Practical Apache Spark

Using the Scala API

Subhashini Chellappan
Dharanitharan Ganesan
# Table of Contents

About the Authors ............................................................................................................ ix
About the Technical Reviewers ......................................................................................... xi
Acknowledgments ............................................................................................................. xiii
Introduction ......................................................................................................................... xv

## Chapter 1: Scala: Functional Programming Aspects ......................................................... 1

What Is Functional Programming? ..................................................................................... 2
  What Is a Pure Function? ................................................................................................. 2
  Example of Pure Function .............................................................................................. 3

Scala Programming Features ............................................................................................. 4
  Variable Declaration and Initialization .......................................................................... 5
  Type Inference .................................................................................................................. 6
  Immutability .................................................................................................................... 7
  Lazy Evaluation .............................................................................................................. 8
  String Interpolation ....................................................................................................... 10
  Pattern Matching .......................................................................................................... 13

Scala Class vs. Object ....................................................................................................... 14
Singleton Object ............................................................................................................... 15
Companion Classes and Objects ....................................................................................... 17
Case Classes ..................................................................................................................... 18
Scala Collections .............................................................................................................. 21

Functional Programming Aspects of Scala .................................................................... 27
  Anonymous Functions .................................................................................................... 27
  Higher Order Functions ................................................................................................. 29
  Function Composition .................................................................................................... 30
  Function Currying .......................................................................................................... 31


# TABLE OF CONTENTS

- Nested Functions ............................................................................................................. 32
- Functions with Variable Length Parameters ................................................................. 34
- Reference Links .................................................................................................................. 37
- Points to Remember .......................................................................................................... 37

## Chapter 2: Single and Multinode Cluster Setup ............................................................... 39
- Spark Multinode Cluster Setup .......................................................................................... 39
  - Recommended Platform ..................................................................................................... 39
  - Prerequisites .................................................................................................................... 61
  - Spark Installation Steps .................................................................................................. 62
  - Spark Web UI ................................................................................................................... 66
  - Stopping the Spark Cluster ............................................................................................ 70
- Spark Single-Node Cluster Setup ....................................................................................... 70
  - Prerequisites .................................................................................................................... 71
  - Spark Installation Steps .................................................................................................. 73
  - Spark Master UI .............................................................................................................. 76
- Points to Remember .......................................................................................................... 77

## Chapter 3: Introduction to Apache Spark and Spark Core ................................................ 79
- What Is Apache Spark? ....................................................................................................... 80
- Why Apache Spark? ........................................................................................................... 80
- Spark vs. Hadoop MapReduce .......................................................................................... 81
- Apache Spark Architecture ............................................................................................... 82
- Spark Components .......................................................................................................... 84
  - Spark Core (RDD) ........................................................................................................... 84
  - Spark SQL ......................................................................................................................... 84
  - Spark Streaming ............................................................................................................. 85
  - MLlib ................................................................................................................................. 85
  - GraphX ............................................................................................................................ 85
  - SparkR .............................................................................................................................. 85
  - Spark Shell ..................................................................................................................... 85
  - Spark Core: RDD ............................................................................................................ 86
# RDD Operations

- Creating an RDD ................................................................. 88
- RDD Transformations .......................................................... 91
- RDD Actions ........................................................................ 95
- Working with Pair RDDs ...................................................... 98
- Direct Acyclic Graph in Apache Spark ..................................... 101
  - How DAG Works in Spark ................................................ 101
  - How Spark Achieves Fault Tolerance Through DAG .......... 103
- Persisting RDD ................................................................. 104
- Shared Variables ............................................................ 105
  - Broadcast Variables ......................................................... 106
  - Accumulators .................................................................... 106
- Simple Build Tool (SBT) ..................................................... 107
- Assignments ........................................................................ 112
- Reference Links ................................................................... 112
- Points to Remember .......................................................... 113

# Chapter 4: Spark SQL, DataFrames, and Datasets ....................... 115

- What Is Spark SQL? ............................................................. 116
  - Datasets and DataFrames ................................................ 116
- Spark Session ...................................................................... 116
- Creating DataFrames .......................................................... 117
  - DataFrame Operations ..................................................... 118
  - Running SQL Queries Programmatically .......................... 121
  - Dataset Operations ........................................................... 123
  - Interoperating with RDDs ............................................... 125
  - Different Data Sources .................................................... 129
  - Working with Hive Tables ................................................. 133
  - Building Spark SQL Application with SBT ....................... 135
- Points to Remember .......................................................... 139
# Table of Contents

## Chapter 5: Introduction to Spark Streaming ................................. 141
- Data Processing ................................................................. 142
- Streaming Data .................................................................. 142
  - Why Streaming Data Are Important .................................. 142
- Introduction to Spark Streaming ........................................... 142
  - Internal Working of Spark Streaming .................................. 143
- Spark Streaming Concepts .................................................. 144
- Spark Streaming Example Using TCP Socket ......................... 145
- Stateful Streaming .............................................................. 149
  - Window-Based Streaming .................................................. 149
  - Full-Session-Based Streaming .............................................. 152
- Streaming Applications Considerations ................................ 155
- Points to Remember ............................................................ 156

## Chapter 6: Spark Structured Streaming ................................. 157
- What Is Spark Structured Streaming? .................................... 158
- Spark Structured Streaming Programming Model ................... 158
  - Word Count Example Using Structured Streaming ............... 158
- Creating Streaming DataFrames and Streaming Datasets ........... 160
- Operations on Streaming DataFrames/Datasets ....................... 163
- Stateful Streaming: Window Operations on Event-Time ............ 167
- Stateful Streaming: Handling Late Data and Watermarking ........ 170
- Triggers .............................................................................. 171
- Fault Tolerance ................................................................. 173
- Points to Remember ............................................................ 174

## Chapter 7: Spark Streaming with Kafka .............................. 175
- Introduction to Kafka ........................................................ 175
  - Kafka Core Concepts .......................................................... 176
  - Kafka APIs ...................................................................... 176
- Kafka Fundamental Concepts .............................................. 177
- Kafka Architecture ............................................................ 178
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kafka Topics</td>
<td>179</td>
</tr>
<tr>
<td>Leaders and Replicas</td>
<td>179</td>
</tr>
<tr>
<td>Setting Up the Kafka Cluster</td>
<td>180</td>
</tr>
<tr>
<td>Spark Streaming and Kafka Integration</td>
<td>182</td>
</tr>
<tr>
<td>Spark Structure Streaming and Kafka Integration</td>
<td>185</td>
</tr>
<tr>
<td>Points to Remember</td>
<td>187</td>
</tr>
<tr>
<td>Chapter 8: Spark Machine Learning Library</td>
<td>189</td>
</tr>
<tr>
<td>What Is Spark MLlib?</td>
<td>190</td>
</tr>
<tr>
<td>Spark MLlib APIs</td>
<td>190</td>
</tr>
<tr>
<td>Vectors in Scala</td>
<td>191</td>
</tr>
<tr>
<td>Basic Statistics</td>
<td>194</td>
</tr>
<tr>
<td>Extracting, Transforming, and Selecting Features</td>
<td>200</td>
</tr>
<tr>
<td>ML Pipelines</td>
<td>215</td>
</tr>
<tr>
<td>Points to Remember</td>
<td>236</td>
</tr>
<tr>
<td>Chapter 9: Working with SparkR</td>
<td>237</td>
</tr>
<tr>
<td>Introduction to SparkR</td>
<td>237</td>
</tr>
<tr>
<td>SparkDataFrame</td>
<td>237</td>
</tr>
<tr>
<td>SparkSession</td>
<td>238</td>
</tr>
<tr>
<td>Starting SparkR from RStudio</td>
<td>238</td>
</tr>
<tr>
<td>Creating SparkDataFrames</td>
<td>241</td>
</tr>
<tr>
<td>From a Local R DataFrame</td>
<td>241</td>
</tr>
<tr>
<td>From Other Data Sources</td>
<td>242</td>
</tr>
<tr>
<td>From Hive Tables</td>
<td>243</td>
</tr>
<tr>
<td>SparkDataFrame Operations</td>
<td>244</td>
</tr>
<tr>
<td>Selecting Rows and Columns</td>
<td>244</td>
</tr>
<tr>
<td>Grouping and Aggregation</td>
<td>245</td>
</tr>
<tr>
<td>Operating on Columns</td>
<td>247</td>
</tr>
<tr>
<td>Applying User-Defined Functions</td>
<td>248</td>
</tr>
<tr>
<td>Run a Given Function on a Large Data Set Using dapply or dapplyCollect</td>
<td>248</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

Running SQL Queries from SparkR ................................................................. 249  
Machine Learning Algorithms ................................................................... 250  
  Regression and Classification Algorithms .............................................. 250  
  Logistic Regression ................................................................................. 255  
  Decision Tree ......................................................................................... 258  
Points to Remember .................................................................................. 260  

**Chapter 10: Spark Real-Time Use Case** ................................................... 261  
  Data Analytics Project Architecture ......................................................... 262  
    Data Ingestion ....................................................................................... 262  
    Data Storage ......................................................................................... 263  
    Data Processing .................................................................................... 263  
    Data Visualization ............................................................................... 264  
Use Cases ..................................................................................................... 264  
  Event Detection Use Case ...................................................................... 264  
  Build Procedure ....................................................................................... 270  
  Building the Application with SBT ........................................................... 271  
Points to Remember .................................................................................. 273  

**Index** ....................................................................................................... 275  

---

viii
About the Authors

Subhashini Chellappan is a technology enthusiast with expertise in the big data and cloud space. She has rich experience in both academia and the software industry. Her areas of interest and expertise are centered on business intelligence, big data analytics and cloud computing.

Dharanitharan Ganesan has an MBA in technology management with a high level of exposure and experience in big data, using Apache Hadoop, Apache Spark, and various Hadoop ecosystem components. He has a proven track record of improving efficiency and productivity through the automation of various routine and administrative functions in business intelligence and big data technologies. His areas of interest and expertise are centered on machine learning algorithms, Blockchain in big data, statistical modeling, and predictive analytics.
About the Technical Reviewers

**Mukund Kumar Mishra** is a senior technologist with strong business acumen. He has more than 18 years of international experience in business intelligence, big data, data science, and computational analytics. He is a regular speaker on big data concepts, Hive, Hadoop, and Spark. Before joining the world of big data, he also worked extensively in the Java and .NET space.

Mukund is also a poet and his first book of poetry was published when he was only 15 years old. Thus far he has written around 300 poems. He runs one of the largest Facebook groups on big data Hadoop (see [https://www.facebook.com/groups/656180851123299/](https://www.facebook.com/groups/656180851123299/)). You can connect with Mukund on LinkedIn at [https://www.linkedin.com/in/mukund-kumar-mishra-7804b38/](https://www.linkedin.com/in/mukund-kumar-mishra-7804b38/).

**Sundar Rajan Raman** has more than 14 years of full stack IT experience, including special interests in machine learning, deep learning, and natural language processing. He has 6 years of big data development and architecture experience including Hadoop and its ecosystems and other No SQL technologies such as MongoDB and Cassandra. He is a design thinking practitioner interested in strategizing using design thinking principles.

Sundar is active in coaching and mentoring people. He has mentored many teammates who are now in respectable positions in their careers.
Acknowledgments

The making of this book was a journey that we are glad we undertook. The journey spanned a few months, but the experience will last a lifetime. We had our families, friends, colleagues, and well-wishers onboard for this journey, and we wish to express our deepest gratitude to each one of them.

We would like to express our special thanks to our families, friends, and colleagues, who provided that support that allowed us to complete this book within a limited time frame.

Special thanks are extended to our technical reviewers for the vigilant review and filling in with their expert opinion.

We would like to thank Celestin Suresh John, Senior Manager, Apress and Springer Science and Business Media, for signing us up for this wonderful creation. We wish to acknowledge and appreciate Aditee Mirashi, coordinating editor, and the team who guided us through the entire process of preparation and publication.
Introduction

Why This Book?

Apache Spark is a fast, open source, general-purpose memory processing engine for big data processing. This book discusses various components of Apache Spark, such as Spark Core, Spark SQL DataFrames and Datasets, Spark Streaming, Structured Streaming, Spark machine learning libraries, and SparkR with practical code snippets for each module. It also covers the integration of Apache Spark with other ecosystem components such as Hive and Kafka. The book has within its scope the following:

* Functional programming features of Scala.
* Architecture and working of different Spark components.
* Work on Spark integration with Hive and Kafka.
* Using Spark SQL DataFrames and Datasets to process the data using traditional SQL queries.
* Work with different machine learning libraries in Spark MLlib packages.

Who Is This Book For?

The audience for this book includes all levels of IT professionals.

How Is This Book Organized?

Chapter 1 describes the functional programming aspects of Scala with code snippets. In Chapter 2, we explain the steps for Spark installation and cluster setup. Chapter 3 describes the need for Apache Spark and core components of Apache Spark. In Chapter 4, we explain how to process structure data using Spark SQL, DataFrames, and Datasets. Chapter 5 provides the basic concepts of Spark Streaming and Chapter 6 covers the
basic concepts of Spark Structure Streaming. In Chapter 7, we describe how to integrate Apache Spark with Apache Kafka. Chapter 8 then explains the machine learning library of Apache Spark. In Chapter 9, we address how to integrate Spark with R. Finally, in Chapter 10 we provide some real-time use cases for Apache Spark.

How Can I Get the Most Out of This Book?

It is easy to leverage this book for maximum gain by reading the chapters thoroughly. Get hands-on by following the step-by-step instructions provided in the demonstrations. Do not skip any of the demonstrations. If need be, repeat them a second time or until the concept is firmly etched in your mind. Happy learning!!!

Subhashini Chellappan
Dharanitharan Ganesan
CHAPTER 1

Scala: Functional Programming Aspects

This chapter is a prerequisite chapter that provides a high-level overview of functional programming aspects of Scala. This chapter helps you understand the functional programming aspects of Scala. Scala is a preferred language to work with Apache Spark. After this chapter, you will be able to understand the building blocks of functional programming and how to apply functional programming concepts in your daily programming tasks. There is a hands-on focus in this chapter and the entire chapter introduces short programs and code snippets as illustrations of important functional programming features.

The recommended background for this chapter is some prior experience with Java or Scala. Experience with any other programming language is also sufficient. Also, having some familiarity with the command line is preferred.

By end of this chapter, you will be able to do the following:

- Understand the essentials of functional programming.
- Combine functional programming with objects and classes.
- Understand the functional programming features.
- Write functional programs for any programming tasks.

Note: It is recommended that you practice the code snippets provided and practice the exercises to develop effective knowledge of the functional programming aspects of Scala.
What Is Functional Programming?

Functional programming (FP) is a way of writing computer programs as the evaluation of mathematical functions, which avoids changing the state or mutating data. The programs are constructed using pure functions. Functional programs are always declarative, where the programming is done with declarations and expressions instead of statements. Functional programming languages are categorized into two groups:

1. Pure function
2. Impure function

What Is a Pure Function?

A function that has no side effects is called a pure function. So, what are side effects? A function is said to be having side effects if it does any of the following other than just returning a result:

- Modifies an existing variable.
- Reads from a file or writes to a file.
- Modifies a data structure (e.g., array, list).
- Modifies an object (setting a field in an object).

The output of a pure function depends only on the input parameter passed to the function. The pure function will always give the same output for the same input arguments, irrespective of the number of times it is called.

The impure function can give different output every time it is called and the output of the function is not dependent only on the input parameters.

**Hint** Let us try to understand pure and impure functions using some Java concepts (if you are familiar with). The mutator method (i.e., the setter method) is an impure function and the accessor method (i.e., the getter method) is a pure function.
Example of Pure Function

The following function is an example of a pure function:

```scala
def squareTheNumber(num : Int) :Int ={
  return num*num
}
```

The function `squareTheNumber` (see Figure 1-1) accepts an integer parameter and always returns the square of the number. Because it has no side effects and the output is dependent only on the input parameter, it is considered a pure function.

```scala
scala> def squareTheNumber(num : Int) :Int ={
|   return num*num
| }
scala> squareTheNumber: (num: Int)Int
scala> squareTheNumber(10)
res3: Int = 100
scala>
```

Figure 1-1. Example of a pure function

Here are some the typical examples of pure functions:

- Mathematical functions such as addition, subtraction, division, and multiplication.
- String class methods like `length`, `toUpperCase`, and `toLowerCase`.

These are some typical examples of impure functions:

- A function that generates a random number.
- Date methods like `getDate()` and `getTime()` as they return different values based on the time they are called.
PURE AND IMPURE FUNCTIONS EXERCISE

1. Find the type of function and give the reason.
   
   ```scala
def myFunction(a : Int) :Int ={
    return a
}
```

2. Find the type of function and give the reason.
   
   ```scala
def myFunction() : Double = {
    var a = Math.random()
    return a
}
```

3. The following function is said to be an impure function. Why?
   
   ```scala
def myFunction(emp : Employee) : Double = {
    emp.setSalary(100000)
    return emp.getSalary()
}
```

4. Give five differences between pure functions and impure functions.

5. A function named `acceptUserInput()` contains a statement to get input from the console. Identify whether the function is pure or impure and justify the reason.

---

**Note**  The last statement of the function is always a `return` statement in Scala. Hence, it is not necessary to explicitly specify the `return` keyword.

The semicolon is not needed to specify the end of a statement in Scala. By default, the newline character (`\n`) is considered the end of a statement. However, a semicolon is needed if multiple statements are to be written in a single line.

---

**Scala Programming Features**

Let us turn to the Scala programming features, as illustrated in 1-2.
Variable Declaration and Initialization

The variables can be declared through `var` and `val` keywords. The difference between `var` and `val` is explained later in this chapter. The code here describes `val` and `var`:

```scala
val bookId=100
var bookId=100
```

Figure 1-3 displays the output.

```scala
scala> val bookId = 100
bookId: Int = 100

scala> var bookId = 100
bookId: Int = 100
```

Figure 1-3. Variable declaration and initialization
Type Inference

In Scala, it is not mandatory to specify the data type of variables explicitly. The compiler can identify the type of variable based on the initialization of the variable by the built-in type inference mechanism. The following is the syntax for declaring the variable:

```
var <variable_name> : [<data_type>] = <value>
```

The `[<data_type>]` is optional. The code describes type inference mechanism.

```scala
var bookId = 101
var bookName = "Practical Spark"
```

Refer to Figure 1-4 for the output.

![Type inference without an explicit type specification](image1.png)

**Figure 1-4.** Type inference without an explicit type specification

However, you can explicitly specify the type for variables during declaration as shown here:

```scala
var bookId:Int = 101
var bookName:String = "Practical Spark"
```

Figure 1-5 shows the output.

![Type inference with an explicit type specification](image2.png)

**Figure 1-5.** Type inference with an explicit type specification
Immutability

Immutability means the value of a variable cannot be changed once it is declared. The keyword `val` is used to declare immutable variables, whereas mutable variables can be declared using the keyword `var`. Data immutability helps you achieve concurrency control while managing data. The following code illustrates a mutable variable.

```scala
var bookName = "Spark"
bookName = "Practical Spark"
print("The book Name is" + bookName)
```

Figure 1-6 shows mutable variables.

Figure 1-6.  **Mutable variables using the var keyword**

Hence, variable reassignment is possible if the variable is declared using the `var` keyword. The code shown here illustrates an immutable variable.

```scala
val bookName = "Spark"
bookName = "Practical Spark"
```

Refer to Figure 1-7 for immutable variables.
As you can see, variable reassignment is not possible if the variable is declared using the `val` keyword.

**Hint** Declaring immutable variables using the `val` keyword is like declaring `final` variables in Java.

### Lazy Evaluation

The lazy evaluation feature allows the user to defer the execution of any expression until it is needed using the `lazy` keyword. When the expression is declared with the `lazy` keyword, it will be executed only when it is being called explicitly. The following code and Figure 1-8 illustrates immediate expression evaluation.

```scala
class Interpreter {
  def interpret(expression: String): String = {
    // Evaluate the expression
    println(expression)
    expression
  }
}

val x = 10
val y = 10
val sum = x+y
```
In the following code the expression \( y \) is defined with the `lazy` keyword. Hence, it is evaluated only when it is called. Refer to Figure 1-9 for the output.

```scala
val x = 10
val y = 10
lazy val y = 10
print(sum)
```

Figure 1-8. **Immediete expression evaluation without the `lazy` keyword**

In the following code the expression \( y \) is defined with the `lazy` keyword. Hence, it is evaluated only when it is called. Refer to Figure 1-9 for the output.

```scala
val x = 10
val y = 10
lazy val y = 10
print(sum)
```

Figure 1-9. **Lazy evaluation with the `lazy` keyword**
It is important to note that the lazy evaluation feature can be used only with `val` (i.e., immutable variables). Refer to the code given here and Figure 1-10.

```scala
var x = 10
var y = 10
lazy sum = x+y
```

![Figure 1-10. Lazy evaluation cannot be used with mutable variables](image)

**String Interpolation**

String interpolation is the process of creating a string from the data. The user can embed the references of any variable directly into the processed string literals and format the string. The code shown here describes string processing without using string interpolation.

```scala
var bookName = "practical Spark"
println("The Book name is" + bookName)
```

Refer to Figure 1-11 for the output

```scala
scala> var bookName = "Practical Spark"
bookName: String = Practical Spark

scala> println("The Book name is "+bookName)
The Book name is Practical Spark
```

*Figure 1-11. String processing without using interpolation*
These are the available string interpolation methods:

- s interpolator.
- f interpolator.
- raw interpolator.

String - s Interpolator

Using the interpolator s, to the string literal allows the user to use the reference variables to append the data directly. The following code illustrates the s interpolator and the result is shown in Figure 1-12.

```
var bookName = "practical Spark"
println(s"The Book name is $bookName")
```

```
scala> var bookName = "Practical Spark"
bookName: String = Practical Spark

scala> println(s"The Book name is $bookName")
The Book name is Practical Spark
```

**Figure 1-12. String processing using the s interpolator**

Observe the difference in `println` method syntax to form the string with and without string interpolation.

Also, the arbitrary expressions can be evaluated using the string interpolators, as shown in the following code. Refer to Figure 1-13 for the output.

```
val x = 10
val y =15
println(s"The sum of $x and $y is ${x+y}")
```

String - f Interpolator

Scala offers a new mechanism to create strings from your data. Using the interpolator \( f \) to the string literal allows the user to create the formatted string and embed variable references directly in processed string literals. The following code illustrates the \( f \) interpolator and the output is shown in Figure 1-14.

```scala
class ChapTer1 scala:
SCaLa: FunCTIoNaL prOgraMMI ng aSPECTS

```
The formats allowed after % are based on string format utilities available from Java.

**String - raw Interpolator**

The raw interpolator does not allow the escaping of literals. For example, using \n with the raw interpolator does not return a newline character. The following code illustrates the raw interpolator and the output is shown in Figure 1-15.

```scala
val bookId = 101
val bookName = "Practical Spark"
println(s"The book id is $bookId. \n The book name is $bookName")
println(raw"The id is $bookId. \n The book name is $bookName")
```

```scala
scala> val bookId = 101
bookId: Int = 101
scala> val bookName = "Practical Spark"
bookName: String = Practical Spark
scala> println(s"The book id is $bookId. \n The book name is $bookName")
The book id is 101.
The book name is Practical Spark
scala> println(raw"The book id is $bookId. \n The book name is $bookName")
The book id is 101. \n The book name is Practical Spark
```

**Figure 1-15. String processing using the raw interpolator**

**Pattern Matching**

The process of checking a pattern against a value is called pattern matching. A successful match returns a value associated with the case. Here is the simple syntax to use pattern matching.

```scala
<reference_name> match {
  case <option 1> => <return_value 1>
  case <option 2> => <return_value 2>
  case <option n> => <return_value n>
  case <default_option> => <default return_value>
}
```
The pattern matching expression can be defined for a function as shown here.

```scala
def chapterName(chapterNo:Int) = chapterNo match {
  case 1 => "Scala Features"
  case 2 => "Spark core"
  case 3 => "Spark Streaming"
  case _ => "Chapter not defined"
}
```

Refer to Figure 1-16 for the output.

```scala
scala> def chapterName(chapterNo:Int) = chapterNo match {
|   case 1 => "Scala Features"
|   case 2 => "Spark core"
|   case 3 => "Spark Streaming"
|   case _ => "Chapter not defined"
| }
| CHAPTERName: (chapterNo: Int)String
| scala> chapterName(1)
| res30: String = Scala Features
| scala> chapterName(5)
| res31: String = Chapter not defined
```

*Figure 1-16. Example for pattern matching*

**Scala Class vs. Object**

A class is a collection of variables, functions, and objects that is defined as a blueprint for creating objects (i.e., instances). A Scala class can be instantiated (object can be created). The following code describes class and objects.

```scala
scala> class SparkBook {
|   val bookId = 101
|   val bookName = "Practical Spark"
|   val bookAuthor = "Dharanitharan G"
|   def printBookDetails(){
|     println(s"The $bookName is written by $bookAuthor")
|   }
| }
```
defined class SparkBook

scala> val book = new SparkBook()
book: SparkBook = SparkBook@96be74

scala> book.printBookDetails()
The Practical Spark is written by Dharanitharan G

Figure 1-17 displays the output.

Figure 1-17. Example for class and objects

The functions in the class can be called by using the object reference. The new keyword is used to create an object, or instance of the class.

Singleton Object

Scala classes cannot have static variables and methods. Instead, a Scala class can have a singleton object or companion object. You can use singleton object when there is a need for only one instance of a class. A singleton is also a Scala class but it has only one instance (i.e., Object). The singleton object cannot be instantiated (object creation). It can be created using the object keyword. The functions and variables in the singleton object can be directly called without object creation. The code shown here describes SingletonObject and the output is displayed in Figure 1-18.
Generally, the `main` method is created in a singleton object. Hence, the compiler need not create an object to call the `main` method while executing. Add the following code in a `.scala` file and execute it in a command prompt (REPL) to understand how the Scala compiler calls the `main` method in singleton object.

```scala
object SingletonObjectMainDemo {
  def main(args: Array[String]) {
    println("This is printed in main method")
  }
}
```

Save this code as `SingletonObjectMainDemo.scala` and execute the program using these commands at the command prompt.

```
scalac SingletonObjectMainDemo.scala
scala SingletonObjectMainDemo
```

The `scalac` keyword invokes the compiler and generates the byte code for `SingletonObjectMainDemo`. The `scala` keyword is used to execute the byte code generated by compiler. The output is shown in Figure 1-19.

![Singleton object](image)

**Figure 1-18. Singleton object**