Java Development - Tools

Venkatesh

WDC

August 16, 2018

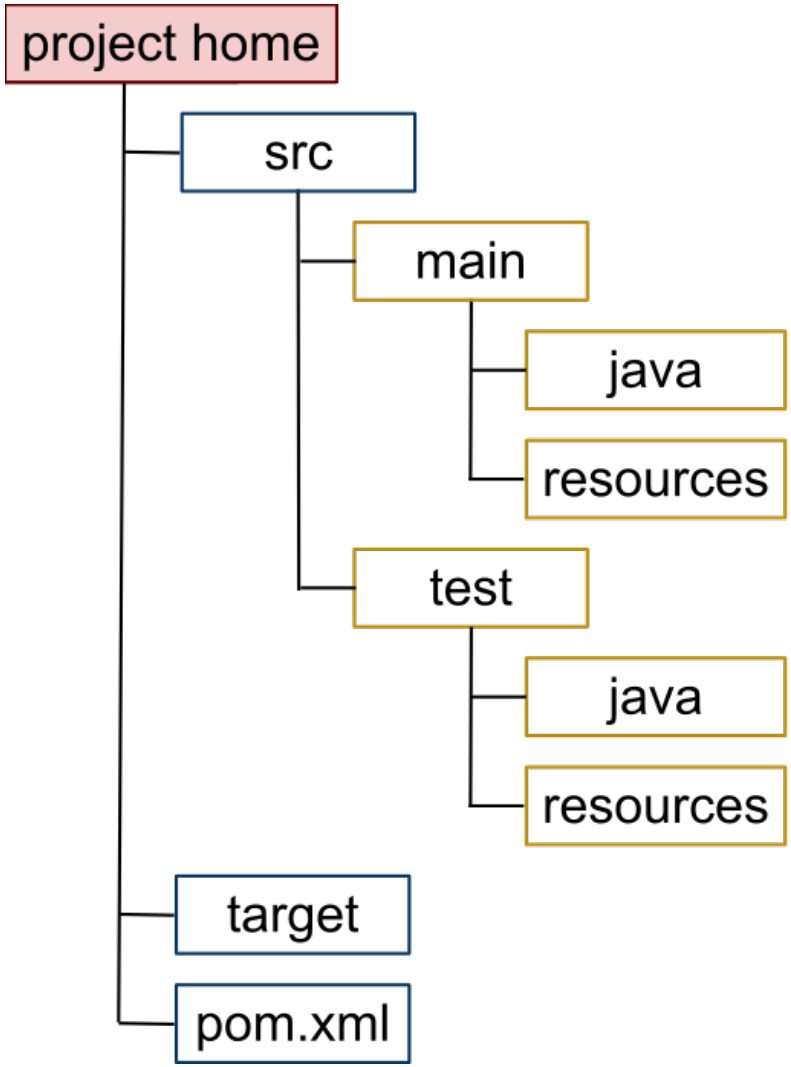
Outline

1. Maven - build automation tool
2. Checkstyle - coding style
3. SpotBugs - static analysis tool
4. JUnit 5 - Unit Testing

Maven

- An `XML` file describes the project being built, its dependencies
- Maven is plugin-excution framework
  - All work is done by plugins
- Dynamically downloads plug-ins from one or more repositories

Maven


project home
src
main
java
resources
test
java
resources
target
pom.xml

Maven Plugins

A plugin provides a set of goals

syntax for execution

```
mvn [plugin-name]:[goal-name]
```

Example

```
mvn compiler:compile
```

Build lifecycle

- validate
- generate-sources
- process-sources
- generate-resources
- process-resources
- compile
- process-test-sources
- process-test-resources
- test-compile
- test
- package
- install
- deploy

Example

CheckStyle

Checkstyle can examine

- Javadoc comments for classes, attributes and methods
- Naming conventions of attributes and methods
- Limit of the number of function parameters, line lengths
- The spaces between some characters
- ...

Maven Checkstyle plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>3.0.0</version>

  <dependencies>
    <dependency>
      <groupId>com.puppycrawl.tools</groupId>
      <artifactId>checkstyle</artifactId>
      <version>8.11</version>
    </dependency>
  </dependencies>
```

Maven Checkstyle plugin...

```
  <executions>
    <execution>
      <id>validate</id>
      <phase>validate</phase>
      <configuration>
        <configLocation>google_checks.xml</configLocation>
        <encoding>UTF-8</encoding>
        <consoleOutput>true</consoleOutput>
        <failsOnError>true</failsOnError>
        <failOnViolation>true</failOnViolation>
        <violationSeverity>warning</violationSeverity>
      </configuration>
      <goals>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

SpotBugs

- SpotBugs is the successor of FindBugs
- Static analysis tool to look for bugs in Java code
- It checks for more than 400 bug patterns
- `spotbugs-maven-plugin` can be added to pom.xml

Bug Descriptions

https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html

JUnit 5

- Unit testing framework
- *JUnit 5* requires *Java 8* (or higher) at runtime
- *JUnit 5* can be added to pom.xml

A first test case

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

Annotations

- `@Test`
- *@ParameterizedTest*
- *@RepeatedTest*
- `@BeforeEach`
- `@AfterEach`
- `@BeforeAll` : must be static methods
- `@AfterAll` : must be static methods
- `@Disabled`

Test Classes and Methods

Test method

method that is annotated with `@Test`, `@RepeatedTest`, `@ParameterizedTest`, `@TestFactory`, or `@TestTemplate`

Test class

any top level or static member class that contains at least one test method

Test Classes and Methods ...

Methods annotated with

- `@Test`
- `@TestTemplate`
- `@RepeatedTest`
- `@BeforeAll`
- `@AfterAll`
- `@BeforeEach` or
- `@AfterEach`

annotations *must not return* a value

Example

Assertions

- assertions are `static` methods in the `org.junit.jupiter.api.Assertions` class

Example

Repeated Tests

- Repeat a test a specified number of times simply by annotating a method with `@RepeatedTest`

Example

```
@RepeatedTest(10)
void repeatedTest() {
    // ...
}
```

Parameterized Tests

- Run a test multiple times with different arguments

Example

```
@ParameterizedTest
@ValueSource(strings = { "racecar", "radar", "121"})
void palindromesTest(String candidate) {
    assertTrue(isPalindrome(candidate));
}
```

Sources of Arguments

- `@ValueSource`
  - Used for providing a single parameter per parameterized test
- `@MethodSource`
  - factory method, must be static
  - must return a `Stream`, `Iterable`, `Iterator`, or array of arguments
- `@CsvSource`

@ValueSource

Following types of literal values are supported

- short
- byte
- int
- long
- float
- double
- char
- java.lang.String
- java.lang.Class

@ValueSource...

Example

```
@ParameterizedTest
@ValueSource(ints = { 1, 2, 3 })
void testWithValueSource(int argument) {
    assertTrue(argument > 0 && argument < 4);
}
```

@MethodSource

Example

```
@ParameterizedTest
@MethodSource("stringProvider")
void testWithSimpleMethodSource(String argument) {
    assertNotNull(argument);
}

static Stream<String> stringProvider() {
    return Stream.of("foo", "bar");
}
```

@CsvSource

- Allows you to express argument lists as *comma-separated* values

Example

```
@ParameterizedTest
@CsvSource({ "foo, 1", "bar, 2", "'baz, qux', 3" })
void testWithCsvSource(String first, int second) {
    assertNotNull(first);
    assertNotEquals(0, second);
}
```

References

- JUnit 5 User Guide
- https://junit.org/junit5/docs/5.0.0-M2/api/org/junit/jupiter/api/Assertions.html

Thank You