Instructor Notes:

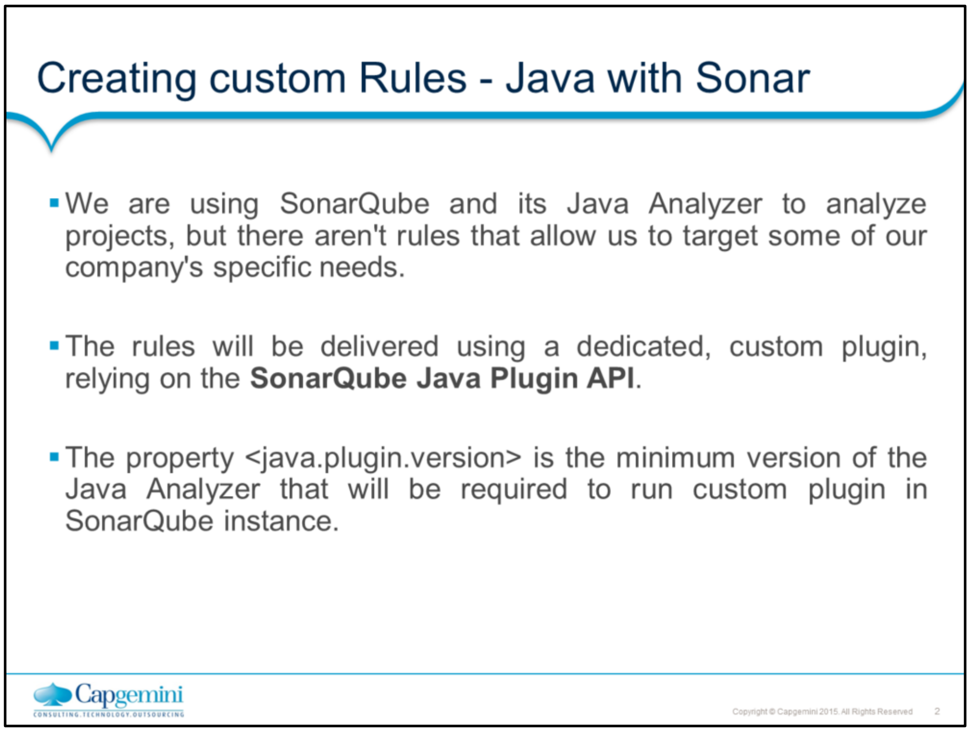
Add instructor notes here.

DevOps

Appendices

Instructor Notes:

Add instructor notes here.

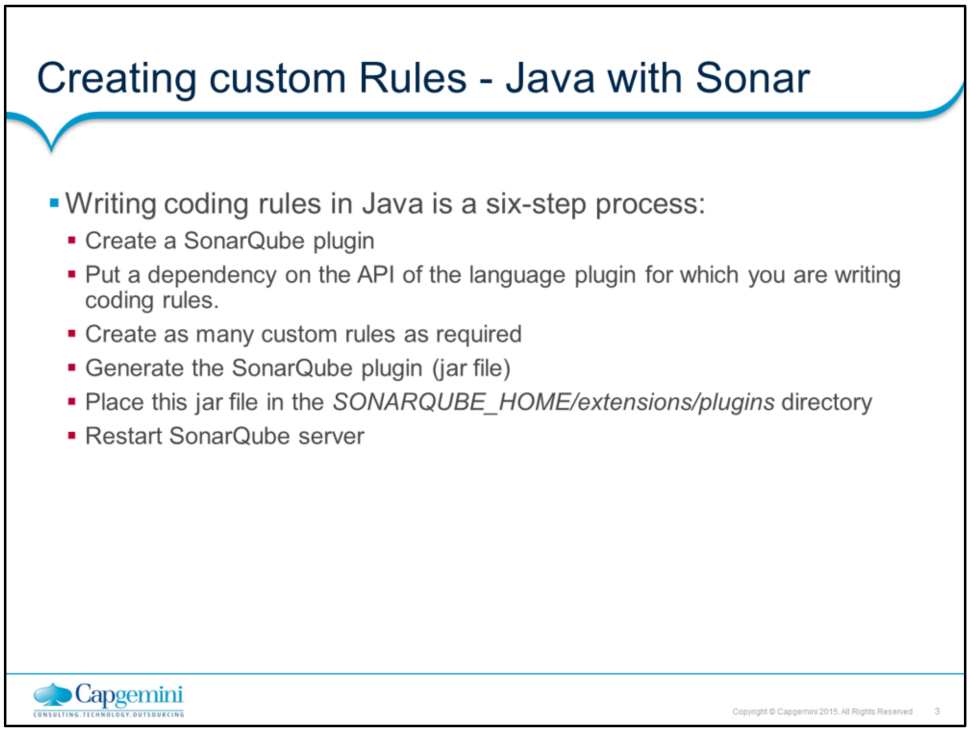


Creating custom Rules - Java with Sonar

- We are using SonarQube and its Java Analyzer to analyze projects, but there aren't rules that allow us to target some of our company's specific needs.
- The rules will be delivered using a dedicated, custom plugin, relying on the **SonarQube Java Plugin API**.
- The property <java.plugin.version> is the minimum version of the Java Analyzer that will be required to run custom plugin in SonarQube instance.

Instructor Notes:

Add instructor notes here.



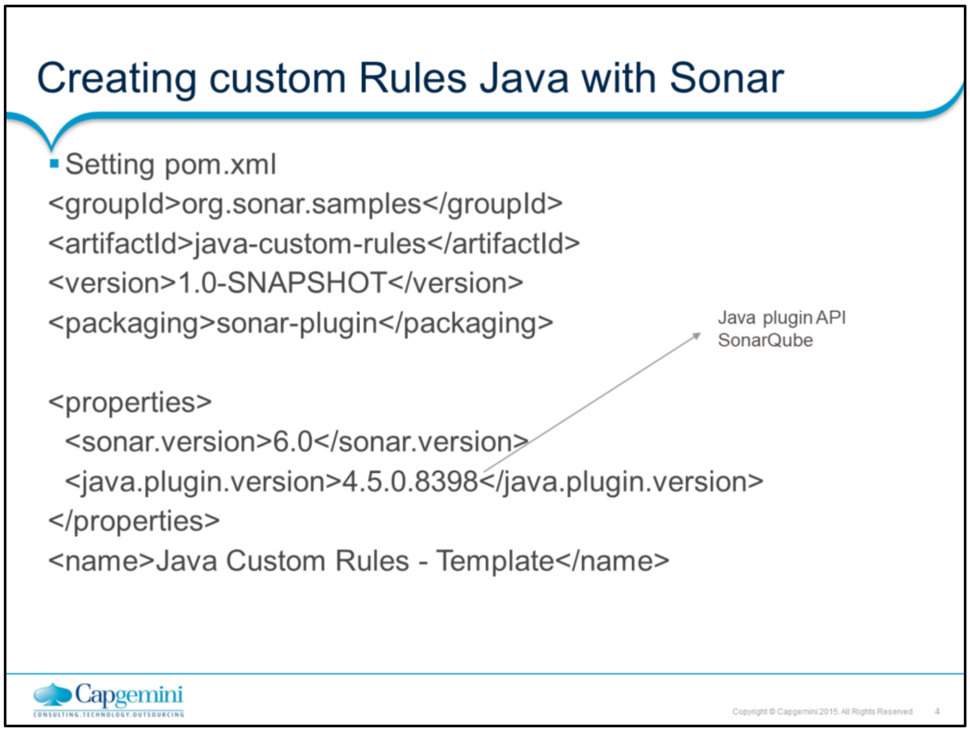
Creating custom Rules - Java with Sonar

- Writing coding rules in Java is a six-step process:
  - Create a SonarQube plugin
  - Put a dependency on the API of the language plugin for which you are writing coding rules.
  - Create as many custom rules as required
  - Generate the SonarQube plugin (jar file)
  - Place this jar file in the *SONARQUBE_HOME/extensions/plugins* directory
  - Restart SonarQube server

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Instructor Notes:

Add instructor notes here.



Creating custom Rules Java with Sonar

- Setting pom.xml

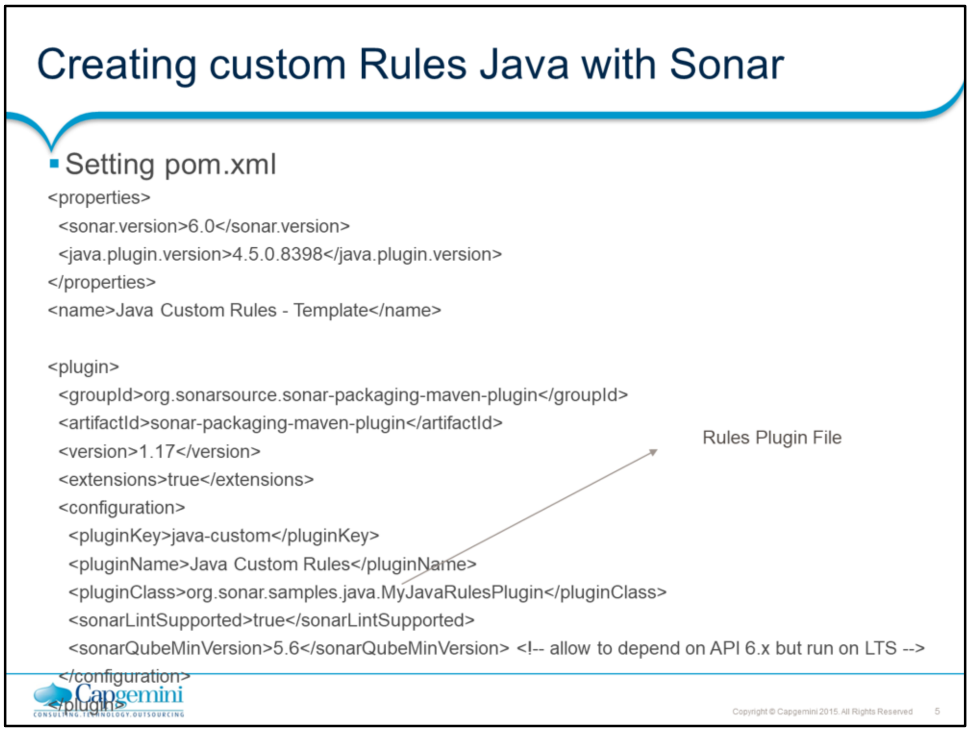
```
<groupId>org.sonar.samples</groupId>
<artifactId>java-custom-rules</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>sonar-plugin</packaging>

<properties>
  <sonar.version>6.0</sonar.version>
  <java.plugin.version>4.5.0.8398</java.plugin.version>
</properties>
<name>Java Custom Rules - Template</name>
```

Java plugin API SonarQube

Instructor Notes:

Add instructor notes here.



Creating custom Rules Java with Sonar

- Setting pom.xml

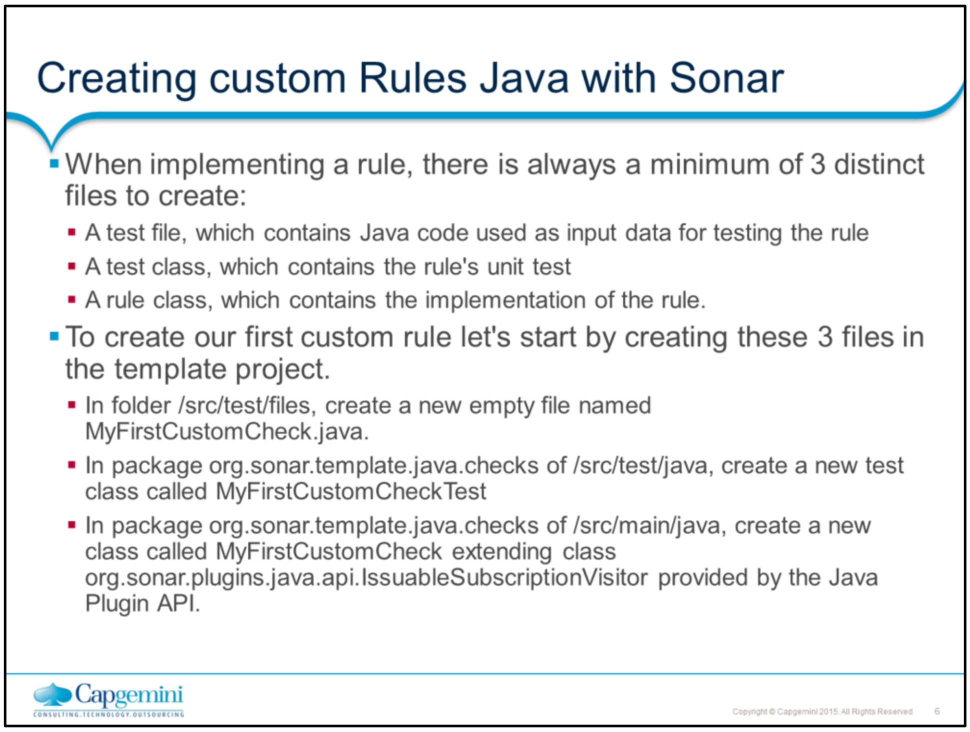
```
<properties>
  <sonar.version>6.0</sonar.version>
  <java.plugin.version>4.5.0.8398</java.plugin.version>
</properties>
<name>Java Custom Rules - Template</name>

<plugin>
  <groupId>org.sonarsource.sonar-packaging-maven-plugin</groupId>
  <artifactId>sonar-packaging-maven-plugin</artifactId>
  <version>1.17</version>
  <extensions>true</extensions>
  <configuration>
    <pluginKey>java-custom</pluginKey>
    <pluginName>Java Custom Rules</pluginName>
    <pluginClass>org.sonar.samples.java.MyJavaRulesPlugin</pluginClass>
    <sonarLintSupported>true</sonarLintSupported>
    <sonarQubeMinVersion>5.6</sonarQubeMinVersion> <!-- allow to depend on API 6.x but run on LTS -->
  </configuration>
</plugin>
```

Rules Plugin File

Instructor Notes:

Add instructor notes here.

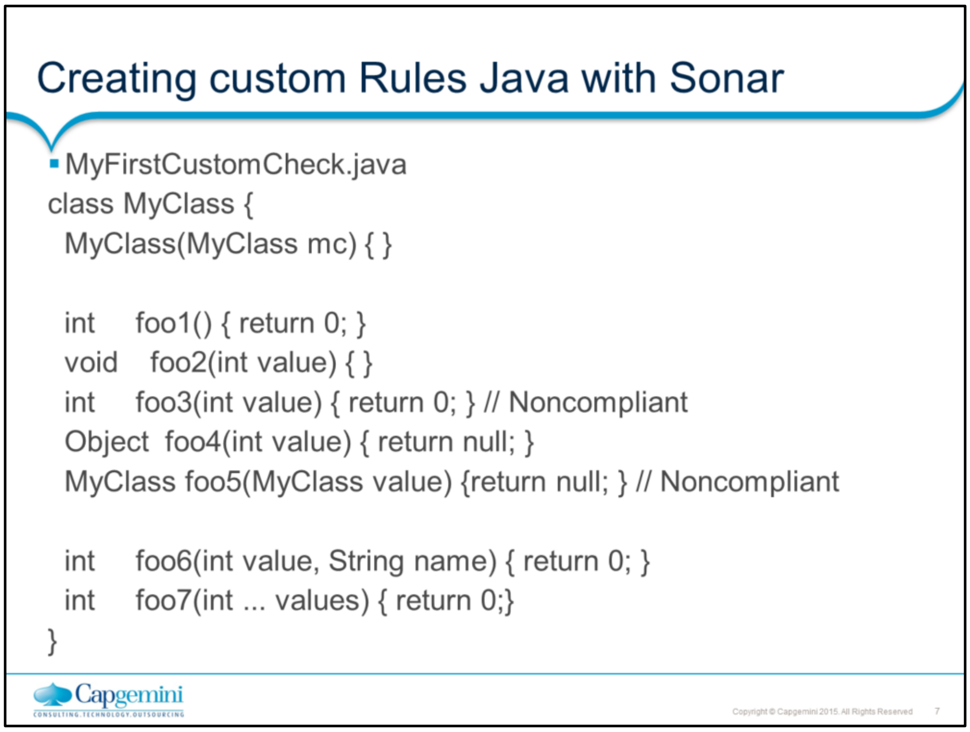


Creating custom Rules Java with Sonar

- When implementing a rule, there is always a minimum of 3 distinct files to create:
  - A test file, which contains Java code used as input data for testing the rule
  - A test class, which contains the rule's unit test
  - A rule class, which contains the implementation of the rule.
- To create our first custom rule let's start by creating these 3 files in the template project.
  - In folder /src/test/files, create a new empty file named MyFirstCustomCheck.java.
  - In package org.sonar.template.java.checks of /src/test/java, create a new test class called MyFirstCustomCheckTest
  - In package org.sonar.template.java.checks of /src/main/java, create a new class called MyFirstCustomCheck extending class org.sonar.plugins.java.api.IssuableSubscriptionVisitor provided by the Java Plugin API.

Instructor Notes:

Add instructor notes here.



Creating custom Rules Java with Sonar

- MyFirstCustomCheck.java

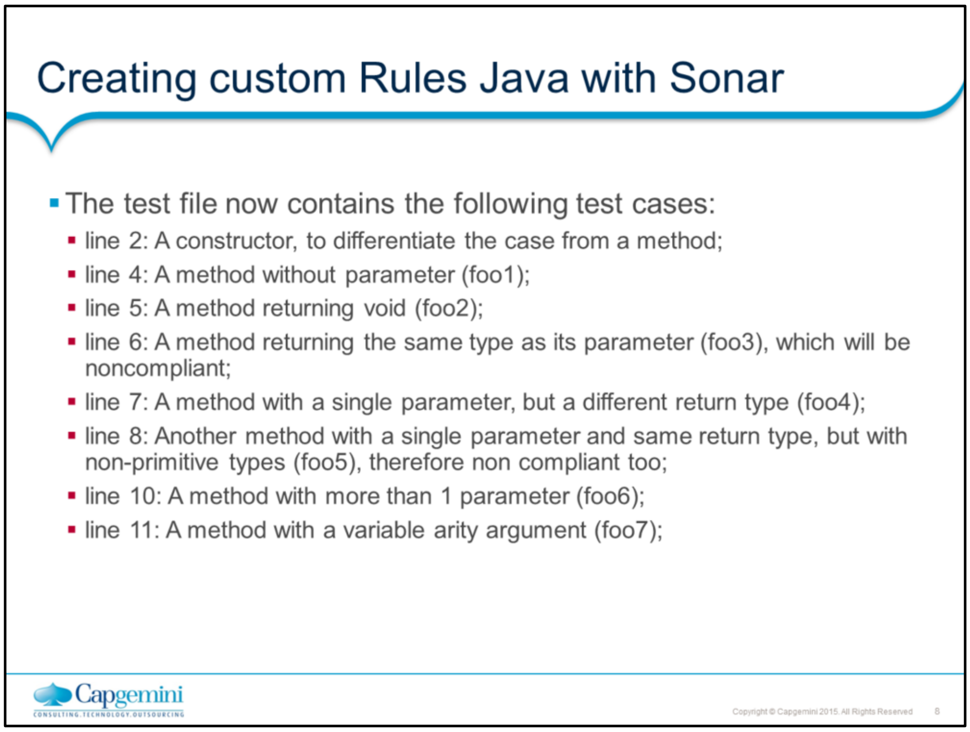
```
class MyClass {
  MyClass(MyClass mc) { }

  int     foo1() { return 0; }
  void    foo2(int value) { }
  int     foo3(int value) { return 0; } // Noncompliant
  Object  foo4(int value) { return null; }
  MyClass foo5(MyClass value) {return null; } // Noncompliant

  int     foo6(int value, String name) { return 0; }
  int     foo7(int ... values) { return 0;}
}
```

Instructor Notes:

Add instructor notes here.



Creating custom Rules Java with Sonar

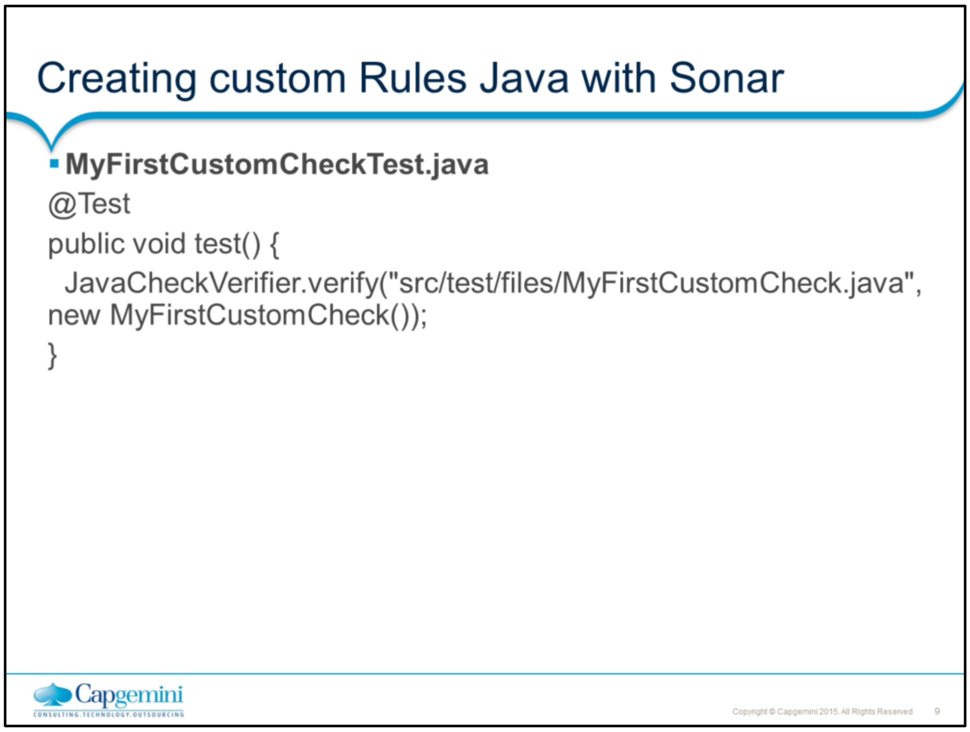
- The test file now contains the following test cases:
  - line 2: A constructor, to differentiate the case from a method;
  - line 4: A method without parameter (foo1);
  - line 5: A method returning void (foo2);
  - line 6: A method returning the same type as its parameter (foo3), which will be noncompliant;
  - line 7: A method with a single parameter, but a different return type (foo4);
  - line 8: Another method with a single parameter and same return type, but with non-primitive types (foo5), therefore non compliant too;
  - line 10: A method with more than 1 parameter (foo6);
  - line 11: A method with a variable arity argument (foo7);

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

8

Instructor Notes:

Add instructor notes here.



Creating custom Rules Java with Sonar

- **MyFirstCustomCheckTest.java**

```
@Test
public void test() {
  JavaCheckVerifier.verify("src/test/files/MyFirstCustomCheck.java",
new MyFirstCustomCheck());
}
```

Instructor Notes:

Add instructor notes here.

Appendix B. Code Examples