Basic Structure of Scala

Structure of Scala Programming

By Dhandapani Yedappalli Krishnamurthi Oct 6, 2025

🧩1️⃣Basic Scala Program Structure

A minimal Scala program looks like this:

```
// HelloWorld.scala
object HelloWorld {
  def main(args: Array[String]): Unit = {
    println("Hello, Scala!")
  }
}
```

🔍 Explanation

Component	Description
object HelloWorld	Defines a **singleton object** (no static keyword in Scala — everything lives in objects).
def main(args: Array[String]): Unit	Entry point of a Scala program (like public static void main in Java).
println("Hello, Scala!")	Prints text to the console.
Unit	Equivalent to void in Java — means "no return value."

Run it using:

```
scalac HelloWorld.scala   # compile
scala HelloWorld          # run
```

🧱2️⃣Structure with Class and Object

```
// Example: ClassExample.scala
class Greeting {
  def sayHello(name: String): Unit = {
    println(s"Hello, $name!")
  }
}
object ClassExample {
  def main(args: Array[String]): Unit = {
    val greet = new Greeting()
    greet.sayHello("Dani")
  }
}
```

💡 Key Points:

- class Greeting → defines a **class**.
- object ClassExample → companion object containing main.

- new Greeting() → creates an instance of the class.
- s"Hello, $name!" → **String interpolation** (inserts variable values).

🧠3️⃣Structure with Package and Imports

```
// File: src/com/example/BasicStructure.scala
package com.example
import java.time.LocalDateTime
object BasicStructure {
  def main(args: Array[String]): Unit = {
    val now = LocalDateTime.now
    println(s"Program started at: $now")
  }
}
```

💡 Explanation:

- package com.example → groups related code together.
- import → used to bring libraries/classes into scope.
- Follows Java-like directory structure:

```
src/
└── com/
    └── example/
        └── BasicStructure.scala
```

⚙️4️⃣Using Functions Outside main

```
object MathOps {
  def add(a: Int, b: Int): Int = a + b
  def multiply(a: Int, b: Int): Int = a * b
  def main(args: Array[String]): Unit = {
    println(s"Addition: ${add(10, 20)}")
    println(s"Multiplication: ${multiply(10, 20)}")
  }
}
```

🧾 Output:

```
Addition: 30
Multiplication: 200
```

🧩5️⃣Basic Skeleton Template

Here’s a template you can reuse:

```
package your.package.name
object YourProgramName {
  // Define constants or variables here
  val constantValue = 100
  // Define methods/functions here
  def yourFunction(param1: String): String = {
    s"Received: $param1"
  }
  // Main entry point
  def main(args: Array[String]): Unit = {
    println("=== Scala Program Started ===")
    println(yourFunction("Dani"))
    println("=== Program Completed ===")
  }
}
```

⚡ Quick Summary Table

Component	Purpose
package	Organize code
import	Include libraries
object	Entry point (singleton)
class	Define blueprint for objects
def	Define function/method
main()	Program starting point
println()	Print output
Unit	No return value

🧱1️⃣What is SBT?

SBT (**Scala Build Tool**) is like **Maven/Gradle for Java** — it:

- Compiles Scala code
- Manages dependencies (e.g., Spark, JDBC, etc.)
- Runs, tests, and packages applications

🗂️2️⃣Standard SBT Project Structure

Here's the recommended directory layout 👇

```
my-scala-project/
│
├── build.sbt
├── project/
│   └── build.properties
│
├── src/
│   ├── main/
│   │   ├── scala/
│   │   │   └── com/
│   │   │       └── example/
│   │   │           └── HelloWorld.scala
│   │   └── resources/
│   │       └── application.conf
│   │
│   └── test/
│       └── scala/
│           └── com/
│               └── example/
│
```

```
│                   └── HelloWorldTest.scala
│
└── README.md
```

⚙️3️⃣Step-by-Step Setup

🧩 Step 1: Create project folder

```
mkdir my-scala-project
cd my-scala-project
```

🧩 Step 2: Create build.sbt

```
name := "MyScalaProject"
version := "1.0"
scalaVersion := "2.13.16"   // or latest 2.13.x
libraryDependencies ++= Seq(
  "org.scalatest" %% "scalatest" % "3.2.19" % Test
)
```

📘 **Explanation**

Setting	Description
name	Project name
version	Version number
scalaVersion	Scala compiler version
libraryDependencies	External libraries (Scalatest for unit testing here)

🧩 Step 3: Create project/build.properties

```
sbt.version=1.10.2
```

🧠 This file locks the SBT version (like requirements.txt for Python).

🧩 Step 4: Create a simple Scala file

Path: src/main/scala/com/example/HelloWorld.scala

```
package com.example
object HelloWorld {
  def main(args: Array[String]): Unit = {
    println("Hello from SBT project!")
  }
}
```

🧩 Step 5: Create a simple test (optional)

Path: src/test/scala/com/example/HelloWorldTest.scala

```
package com.example
import org.scalatest.funsuite.AnyFunSuite
class HelloWorldTest extends AnyFunSuite {
  test("Sample test: true is true") {
    assert(true)
  }
}
```

🚀4️⃣Run the Project

Inside project root (my-scala-project/):

🧪 Compile

```
sbt compile
```

▶️ Run

```
sbt run
```

You'll see:

```
Hello from SBT project!
```

✅ Test

```
sbt test
```

🧰5️⃣Common SBT Commands

Command	Purpose
sbt compile	Compiles all source files
sbt run	Runs the main class
sbt test	Runs unit tests
sbt clean	Deletes old compiled files
sbt package	Creates JAR under target/scala-2.13/
sbt console	Opens interactive Scala REPL with dependencies loaded

🧩6️⃣Add Dependencies (Example)

If you want to use Spark + PostgreSQL (for ETL work):

```
libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "3.5.3" % "provided",
  "org.apache.spark" %% "spark-sql"  % "3.5.3" % "provided",
  "org.postgresql"   %  "postgresql" % "42.7.4",
  "org.scalatest"    %% "scalatest"  % "3.2.19" % Test
)
```

💡 Use % "provided" for libraries like Spark that your cluster already provides.

📦7️⃣Sample Program for Spark + Scala (Optional)

File: src/main/scala/com/example/SparkETL.scala

```
package com.example
import org.apache.spark.sql.SparkSession
object SparkETL {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("SimpleETLJob")
      .master("local[*]")
      .getOrCreate()
    import spark.implicits._
    val data = Seq(("Dani", 28), ("Anita", 32))
      .toDF("name", "age")
    data.show()
    spark.stop()
  }
}
```

Run with:

```
sbt run
```

Output:

```
+-----+---+
| name|age|
+-----+---+
| Dani| 28|
|Anita| 32|
+-----+---+
```

🧭8️⃣Project Summary

Folder	Purpose

src/main/scala	Main source code
src/main/resources	Configs, log4j, etc.
src/test/scala	Unit test files
project	SBT metadata
build.sbt	Build configuration
target	Auto-generated output (JARs, compiled code)

🧾 Final Tip

To open your project in **IntelliJ IDEA**:

1. Open IntelliJ → “Import Project” → Select build.sbt
2. IntelliJ will detect dependencies and set up the environment.
3. You can run from the IDE or terminal using sbt run.