
| \-------J.VariableDeclarations | "final int b"
| |---J.Modifier | "final"
| |---J.Primitive | "int"
| \-------J.VariableDeclarations.NamedVariable | "b"
| \---J.Identifier | "b"
\---J.Block
\-------J.Return | "return a+b"
\---J.Binary | "a+b"
|---J.Identifier | "a"

WHAT IS A RECIPE?

- a set of instructions on how and when to transform the LST

HOW TO USE A RECIPE

- build tool plugin
  - gradle
  - maven
- moderne cli tool
  - log in with Github or Bitbucket
  - free for public open source projects
  - for private projects a contract with moderne is required

```
./gradlew rewriteRun      # actually apply the configured recipes
./gradlew rewriteDryRun   # apply the configured recipes and create a patch file
./gradlew rewriteDiscover # get a list of available recipes provided by moderne
```

THE GRADLE PLUGIN

```
plugins {
    id("org.openrewrite.rewrite") version "6.11.2"
}

rewrite{
    activeRecipe("<recipe-name-1>")
    activeRecipe("<recipe-name-2>")
}

dependencies {
   rewrite("<rewrite-recipe-lib>")
}
```

THE REWRITE CONFIGURATION FILE

```yaml
---
type: specs.openrewrite.org/v1beta/recipe
name: de.my.recipe.definition
displayName: My Recipe definition
recipeList:
  - <recipe-name-1>
  - <recipe-name-2>
  - <recipe-name-3>:
      paramOne: <value-one>
      paramTwo: <value-one>
```

THE REWRITE CONFIGURATION FILE

- In the root of the project
- rewrite.yml
- activate it with the own name (here de.my.recipe.definition)

REWRITEDISCOVER

Finding recipes fast

```
./gradlew rewriteDiscover
```

ASSIGNMENT: 01_INTRO

ASSIGNMENT:
02_SPRING_BOOT_UPGRADE

DO I NEED TO CONFIGURE ALL THE RECIPES SEPARATELY FOR ALL MY PROJECTS?

Of course not!

CREATING AN OPENREWRITE RECIPE LIBRARY

HOW TO PACKAGE AN OWN RECIPE LIBRARY

- A declarative library uses the same rewrite.yml as configuring a recipe
- To use a recipe put the rewrite.yml in the root
- To package it put it in "src/main/resources/META-INF/rewrite"
- a recipe library is packaged as a normal java libary

ASSIGNMENT:
03_WRITE_DECLARATIVE_RECIPE

WHAT IF THERE IS NO RECIPE AVAILABLE FOR MY REFACTORING?

- utilize openrewrite powerful java api
  - requires deep understanding of the LST
  - a lot of code may be necessary even for seemingly small refactorings
- alternative: refaster templates

  what is this?

A SMALL EXCURSUS TO GOOGLE REFASTER TEMPLATES

- is part of the google Error Prone tool
- may be used for simple refactorings like
  - migrate uses of method A to method B
  - migrate use of method A where the arguments are of some particular type to method B
  - migrate a particular fluent sequence of method invocations to some other pattern
  - migrate a sequence of consecutive statements to an alternative

Source: Error Prone documentation

```java
import com.google.errorprone.refaster.annotation.AfterTemplate;
import com.google.errorprone.refaster.annotation.AlsoNegation;
import com.google.errorprone.refaster.annotation.BeforeTemplate;

public class StringIsEmpty {
  @BeforeTemplate
  boolean equalsEmptyString(String string) {
    return string.equals("");
  }

  @BeforeTemplate
  boolean lengthEquals0(String string) {
    return string.length() == 0;
  }

```

BEFORE TEMPLATE

- describes the code pattern that should be replaced
- the parameter(s) represent any expression of the specified type
  - the string parameter in the example stands in for any expression of type String

```
boolean equalsEmptyString(String string) {  // string => every expression of type strin
    return string.equals("");
}
```

- can contain multiple lines to be replaced
- for more advanced examples visit refaster docs

AFTER TEMPLATE

- describes the desired pattern
- has the same arguments as the before pattern
- can contain multiple lines
- for more advanced examples visit refaster docs

ALSO NEGATION

- used to signal that the rule can also match the logical negation of the @BeforeTemplate
- for more advanced examples visit refaster docs

DOES OPENREWRITE USE ERROR PRONE?

No!

A recipe is generated through an annotation processor by openrewrite

The final class ends with "Recipe" or "Recipes"

REQUIRED DEPENDENCIES FOR REFASTER

```
annotationProcessor("org.openrewrite:rewrite-templating:latest.release")
implementation("org.openrewrite:rewrite-templating")

compileOnly("com.google.errorprone:error_prone_core:2.26.1") {
   exclude("com.google.auto.service", "auto-service-annotations")
}
```

ASSIGNMENT:
04_CUSTOM_REFASTER_RECIPE

TESTING

TESTING

- support for writing unit tests
- supports different SourceSpecs (java, yaml, xml, etc...)
- can fine tune the test execution environment before applying a recipe
- tests can use newer version of Java than the recipe (e.g. to make use of multiline strings)

DEPENDENCIES

```
dependencies {
    implementation(platform("org.openrewrite.recipe:rewrite-recipe-bom:2.8.1"))

    testImplementation("org.openrewrite:rewrite-java")
    testImplementation("org.openrewrite:rewrite-maven")
    testImplementation("org.openrewrite.recipe:rewrite-java-dependencies")
    testImplementation("org.openrewrite:rewrite-java-21")
    testImplementation("org.openrewrite:rewrite-test")
    testImplementation("org.junit.jupiter:junit-jupiter-api:latest.release")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:latest.release")
}
```

TEST PREPARATION

```java
import org.openrewrite.test.RecipeSpec;
import org.openrewrite.test.RecipeTest;

class MyRecipeTest implements RewriteTest {

   @Override
   public void defaults(RecipeSpec spec) {
      //for Java written recipes
      spec.recipe(new MyRecipe());
      // for declarative recipes
      spec.recipe(RecipeTest.fromRuntimeClasspath("de.my.package.MyRecipe"));
   }
}
```

FIRST TEST

```
1  @Test
2  void myFirstTest() {
3     rewriteRun(
4        java(
5           """
6           class A {}
7           """,
8           """
9           class A {}
10          """
11       )
12    );
13 }
```

REWRITERUN

- expects a list of SourceSpecs (here one java)
- SourceSpec content must be valid as it gets parsed like the real source code
- SourceSpec describes a before and after state when the recipe was executed
- second String can be omitted if no change is expected
- RecipeSpec can be changed for a test case (e.g. adding a library to the classpath)

ADAPT RECIPESPEC

```
1  @Test
2  void otherTest() {
3     rewriteRun(
4        spec -> spec.parser(JavaParser.fromJavaVersion()
5                    .classpath("junit-4.13")),
6        java(
7           """
8           import org.junit.Test;
9           public class A {}
10          """
11       )
12    );
13 }
```

ASSIGNMENT: 05_TEST_RECIPE

GETTING OUR HANDS DIRTY!

Writing our own recipe with the open rewrite api

RECIPE CLASS

```java
import lombok.EqualsAndHashCode;
import lombok.Value;
import org.openrewrite.Recipe;

@EqualsAndHashCode(callSuper = false)
@Value
public class MyRecipe extends Recipe {

  @Option(displayName = "An config argument for my recipe",
          description = "Recipes can be configured like the RenamePackage recipe")
  String myArgument;

  @Override
  public String getDisplayName() {
    return "This is my recipe";
```

RECIPE

- Defines the configuration of the recipe
- Can have optional arguments
- Defines information that will be displayed when doing rewriteDiscover
- Defines a visitor to traverse the code and make changes
- Has to be serializable

DIFFERENT VISITORS

- TreeVisitor (abstract base class)
- JavaIsoVisitor
- MavenVisitor
- PlainTextVisitor
- YamlIsoVisitor
- XmlIsoVisitor
- ...

JAVAVISITOR

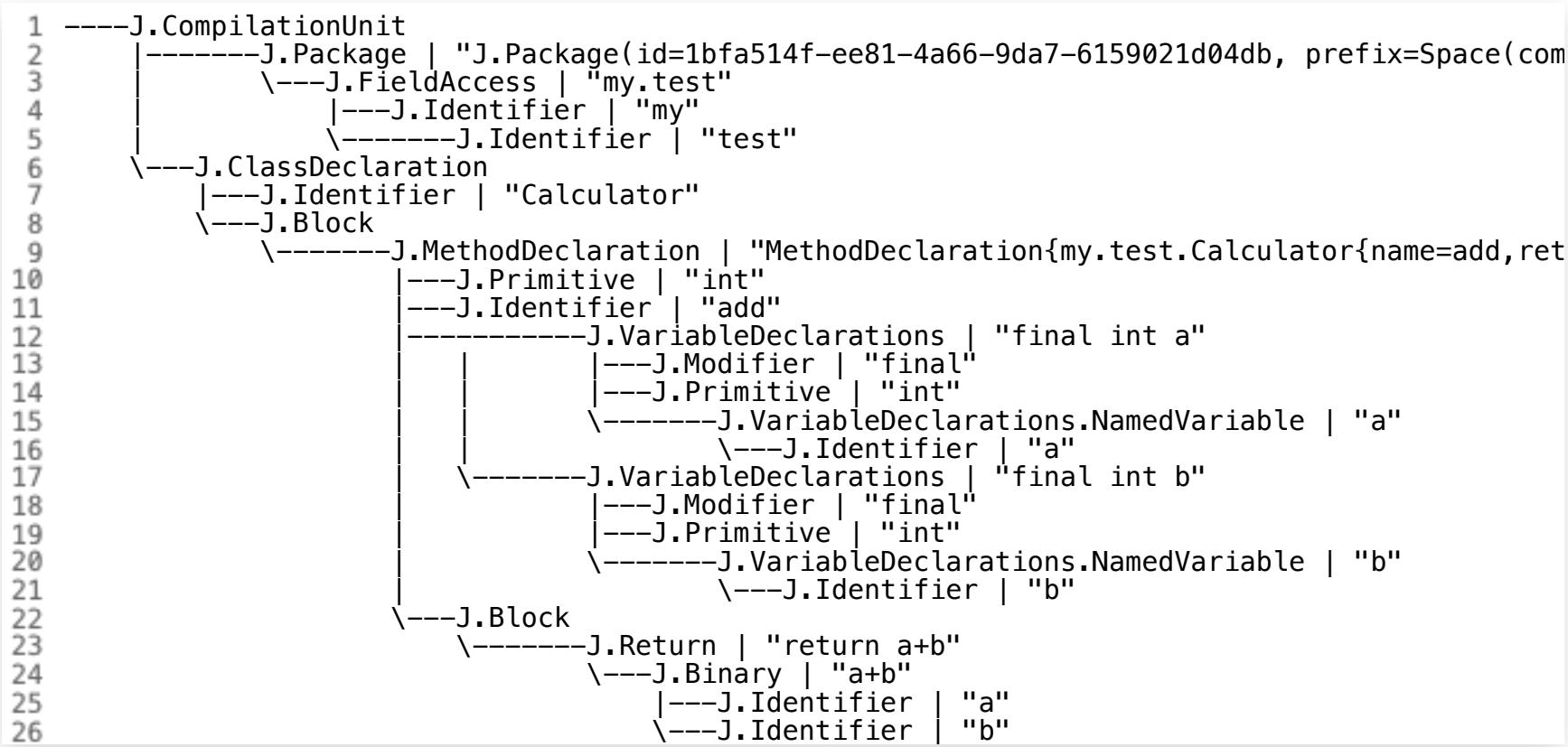
```
1  class JavaVisitor<P> extends TreeVisitor<J, P> {
2    J visitStatement(Statement statement) {...}
3    J visitTypeName(NameTree name) {...}
4    J visitAnnotatedType(J.AnnotatedType annotatedType)  {...}
5    J visitAnnotation(J.Annotation annotation) {...}
6    J visitArrayType(J.ArrayType arrayType) {...}
7    J visitAssert(J.Assert azzert) {...}
8    J visitAssignment(J.Assignment assign) {...}
9    J visitAssignmentOperation(J.AssignmentOperation assignOp) {...}
10   J visitBinary(J.Binary binary) {...}
11   Cursor getCursor() {...}
12   ...
13 }
```

VISITOR PATTERN

- Visitor will be called when ever traversing a matching block in the LST
- visit methods run independently and will be called by OpenRewrite
- Visit methods available on all level of the LST (from CompilationUnit to single statement)
- Visit methods return for isomorphic Visitors the same type of LST element as visited

DEBUGGING

```
System.out.println(TreeVisitingPrinter.printTree(getCursor()));
```


----J.CompilationUnit
|-------J.Package | "J.Package(id=1bfa514f-ee81-4a66-9da7-6159021d04db, prefix=Space(com
| \---J.FieldAccess | "my.test"
| |---J.Identifier | "my"
| \-------J.Identifier | "test"
\---J.ClassDeclaration
|---J.Identifier | "Calculator"
\---J.Block
\-------J.MethodDeclaration | "MethodDeclaration{my.test.Calculator{name=add,ret
|---J.Primitive | "int"
|---J.Identifier | "add"
|-----------J.VariableDeclarations | "final int a"
| | |---J.Modifier | "final"
| | |---J.Primitive | "int"
| | \-------J.VariableDeclarations.NamedVariable | "a"
| | \---J.Identifier | "a"
| \-------J.VariableDeclarations | "final int b"
| |---J.Modifier | "final"
| |---J.Primitive | "int"
| \-------J.VariableDeclarations.NamedVariable | "b"
| \---J.Identifier | "b"
\---J.Block
\-------J.Return | "return a+b"
\---J.Binary | "a+b"
|---J.Identifier | "a"
\---J.Identifier | "b"

BUT HOW TO START?

Correct! The openrewrite recipe starter!

```java
package com.yourorg;

import lombok.EqualsAndHashCode;
import lombok.Value;
import org.openrewrite.ExecutionContext;
import org.openrewrite.Preconditions;
import org.openrewrite.Recipe;
import org.openrewrite.TreeVisitor;
import org.openrewrite.java.JavaIsoVisitor;
import org.openrewrite.java.JavaParser;
import org.openrewrite.java.JavaTemplate;
import org.openrewrite.java.MethodMatcher;
import org.openrewrite.java.search.UsesType;
import org.openrewrite.java.tree.Expression;
import org.openrewrite.java.tree.J;
```

Source: https://github.com/moderneinc/rewrite-recipe-starter

THE CURSOR

- keeps track of a visitor's position within the LST
- used to access parent or sibling LSTs
- discarded if visiting is complete
- contains map to store and share data between visit methods
- organized as stack

CURSOR EXAMPLES

```
getCursor().putMessageOnFirstEnclosing(J.ClassDeclaration.class, "FOUND_METHOD", methodDeclarat
...
getCursor().pollMessage("FOUND_METHOD"); // removes message from cursor
...
getCursor().getMessage("FOUND_METHOD");  // leaves message on cursor
...
getCursor().getNearestMessage("FOUND_METHOD");
```

```
getCursor().getParentOrThrow()
```

ASSIGNMENT:
07_APPENDIX_CUSTOM_RECIPE_OPENREWRITE_API

JAVA TEMPLATES

```java
public class ChangeMethodInvocation extends JavaIsoVisitor<ExecutionContext> {
   private final JavaTemplate template =
       JavaTemplate.builder("withString(#{any(java.lang.String)}).length()")   // Code Snipp
           .javaParser(
               JavaParser.fromJavaVersion()                                    // Parser
                   .classpath("example-utils"))                                // Classpath
           .staticImports("org.example.StringUtils.withString")                // Additional
           .build();
}
```

JAVA TEMPLATES

- Generates code (LST elements) based on a string template
- String must be syntical correct
- ensures correct formatting
- can be applied on elements in the visit methods
- able to reference symbols
- can add needed imports for code snippet

ADD IMPORTS

```
JavaTemplate.builder("new SecureRandom()")
     .imports("java.security.SecureRandom")
     .build();
```

APPLY TO LST

```
1  public class ChangeMethodInvocation extends JavaIsoVisitor<ExecutionContext> {
2      private final JavaTemplate template = JavaTemplate.builder("withString(#{any(java.lang.S
3          .javaParser(JavaParser.fromJavaVersion().classpath("example-utils"))
4          .staticImports("org.example.StringUtils.withString")
5          .build();
6
7      public J.MethodInvocation visitMethodInvocation(J.MethodInvocation method, ExecutionCont
8          J.MethodInvocation m = super.visitMethodInvocation(method, p);
9          if (m.getSimpleName().equals("countLetters")) {
10
11             m = template.apply(getCursor(), m.getCoordinates().replace(), m.getArguments().g
12             maybeAddImport("org.example.StringUtils", "withString");
13         }
14         return m;
15     }
```

WRAP UP AND OUTLOOK

when should I use openrewrite

- medium to large code base
- refactoring affects numerous files
- framework and library updates

how should I use openrewrite

- prefer existing recipes
- prefer declarative recipes
- try to achieve what you need with refaster template recipes
- in other words: try to avoid writing your own recipes in java

WRAP UP AND OUTLOOK

you need a custom imperative recipe?

- use the starter
- read the docs
- use the moderne you tube channel
- ask for help in the openrewrite slack channel

THANK YOU!