# Scala Code Samples

Case Classes

By Dhandapani Yedappalli Krishnamurthi Oct 6, 2025

## 🧩 What is a Case Class in Scala?

A **case class** in Scala is a special kind of class that is:

- **Immutable by default**
- **Lightweight** and **boilerplate-free**
- **Designed to hold data**
- **Automatically supports pattern matching**

It is mostly used to **model data records**, similar to how you might use:

- A POJO in Java
- A dataclass in Python
- Or a simple "record" structure in ETL pipelines.

### 🧠 Why "case" class?

Because they are designed to be used in **pattern matching** —for example, you can easily *match cases* based on their data (not memory reference).

### ⚙️ Basic Syntax

case class ClassName(param1: Type1, param2: Type2, ...)

### ✅ Example 1 — Basic Case Class

```
case class Person(name: String, age: Int)

object Demo extends App {
  val p1 = Person("Aarav", 25)
  val p2 = Person("Diya", 30)
  println(p1)        // Person(Aarav,25)
  println(p1 == p2)   // false (value comparison)
}
```

💡 **No new keyword is needed** — the compiler automatically provides an apply() method to create instances.

### 🧩 Key Features of Case Classes

| Feature | Explanation |
|---|---|
| **1. Immutable fields** | All constructor parameters are treated as val by default. |
| **2. No new keyword** | You can create objects like Person("Aarav", 25). |
| **3. Auto-generated methods** | Scala generates toString, equals, hashCode, copy, and apply automatically. |

| | |
|---|---|
| **4. Pattern matching** | Easily used in match expressions. |
| **5. Companion object** | Automatically created by the compiler. |
| **6. Useful for Spark Dataset schema** | Perfect for defining structured data in Spark. |

## Example 1 — Basic Case Class

```
case class Person(name: String, age: Int)

object IndianNamesDemo extends App {
  val p1 = Person("Aarav", 25)
  val p2 = Person("Diya", 30)
  val p3 = p1.copy(name = "Kavya")

  println(p1)        // Person(Aarav,25)
  println(p2)        // Person(Diya,30)
  println(p3)        // Person(Kavya,25)
}
```

---

## Example 2 — Pattern Matching

```
case class Order(id: Int, amount: Double, customer: String)

object IndianOrders extends App {
  val order = Order(101, 7999.99, "Rahul Mehta")

  order match {
    case Order(_, amt, cust) if amt > 5000 => println(s"High-value
order by $cust")
    case Order(_, amt, cust) => println(s"Order by $cust: ₹$amt")
  }
}
```

✅ Output:

```
High-value order by Rahul Mehta
```

---

### Example 3 — Nested Case Classes

```
case class Address(city: String, pincode: String)
case class Employee(name: String, dept: String, address: Address)

object IndianEmployeeDemo extends App {
  val emp = Employee("Sneha Iyer", "Finance", Address("Chennai",
"600001"))
  println(s"${emp.name} works in ${emp.dept} at
${emp.address.city}")
}
```

✅ Output:

```
Sneha Iyer works in Finance at Chennai
```

---

### Example 4 — Case Class with List

```
case class Student(name: String, grade: String)

object IndianStudentsDemo extends App {
  val students = List(
    Student("Arjun Reddy", "A"),
    Student("Meera Singh", "B+"),
    Student("Vikram Nair", "A"),
    Student("Ananya Sharma", "A+")
  )
```

```
  students.foreach(s => println(s"${s.name} scored grade
${s.grade}"))
}
```

✅ Output:

```
Arjun Reddy scored grade A
Meera Singh scored grade B+
Vikram Nair scored grade A
Ananya Sharma scored grade A+
```

---

### Example 5 — Spark + Case Class with Indian Employee Data

```
import org.apache.spark.sql.SparkSession

case class Employee(id: Int, name: String, city: String, salary:
Double)

object IndianSparkExample {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder
      .appName("IndianEmployeeCaseClass")
      .master("local[*]")
      .getOrCreate()

    import spark.implicits._

    val data = Seq(
      Employee(1, "Rohan Patel", "Ahmedabad", 75000),
      Employee(2, "Priya Nair", "Kochi", 82000),
      Employee(3, "Karthik R", "Bangalore", 95000),
      Employee(4, "Sneha Iyer", "Chennai", 88000)
```

```
    )

    val ds = data.toDS()
    ds.show()
  }
}
```

✅ Output:

```
+---+-----------+----------+------+
| id|       name|      city|salary|
+---+-----------+----------+------+
|  1|Rohan Patel|Ahmedabad | 75000|
|  2| Priya Nair|     Kochi| 82000|
|  3|   Karthik R|Bangalore| 95000|
|  4|  Sneha Iyer|   Chennai| 88000|
+---+-----------+----------+------+
```

---

### 🚀1️⃣What is Scala?

**Scala** = **Scalable Language**
It's a **modern, hybrid programming language** that combines:

- **Object-Oriented Programming (OOP)** like Java
- **Functional Programming (FP)** like Python or Haskell

💡 Scala runs on the **JVM** (Java Virtual Machine) — so you can use **Java libraries** and integrate easily with Big Data tools like **Apache Spark**.

### ⚙️2️⃣Environment Setup

### ✅ Install Prerequisites

| Tool | Recommended Version (2025) |
|---|---|
| **Java (JDK)** | 11 or 17 |
| **Scala** | 2.12.x or 2.13.x |
| **Build Tool** | SBT (Scala Build Tool) |
| **IDE** | IntelliJ IDEA (Community or Ultimate) |

### ✅ Verify Installation

scala -version

```
sbt sbtVersion
```

## 💻3️⃣First Scala Program

### File: HelloWorld.scala

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

✅ Run:

```
scalac HelloWorld.scala
scala HelloWorld
```

✅ Output:

```
Hello, Scala World!
```

## 🔢4️⃣Variables and Data Types

### Immutable (val) vs Mutable (var)

```
val name: String = "Aarav"  // immutable
var age: Int = 25           // mutable
age = 26
```

| Type | Example |
|---|---|
| Int | 10 |
| Double | 10.5 |
| Boolean | true |
| String | "Hello" |

## 🧮5️⃣Expressions and Conditionals

Everything in Scala is an *expression* that returns a value.

```
val result = if (10 > 5) "Greater" else "Smaller"
println(result)   // Greater
```

## 🔁6️⃣Loops

```
for (i <- 1 to 5) println(s"Count: $i")
var sum = 0
while (sum < 10) {
  sum += 2
  println(sum)
}
```

## 🧠7️⃣Functions

### Simple function

```
def greet(name: String): String = {
  s"Welcome, $name!"
}
println(greet("Meera"))
```

### Function without return type

```
def add(a: Int, b: Int) = a + b
println(add(10, 5))
```

## 🧱8️⃣Classes and Objects

```
class Person(val name: String, val age: Int) {
  def showInfo(): Unit = println(s"$name is $age years old")
}
```

```
object Demo extends App {
  val p = new Person("Rohan", 30)
  p.showInfo()
}
```

## 🧩9️⃣Case Classes

```
case class Employee(name: String, dept: String)
object Test extends App {
  val emp = Employee("Sneha", "Finance")
  println(emp)
}
```

✅ Output:

```
Employee(Sneha,Finance)
```

## 📦 🔟 Collections

### List

```
val fruits = List("Apple", "Banana", "Mango")
fruits.foreach(println)
```

### Map

```
val marks = Map("Math" -> 95, "Science" -> 90)
println(marks("Math"))  // 95
```

### Tuple

```
val student = ("Rahul", 18, "Chennai")
println(student._1) // Rahul
```

## ⚡ 1️⃣1️⃣Higher-Order Functions

Functions can take other functions as arguments.

```
def applyFunc(x: Int, f: Int => Int): Int = f(x)
val result = applyFunc(5, x => x * x)
println(result)  // 25
```

## 🧱 1️⃣2️⃣Object-Oriented Concepts

### Inheritance Example

```
class Animal {
  def sound(): Unit = println("Animal sound")
}
class Dog extends Animal {
  override def sound(): Unit = println("Bark")
}
object InheritDemo extends App {
  val d = new Dog
  d.sound()
}
```

## 💡 1️⃣3️⃣Pattern Matching

```
val x = 2
x match {
  case 1 => println("One")
  case 2 => println("Two")
  case _ => println("Other number")
}
```

## 🧾 1️⃣4️⃣Option and Some/None

To safely handle null-like scenarios.

```
val maybeName: Option[String] = Some("Kavya")
println(maybeName.getOrElse("No name"))
```

### 🧮 15️⃣ Try / Catch for Exceptions

```
try {
  val result = 10 / 0
} catch {
  case e: ArithmeticException => println("Cannot divide by zero")
}
```

### 🔁 16️⃣ Working with Collections – Map, Filter, Reduce

```
val numbers = List(1, 2, 3, 4, 5)
val squares = numbers.map(x => x * x)
val evens = numbers.filter(_ % 2 == 0)
val sum = numbers.reduce(_ + _)
println(squares)  // List(1, 4, 9, 16, 25)
println(evens)    // List(2, 4)
println(sum)      // 15
```

### 🧱 17️⃣ Companion Objects

```
class Student(val name: String)
object Student {
  def apply(name: String): Student = new Student(name)
}
val s = Student("Riya")  // no 'new' keyword
println(s.name)
```

### ✅ Summary

| Concept | Example |
|---|---|
| Print | println("Hello") |
| Variable | val x = 10 |
| Function | def add(a:Int,b:Int)=a+b |
| Class | class Person(...) |
| Object | object Demo extends App {...} |
| Case Class | case class Employee(name:String,dept:String) |
| List | List(1,2,3) |
| Map | Map("A"->1,"B"->2) |
| Pattern Match | x match { case ... } |

**Recommended Scala Tutorial / Learning Repos**

| Repo | Highlights / Why It's Useful |
|---|---|
| **jetbrains-academy/scala-tutorial** | A concise Scala tutorial covering basics: types, control, OOP, FP, etc. GitHub |
| **abdheshkumar/scala-examples** | A rich set of Scala code examples: case classes, pattern matching, closures, recursion, etc. GitHub |
| **rockthejvm/udemy-scala-beginners** | Code from Rock the JVM's beginner Scala course — good for following along with lessons. GitHub |
| **handsonscala/handsonscala** | Executable code for many Scala topics, organized by chapters (good one to drill practice). GitHub |
| **Baeldung/scala-tutorials** | Sample code supporting Scala tutorials with modular structure and topics. GitHub |

| | |
|---|---|
| **amir2b/scala-tutorial** | A simple Scala tutorial repo for beginners (syntax, OOP, FP) GitHub |
| **lukaszlenart/scala-basics** | A workshop-style repo introducing Scala fundamentals: SBT, collections, higher-order functions, etc. GitHub |