

Lecture 2 – Testing and Documentation

SWS121: Secure Programming

Jihyeok Park



2024 Spring

We learned basics of Scala programming in the last lecture.

- Basic Features
 - Basic Data Types
 - Variables
 - Methods
 - Recursion
- Algebraic Data Types (ADTs)
 - Product Types – Case Classes
 - Algebraic Data Types (ADTs) – Enumerations
 - Pattern Matching
 - Methods
- First-Class Functions
- Immutable Collections
 - Lists
 - Options and Pairs
 - Maps and Sets
 - For Comprehensions

1. Simple Build Tool (sbt) for Scala

- Example Project

- Project Structure

- Building a Project

- Running a Project

2. Scala Documentation

- scaladoc – Scala Documentation Tool

- Generating Documentation

- Writing Documentation

3. Scala Test Framework

- Why Software Testing?

- ScalaTest – Test Framework for Scala

- Running Tests

- Writing Tests

- Measuring Code Coverage

1. Simple Build Tool (sbt) for Scala

- Example Project

- Project Structure

- Building a Project

- Running a Project

2. Scala Documentation

- scaladoc – Scala Documentation Tool

- Generating Documentation

- Writing Documentation

3. Scala Test Framework

- Why Software Testing?

- ScalaTest – Test Framework for Scala

- Running Tests

- Writing Tests

- Measuring Code Coverage

In Scala, a library or a program is compiled using the Scala compiler, scalac, as documented in Scala 3 Book.¹

```
@main def hello: Unit = println("Hello, world!") /* hello.scala */
```

```
$ scalac hello.scala
```

```
$ ls -l
```

```
# hello$package$.class
```

```
# hello$package.class
```

```
# hello$package.tasty
```

```
# hello.class
```

```
# hello.scala
```

```
# hello.tasty
```

```
$ scala hello
```

```
# Hello, world!
```

How to handle **multiple files, dependencies, testing**, etc.?

¹<https://docs.scala-lang.org/scala3/book/taste-hello-world.html>

sbt

- [sbt](#) is a **simple build tool** for Scala and Java projects. It is similar to Maven or Ant, but it is designed for **Scala**.
- Rather than using `scalac` directly, [sbt](#) provides a more convenient way to **compile**, **run**, **test**, **document**, and **package** Scala programs.
- [sbt](#) supports a **domain-specific language (DSL)** called `build.sbt` **DSL** for defining the build process of a Scala project.

Here is a **simple example [sbt](#) project** that includes a simple **arithmetic expression** `Expr` and a tree `Tree` data structure:

<https://github.com/ku-plrg-classroom/scala-example>

You can clone the project using the following command:

```
$ git clone https://github.com/ku-plrg-classroom/scala-example.git
```

Please check you have JDK 8 or later and [sbt](#) installed on your system.

```
$ java -version
# java version "21.0.2" 2024-01-16 LTS

$ sbt --script-version
# 1.9.4
```

A typical [sbt](#) project has the following structure:

```
build.sbt           # build definition
project
  build.properties  # sbt version
  plugins.sbt       # sbt plugins
src/
  main/
    resources/      # resources
    scala/           # main Scala sources
  test/
    scala/           # test Scala sources
```


We can define the build process of the project in the build.sbt file:

```
ThisBuild / scalaVersion := "3.3.3"
ThisBuild / scalacOptions += Seq(...)
lazy val root = project
  .in(file("."))
  .settings(
    name := "scala-example",
    libraryDependencies += "org.scalatest" %% "scalatest" % "3.2.15" %
      Test,
    coverageEnabled := true,
    ...
  )
...
```

We can freely utilize Java (JVM-based) libraries in Scala projects:

```
libraryDependencies += "org.scalatest" %% "scalatest" % "3.2.15" % Test,
```

The project directory contains the following files:

```
project/  
  build.properties          # sbt version  
  plugins.sbt              # sbt plugins
```

The example project uses 1.9.9 version of [sbt](#):

```
sbt.version=1.9.9
```

and uses the following plugins:

```
addSbtPlugin("org.wartremover" % "sbt-wartremover" % "3.1.6")  
addSbtPlugin("org.scoverage" % "sbt-scoverage" % "2.0.11")
```

- wartremover is used to **block non-functional Scala features**.
- scoverage is used to measure **code coverage**.

The example project has **three main Scala files**:

- `App.scala` – Main Application
- `Expr.scala` – Arithmetic Expression
- `Tree.scala` – Tree Data Structure

and **two test Scala files**:

- `ExprSpec.scala` – Test Suite for Arithmetic Expression
- `TreeSuite.scala` – Test Suite for Tree Data Structure

You can build the project using the following command:

```
$ sbt compile
# [success] Total time: 0 s, completed ...
```

It is better to use the following command to start the [sbt](#) shell:

```
$ sbt
# [info] ...
sbt:scala-example> compile
# [info] ...
# [success] Total time: 0 s, completed ...
sbt:scala-example>
```

In general, you can **run the project** by using the `sbt run` command:

```
$ sbt run
# Hello, world!
```

In addition, you can **interactively explore the project** with the console (Scala REPL) by running the following command:

```
$ sbt console
```

Then, it shows the following prompt:

```
scala> import kuplrg.{ Expr, Tree }, import Expr.*

scala> val expr: Expr = Mul(Num(2), Add(Var("x"), Var("y")))
val expr: kuplrg.Expr = Mul(Num(2),Add(Var(x),Var(y)))

scala> expr.eval(Map("x" -> 3, "y" -> 5), 0)
val res1: Int = 16

scala>
```

1. Simple Build Tool (sbt) for Scala

Example Project

Project Structure

Building a Project

Running a Project

2. Scala Documentation

scaladoc – Scala Documentation Tool

Generating Documentation

Writing Documentation

3. Scala Test Framework

Why Software Testing?

ScalaTest – Test Framework for Scala

Running Tests

Writing Tests

Measuring Code Coverage

- **Documentation** is an essential part of **secure programming**.
- It helps to correctly **understand** the code, **maintain** the code, and **reuse** the code.
- It helps to **secure** the code by **preventing** security vulnerabilities because it helps to **comply** with **security standards** and **regulations**.
- It guides to automatically **test** or **analyze** the code for security vulnerabilities in an effective and systematic way.

- However, it is **labor-intensive** and **difficult** to write and maintain documentation manually without any tool.



- Let's use [scaladoc](#) to **automatically generate documentation** from **comments** in Scala source code.
- It provides similar features to other comment based documentation systems like javadoc, jekyll, docusaurus, etc.

To **generate the documentation**, you can run the following command:

```
$ sbt doc
```

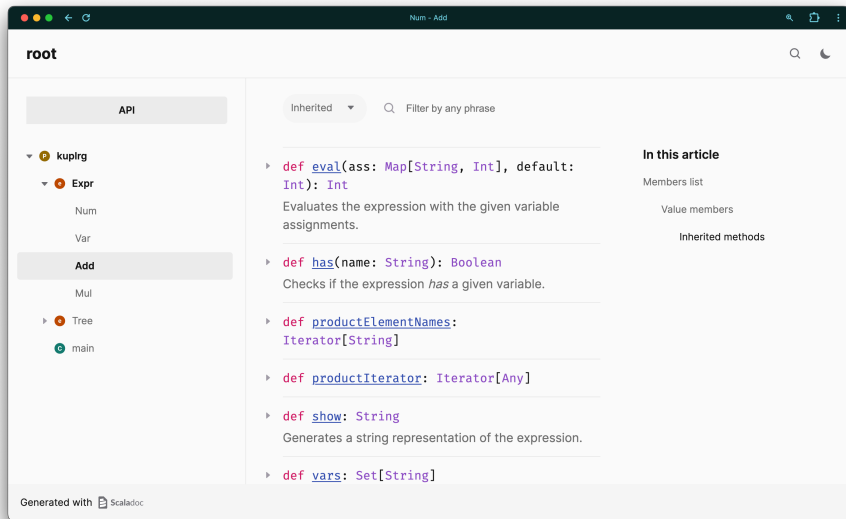
Then, the documentation will be generated in the directory:
target/scala-3.3.3/api.

Please enter the directory and run the server to see the documentation using python3:

```
$ cd target/scala-3.3.3/api  
$ python3 -m http.server 8080
```

Then, you can open the following URL in your web browser:

<http://localhost:8080>



root

API

▼ kuplrg

- ▼ Expr
 - Num
 - Var
 - Add**
 - Mul
- ▶ Tree
- main

Inherited ▾ 🔍 Filter by any phrase

- ▶ `def eval(ass: Map[String, Int], default: Int): Int`
Evaluates the expression with the given variable assignments.
- ▶ `def has(name: String): Boolean`
Checks if the expression *has* a given variable.
- ▶ `def productElementNames: Iterator[String]`
- ▶ `def productIterator: Iterator[Any]`
- ▶ `def show: String`
Generates a string representation of the expression.
- ▶ `def vars: Set[String]`

In this article

- Members list
- Value members
- Inherited methods

Generated with Scaladoc

You can write documentation using **comments** in the Scala source code for [scaladoc](#) with the following **tags**:

- Class/Method specific tags
 - `@constructor` – constructor
 - `@return` – which value is returned
 - `@throws` – which exceptions are thrown
 - `@param` – parameters
 - `@tparam` – type parameters
- Usage tags
 - `@see` – reference to other sources of information
 - `@note` – note for pre- or post- conditions
 - `@example` – example code
- Other tags
 - `@since` – when the feature was added
 - `@deprecated` – deprecated feature

You can use **HTML tags** or **markup** in the comments for [scaladoc](#):

```
`monospace`  
'italic text'  
'''bold text'''  
__underline__  
^superscript^  
,,subscript,,  
[[entity link]],  
    e.g. [[scala.collection.Seq]]  
[[https://external.link External Link]],  
    e.g. [[https://scala-lang.org Scala Language Site]]
```

There are other formatting supported by [scaladoc](#):

- **paragraphs** – started with one (or more) blank lines.
- **code blocks** – enclosed by {{{ and }}}.
- **table** – please refer to [here](#).
- **list blocks** – “-” for unordered list and “1.” for ordered list.

1. Simple Build Tool (sbt) for Scala

Example Project

Project Structure

Building a Project

Running a Project

2. Scala Documentation

scaladoc – Scala Documentation Tool

Generating Documentation

Writing Documentation

3. Scala Test Framework

Why Software Testing?

ScalaTest – Test Framework for Scala

Running Tests

Writing Tests

Measuring Code Coverage

Unexpected faults in **safety-critical software** cause serious problems:

 <p>June 4, 1996: Ariane-5 explodes after lift off Today In History: June 4, 1996: Ariane-5 explodes after lift off Original Source: 2008-04-08 Abdul Samad, Head of Archives</p>	 <p>Knight Capital Says Trading Glitch Cost It BY NATHANIEL POPPER AUGUST 2, 2012 6:07 AM 750 Runaway Trades Spread Turmoil Across Wall St.</p>	 <p>Heathrow Airport apologises for IT failure disruption 14 February 2020</p>	 <p>Cruise recalls all its driverless cars Hit another setback, Cruise updates software on 250 driverless cars to fix its 'Collision Data'</p>
Rocket	Financial	Airport	Auto. Vehicle
(1996)	(2012)	(2020)	(2023)

Then, how can we **prevent** such software faults?

Can we **automatically check** whether a program does not have any software faults?

How do we know whether a software is correct?



Empiricists – Francis Bacon

*It is correct because I **TESTED** several times but no error was found!*

VS.



Rationalists – René Descartes

*It is correct because I formally **PROVED** that no error exists!*



- Imagine you have two choices when boarding a airplane:
 - While an airplane A has **never been proven** to have any run-time errors, it has been **tested** with a finite number of test flights.
 - While an airplane B has been **formally verified** to have no run-time errors, it has **never been tested** in the real world.
- Some people may choose A, while others may choose B.
- In addition, some properties only can be **tested** but not **verified** (e.g., energy consumption, usability, etc.).

- ScalaTest is a **test framework** for Scala and Java Virtual Machine (JVM) that is designed to be **scalable** and **flexible**.
- It is designed to be **easy to learn** and **easy to use**.
- It is designed to be **easy to integrate** with other tools and libraries.
- It supports **different styles** of testing (e.g., FunSuite, FlatSpec, etc.).

We can **test the project** with the following command on [sbt](#):

```
$ sbt test
# [info] TreeSuite:
# [info] - The `has` should return if the tree has the value
# [info] - The `map` should map the tree with the given function
# [info] ...
# [info] ExprSpec:
# [info] `vars`
# [info] - should returns the set of variables in the expression
# [info] `show`
# [info] - should generate a string representation of the expression
# [info] ...
# [info] Run completed in 107 milliseconds.
# [info] Total number of tests run: 8
# [info] Suites: completed 2, aborted 0
# [info] Tests: succeeded 8, failed 0, canceled 0, ignored 0, pending 0
# [info] All tests passed.
# [success] Total time: 0 s, completed ...
```

For example, we can define a test suite for the arithmetic expression (Expr) using FlatSpec style as follows:

```
import org.scalatest.flatspec.AnyFlatSpec

class ExprSpec extends AnyFlatSpec {
  import Expr.*

  // 2 * (x + y)
  val expr3: Expr = Mul(Num(2), Add(Var("x"), Var("y")))

  "`vars`" should "returns the set of variables in the expression" in {
    assert(expr3.vars == Set("x", "y"))
  }

  "`show`" should "generate a string representation of the expression"
    in {
    assert(expr3.show == "2 * (x + y)")
  }
}
```

Or, we can define a test suite for the tree data structure (Tree) using FunSuite style as follows:

```
import org.scalatest.funsuite.AnyFunSuite

class TreeSuite extends AnyFunSuite {
  import Tree.*
  // 1
  // / \
  // 2  3
  val tree2: Tree = Node(1, List(Leaf(3), Leaf(2)))

  test("The `has` should return if the tree has the value") {
    assert(tree2.has(8) == false)
  }

  test("The `map` should map the tree with the given function") {
    assert(tree2.map(_ * 2) == Node(2, List(Leaf(6), Leaf(4))))
  }
}
```

- How to measure the **quality** of the tests?
- One possible way is to measure the **code coverage** of the tests.
- We can **measure the code coverage** of the project using [scoverage](#), the **code coverage tool** for Scala.

Measuring Code Coverage

First, we need to add coverage as a plugin in the `project/plugins.sbt` file:

```
addSbtPlugin("org.scoverage" % "sbt-scoverage" % "2.0.11")
```

and turn on the **coverage mode** in the `build.sbt` file:

```
coverageEnabled := true
```

Then, we need to run the tests with enabled coverage:

```
$ sbt clean coverage test
```







Finally, we can generate the coverage report:

```
$ sbt coverageReport
```

and open the following file in your web browser:

```
<project-dir>/target/scala-3.3.3/scoverage-report/index.html
```

It shows the **overall code coverage** of the project:

All packages	91.09%	Class	Source file	Lines	Methods	Statements	Invoked	Coverage	Branches	Invoked	Coverage		
kuprg	91.09%	App\$	App.scala	11	1	2	0		0.00 %	0	0		0.00 %
		Expr	Expr.scala	130	4	55	49		89.09 %	22	19		86.36 %
		Tree	Tree.scala	190	7	44	43		97.73 %	15	14		93.33 %

For example, we can see which parts are **not covered** by the tests:

```
91 *
92 * Add(Var("x"), Num(1)) // x + 1 = 3 + 1 = 4
93 *
94 * Mul(Num(2), Add(Var("x"), Var("y"))) // 2 * (x + y) = 2 * (3 + 5) = 16
95 * }}}
96 */
97 def eval(ass: Map[String, Int], default: Int): Int = this match
98   case Num(n) => n
99   case Var(x) => ass.get(x) match
100     case Some(n) => n
101     case None => default
102   case Add(l, r) => l.eval(ass, default) + r.eval(ass, default)
103   case Mul(l, r) => l.eval(ass, default) * r.eval(ass, default)
```



ESMeta is a framework that extracts a mechanized specification from a given version of ECMAScript/JavaScript specification (ECMA-262) developed using Scala and sbt.

<https://github.com/es-meta/esmeta>

1. Simple Build Tool (sbt) for Scala

- Example Project

- Project Structure

- Building a Project

- Running a Project

2. Scala Documentation

- scaladoc – Scala Documentation Tool

- Generating Documentation

- Writing Documentation

3. Scala Test Framework

- Why Software Testing?

- ScalaTest – Test Framework for Scala

- Running Tests

- Writing Tests

- Measuring Code Coverage

Exercise #1

- Please see this document on GitHub:

<https://github.com/ku-plrg-classroom/docs/tree/main/scala-tutorial>

- It is just an exercise, and it is **NOT** included in your grade.

- Classes, Traits, and Objects

Jihyeok Park
jihyeok_park@korea.ac.kr
<https://plrg.korea.ac.kr>