Index:

1. Introduction

The PriceBasket program is a command line-based program to calculate the total price of a basket of goods, considering any special offers. It accepts a list of input and provides the subtotal special offer discounts and the final price as output. This documentation provides instructions on how to install, use and test the program.

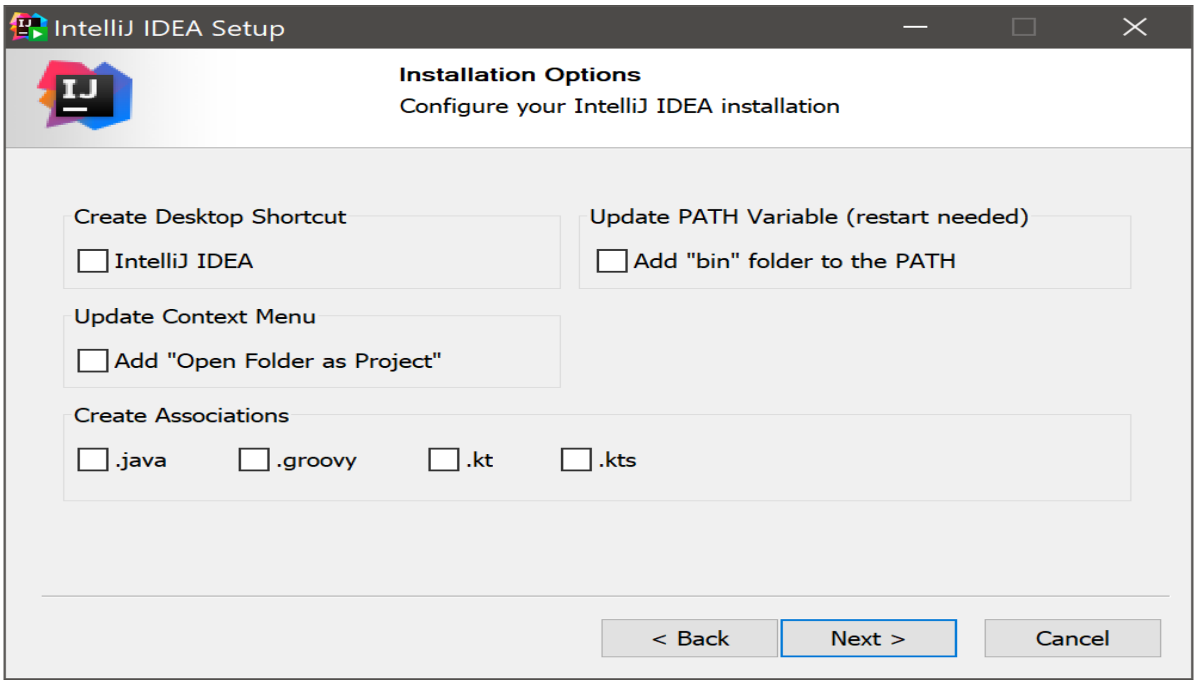
2. Installation

First, we need to install and set up the development environment with necessary dependencies.

Prerequisites:

- IntelliJ IDE
- Scala 2.13.12
- Java Development Kit (JDK) 1.8 or Later
- Download Scala and SBT plugins.
- Operating System- Any

Here are some screenshots of the installations.


IntelliJ IDEA Setup

Installation Options
Configure your IntelliJ IDEA installation

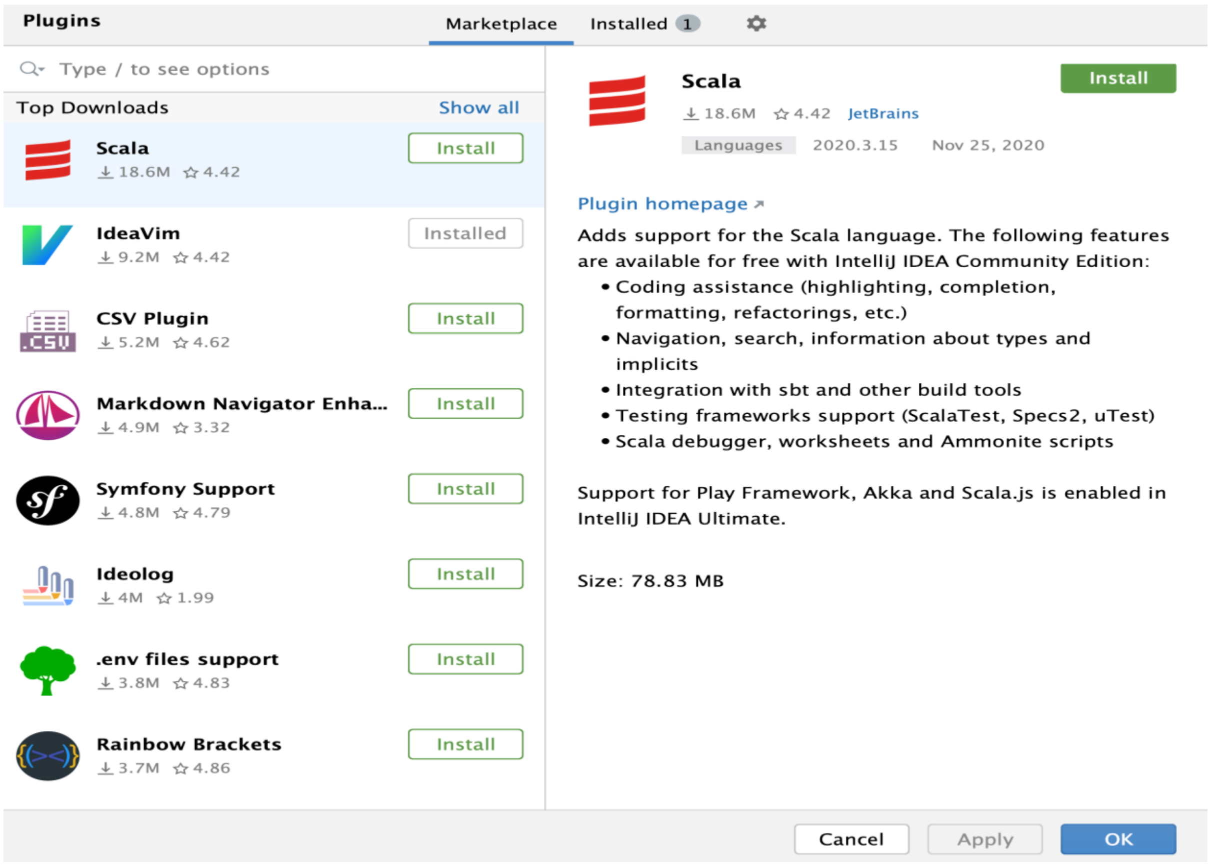
Create Desktop Shortcut
☐ IntelliJ IDEA

Update PATH Variable (restart needed)
☐ Add "bin" folder to the PATH

Update Context Menu
☐ Add "Open Folder as Project"

Create Associations
☐ .java ☐ .groovy ☐ .kt ☐ .kts

Fig1. IntelliJ Installation



Scala	⤓18.6M ☆4.42	Install
IdeaVim	⤓9.2M ☆4.42	Installed
CSV Plugin	⤓5.2M ☆4.62	Install
Markdown Navigator Enha...	⤓4.9M ☆3.32	Install
Symfony Support	⤓4.8M ☆4.79	Install
Ideolog	⤓4M ☆1.99	Install
.env files support	⤓3.8M ☆4.83	Install
Rainbow Brackets	⤓3.7M ☆4.86	Install

Scala

⤓18.6M ☆4.42 JetBrains

Languages 2020.3.15 Nov 25, 2020

Plugin homepage ↗

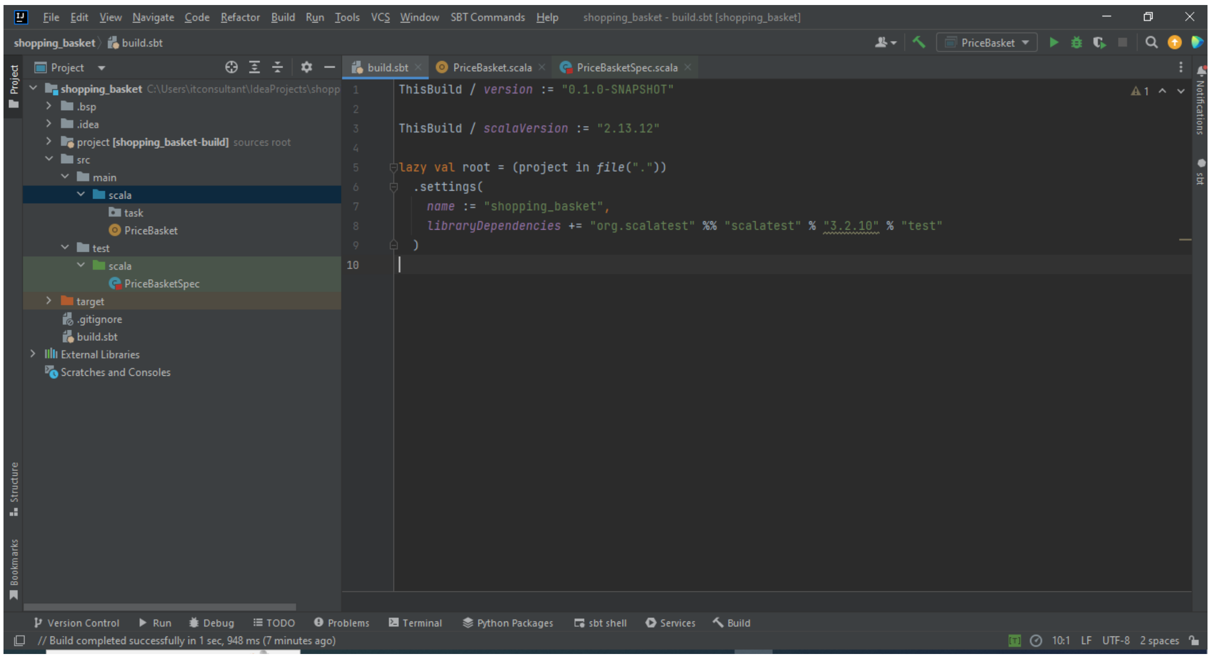
Adds support for the Scala language. The following features are available for free with IntelliJ IDEA Community Edition:

- Coding assistance (highlighting, completion, formatting, refactorings, etc.)
- Navigation, search, information about types and implicits
- Integration with sbt and other build tools
- Testing frameworks support (ScalaTest, Specs2, uTest)
- Scala debugger, worksheets and Ammonite scripts

Support for Play Framework, Akka and Scala.js is enabled in IntelliJ IDEA Ultimate.

Size: 78.83 MB

Fig2. Scala plugin Installation



```
ThisBuild / version := "0.1.0-SNAPSHOT"

ThisBuild / scalaVersion := "2.13.12"

lazy val root = (project in file("."))
  .settings(
    name := "shopping_basket",
    libraryDependencies += "org.scalatest" %% "scalatest" % "3.2.10" % "test"
  )
```

```
// Build completed successfully in 1 sec, 948 ms (7 minutes ago)
```

Fig3. Project

3. Project steps

Here the project has been created by following below steps:

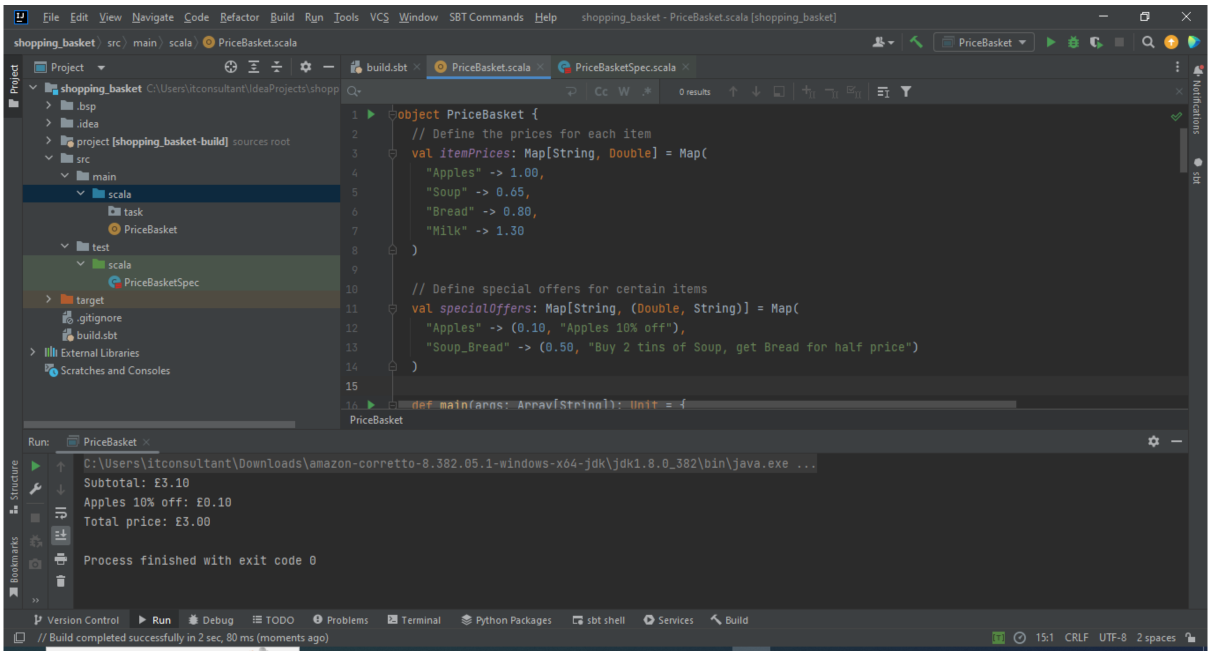
- Once installing IntelliJ IDE, the first step is to add or download plugins for Scala and SBT.
- For the same, click on settings -> plugin -> search for Scala and SBT plugins. Download JDK 1.8 and set environment variables as well.
- To create a new project, click on -> New project-> specify name of the project and select Scala. Also, there we need to add JDK path and build system as SBT.
- Furthermore, here I have created a package named Task. For that just go to src -> main -> Scala and right click on it to create a new package.
- Now, for creating a new Scala class right click on that package i.e., Task -> new Scala class named **PriceBasket**.
- Also, for creating unit test file right click on 'Test' folder in IntelliJ -> Scala-> new Scala class named **PriceBasketSpec**. The test folder holds all unit test files.

4. Program execution steps

- To run the PriceBasket.scala file, first need to provide correct command-line arguments. For example, Apples Milk Bread or Soup Soup Bread. Make sure to add space between the arguments.
- For the above step, click on our Scala file i.e., '**PriceBasket'** on top-middle and then go to the Edit Configuration option where you can pass those above inputs. Also, there we need to add JDK path and select Module which is nothing but Project name i.e., **'shopping_basket'** and provide main class as **Price_Basket**.
- After finishing the above steps click on Apply and ok. Now we are ready to click the run option. In this way we can execute the Scala program.
- For executing unit test file, we need to click on Run option, and we can observe the output on IntelliJ command-line.

5. Output/ Results

The below figure shows the output for the program. Here we passed **Apples Milk Bread** as input. The program is calculating the final price for items mentioned including this week's apples discount.

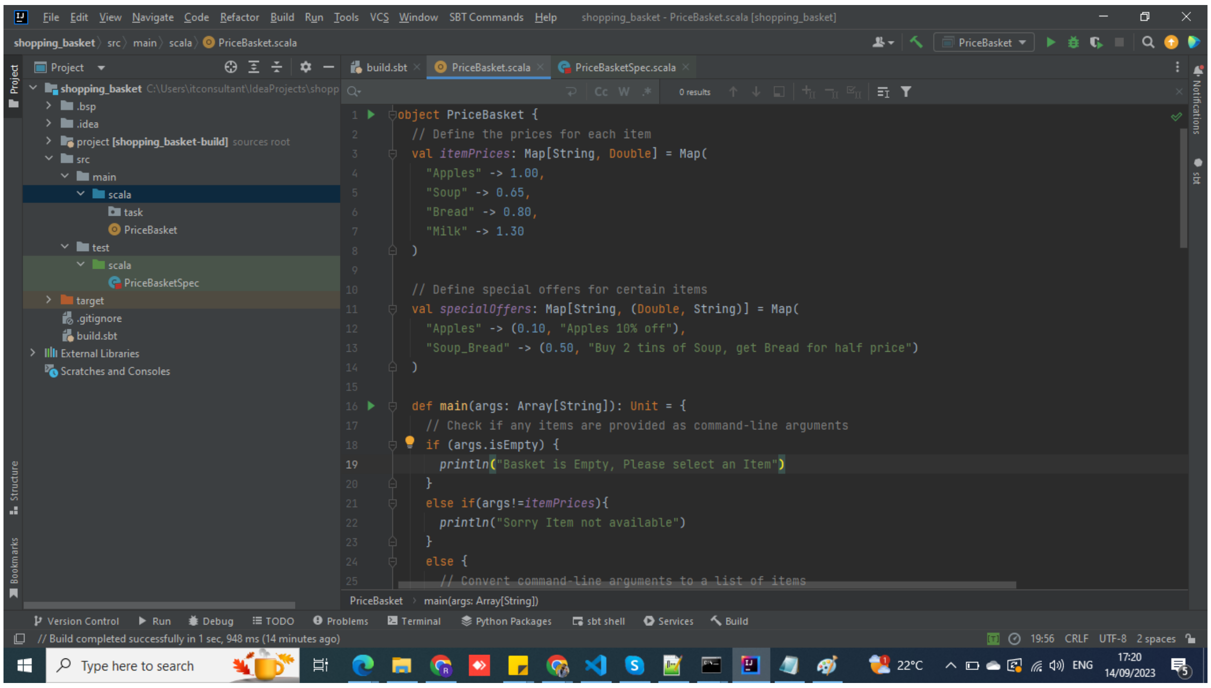


```scala
object PriceBasket {
  // Define the prices for each item
  val itemPrices: Map[String, Double] = Map(
    "Apples" -> 1.00,
    "Soup" -> 0.65,
    "Bread" -> 0.80,
    "Milk" -> 1.30
  )

  // Define special offers for certain items
  val specialOffers: Map[String, (Double, String)] = Map(
    "Apples" -> (0.10, "Apples 10% off"),
    "Soup_Bread" -> (0.50, "Buy 2 tins of Soup, get Bread for half price")
  )

  def main(args: Array[String]): Unit = {
```

```
C:\Users\itconsultant\Downloads\amazon-corretto-8.382.05.1-windows-x64-jdk\jdk1.8.0_382\bin\java.exe ...
Subtotal: £3.10
Apples 10% off: £0.10
Total price: £3.00

Process finished with exit code 0
```

// Build completed successfully in 2 sec, 80 ms (moments ago)

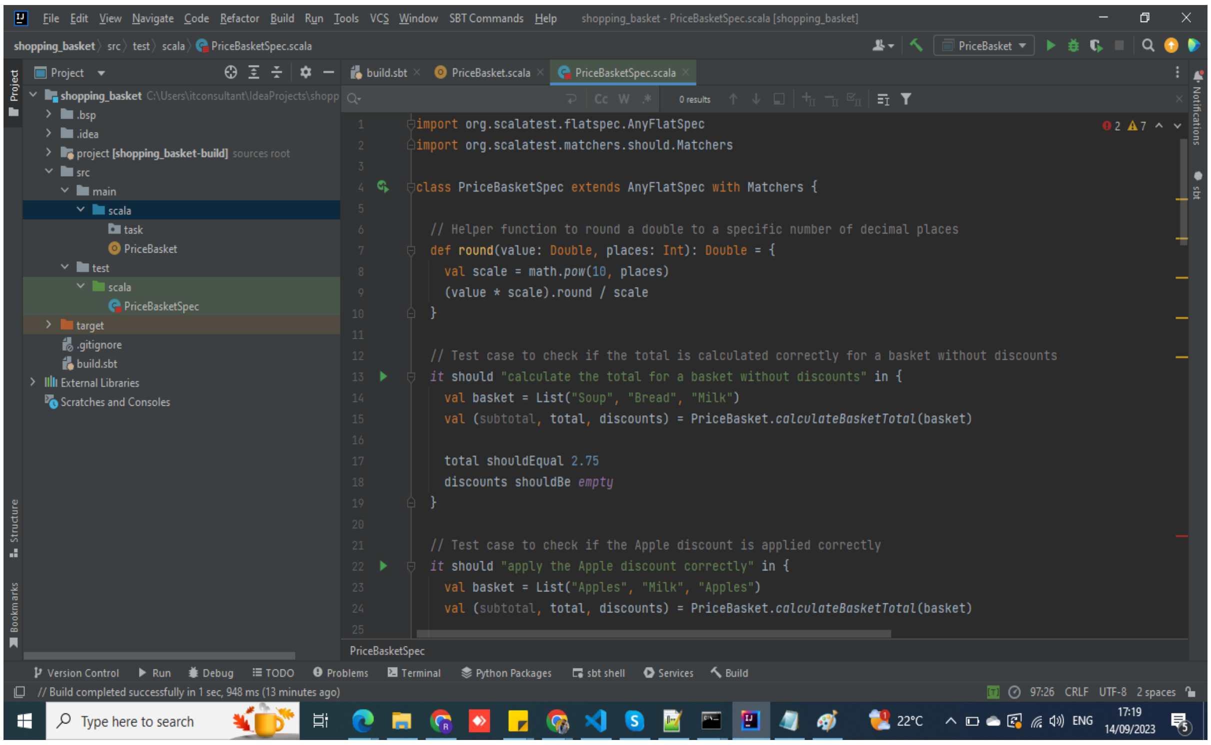
Fig4. Result



```scala
object PriceBasket {
  // Define the prices for each item
  val itemPrices: Map[String, Double] = Map(
    "Apples" -> 1.00,
    "Soup" -> 0.65,
    "Bread" -> 0.80,
    "Milk" -> 1.30
  )

  // Define special offers for certain items
  val specialOffers: Map[String, (Double, String)] = Map(
    "Apples" -> (0.10, "Apples 10% off"),
    "Soup_Bread" -> (0.50, "Buy 2 tins of Soup, get Bread for half price")
  )

  def main(args: Array[String]): Unit = {
    // Check if any items are provided as command-line arguments
    if (args.isEmpty) {
      println("Basket is Empty, Please select an Item")
    }
    else if(args!=itemPrices){
      println("Sorry Item not available")
    }
    else {
      // Convert command-line arguments to a list of items
```

// Build completed successfully in 1 sec, 948 ms (14 minutes ago)

Fig5. Code for 'PriceBasket'

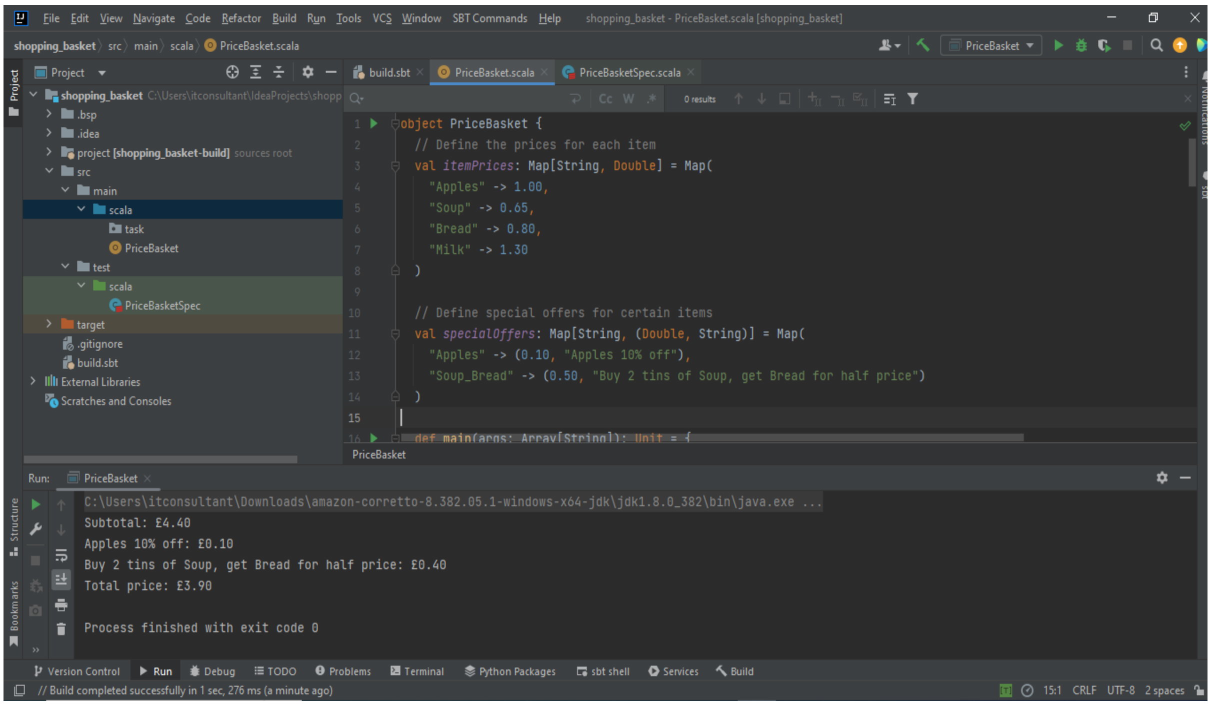


```scala
import org.scalatest.flatspec.AnyFlatSpec
import org.scalatest.matchers.should.Matchers

class PriceBasketSpec extends AnyFlatSpec with Matchers {

  // Helper function to round a double to a specific number of decimal places
  def round(value: Double, places: Int): Double = {
    val scale = math.pow(10, places)
    (value * scale).round / scale
  }

  // Test case to check if the total is calculated correctly for a basket without discounts
  it should "calculate the total for a basket without discounts" in {
    val basket = List("Soup", "Bread", "Milk")
    val (subtotal, total, discounts) = PriceBasket.calculateBasketTotal(basket)

    total shouldEqual 2.75
    discounts shouldBe empty
  }

  // Test case to check if the Apple discount is applied correctly
  it should "apply the Apple discount correctly" in {
    val basket = List("Apples", "Milk", "Apples")
    val (subtotal, total, discounts) = PriceBasket.calculateBasketTotal(basket)
```

Fig6. Unit test code

6. Examples

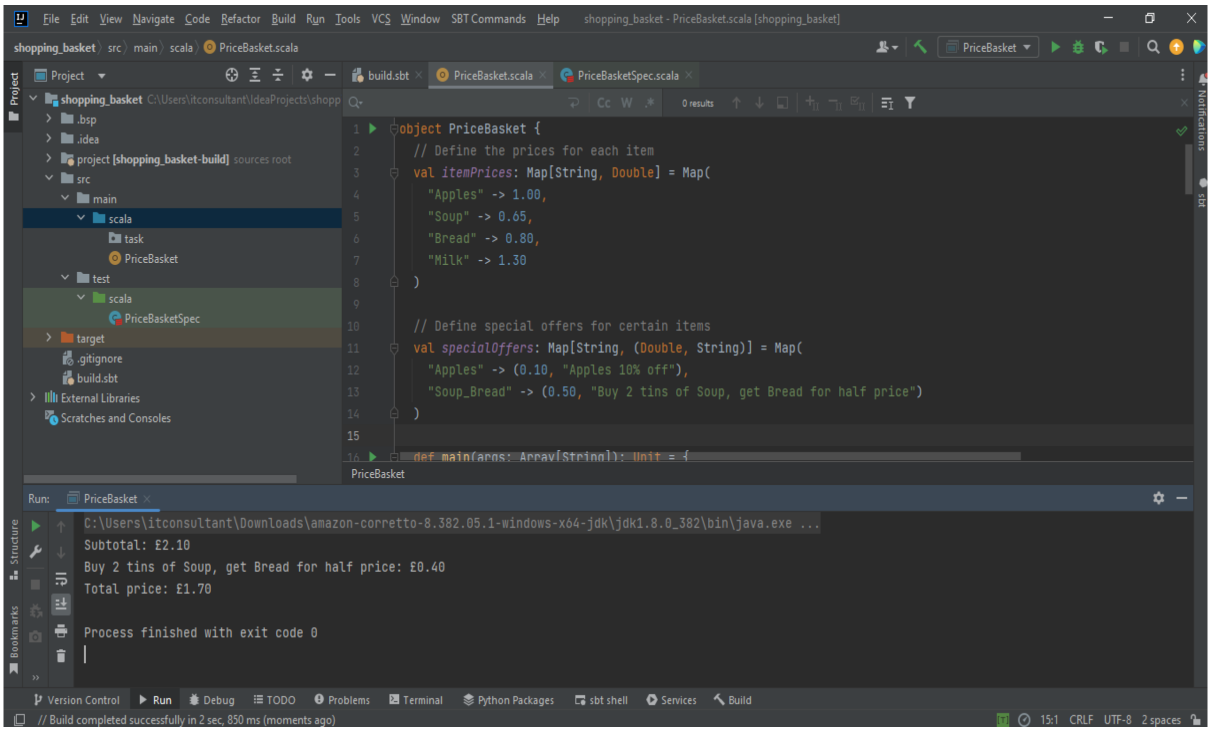
The below figures are the examples with different inputs.



```scala
object PriceBasket {
  // Define the prices for each item
  val itemPrices: Map[String, Double] = Map(
    "Apples" -> 1.00,
    "Soup" -> 0.65,
    "Bread" -> 0.80,
    "Milk" -> 1.30
  )

  // Define special offers for certain items
  val specialOffers: Map[String, (Double, String)] = Map(
    "Apples" -> (0.10, "Apples 10% off"),
    "Soup_Bread" -> (0.50, "Buy 2 tins of Soup, get Bread for half price")
  )

  def main(args: Array[String]): Unit = {
```

```
C:\Users\itconsultant\Downloads\amazon-corretto-8.382.05.1-windows-x64-jdk\jdk1.8.0_382\bin\java.exe ...
Subtotal: £4.40
Apples 10% off: £0.10
Buy 2 tins of Soup, get Bread for half price: £0.40
Total price: £3.90

Process finished with exit code 0
```

Fig7. Inputs as Apples Milk Bread Soup Soup

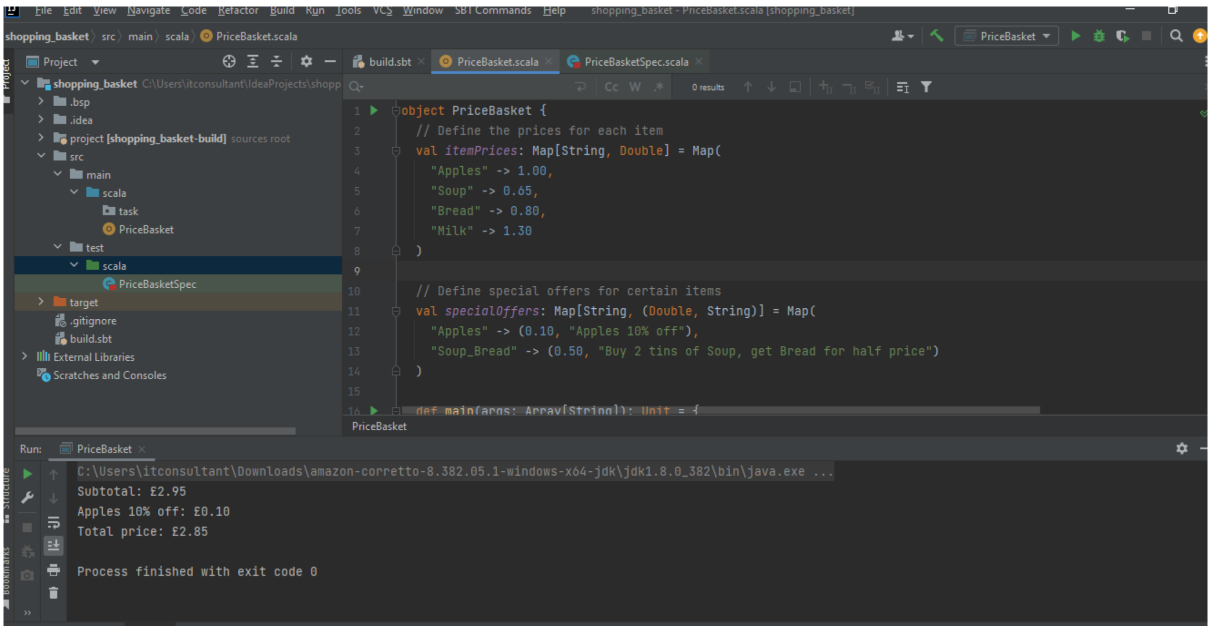


```scala
object PriceBasket {
  // Define the prices for each item
  val itemPrices: Map[String, Double] = Map(
    "Apples" -> 1.00,
    "Soup" -> 0.65,
    "Bread" -> 0.80,
    "Milk" -> 1.30
  )

  // Define special offers for certain items
  val specialOffers: Map[String, (Double, String)] = Map(
    "Apples" -> (0.10, "Apples 10% off"),
    "Soup_Bread" -> (0.50, "Buy 2 tins of Soup, get Bread for half price")
  )

  def main(args: Array[String]): Unit = {
```

```
C:\Users\itconsultant\Downloads\amazon-corretto-8.382.05.1-windows-x64-jdk\jdk1.8.0_382\bin\java.exe ...
Subtotal: £2.10
Buy 2 tins of Soup, get Bread for half price: £0.40
Total price: £1.70

Process finished with exit code 0
```

```
// Build completed successfully in 2 sec, 850 ms (moments ago)
```

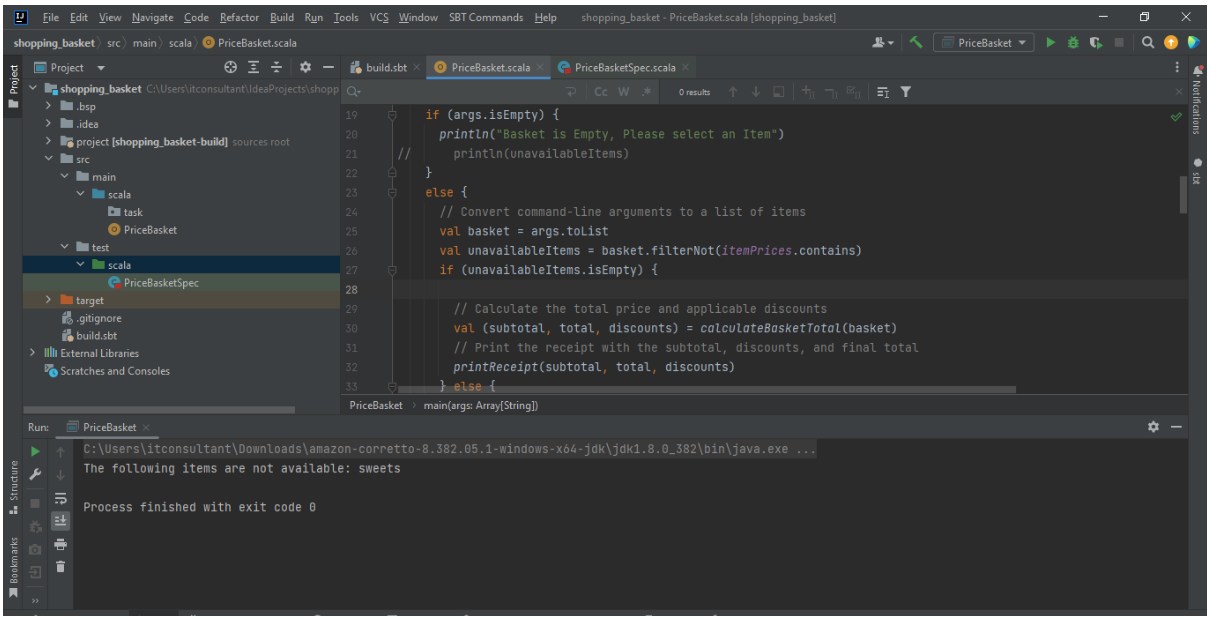
Fig8. Inputs as Two Soups-Bread



```scala
object PriceBasket {
  // Define the prices for each item
  val itemPrices: Map[String, Double] = Map(
    "Apples" -> 1.00,
    "Soup" -> 0.65,
    "Bread" -> 0.80,
    "Milk" -> 1.30
  )

  // Define special offers for certain items
  val specialOffers: Map[String, (Double, String)] = Map(
    "Apples" -> (0.10, "Apples 10% off"),
    "Soup_Bread" -> (0.50, "Buy 2 tins of Soup, get Bread for half price")
  )

  def main(args: Array[String]): Unit = {
```

```
C:\Users\itconsultant\Downloads\amazon-corretto-8.382.05.1-windows-x64-jdk\jdk1.8.0_382\bin\java.exe ...
Subtotal: £2.95
Apples 10% off: £0.10
Total price: £2.85

Process finished with exit code 0
```

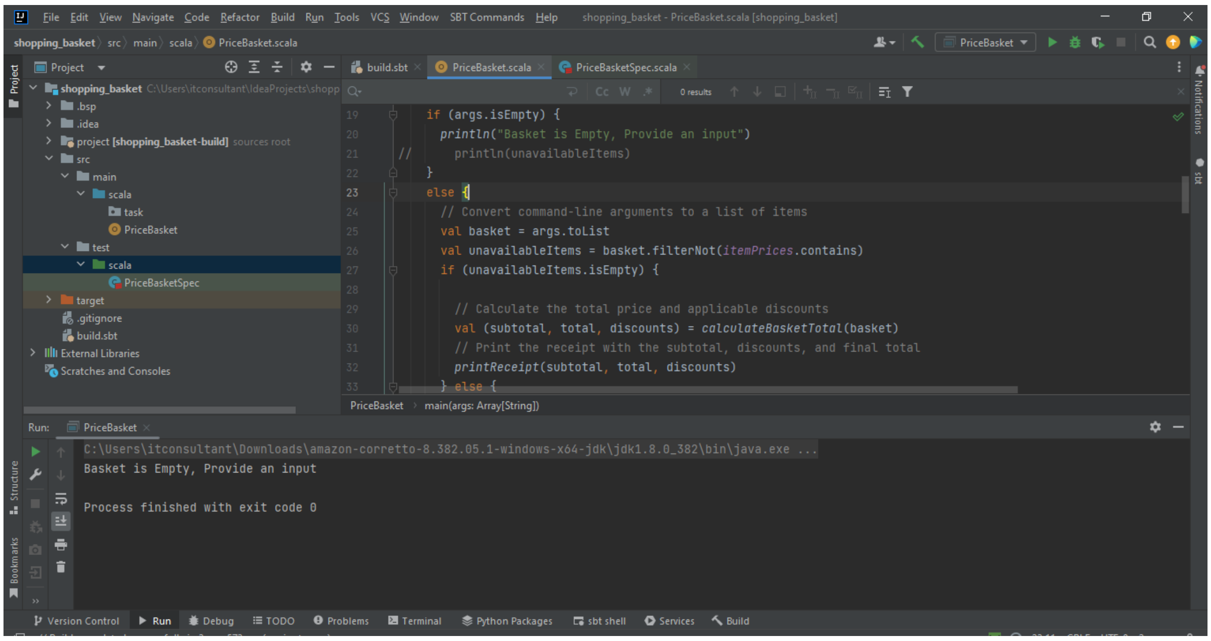
Fig9. Input as Apples Milk Soup



```scala
if (args.isEmpty) {
  println("Basket is Empty, Please select an Item")
//    println(unavailableItems)
}
else {
  // Convert command-line arguments to a list of items
  val basket = args.toList
  val unavailableItems = basket.filterNot(itemPrices.contains)
  if (unavailableItems.isEmpty) {

    // Calculate the total price and applicable discounts
    val (subtotal, total, discounts) = calculateBasketTotal(basket)
    // Print the receipt with the subtotal, discounts, and final total
    printReceipt(subtotal, total, discounts)
  } else {
```

```
C:\Users\itconsultant\Downloads\amazon-corretto-8.382.05.1-windows-x64-jdk\jdk1.8.0_382\bin\java.exe ...
The following items are not available: sweets

Process finished with exit code 0
```

Fig10. Item Not Available



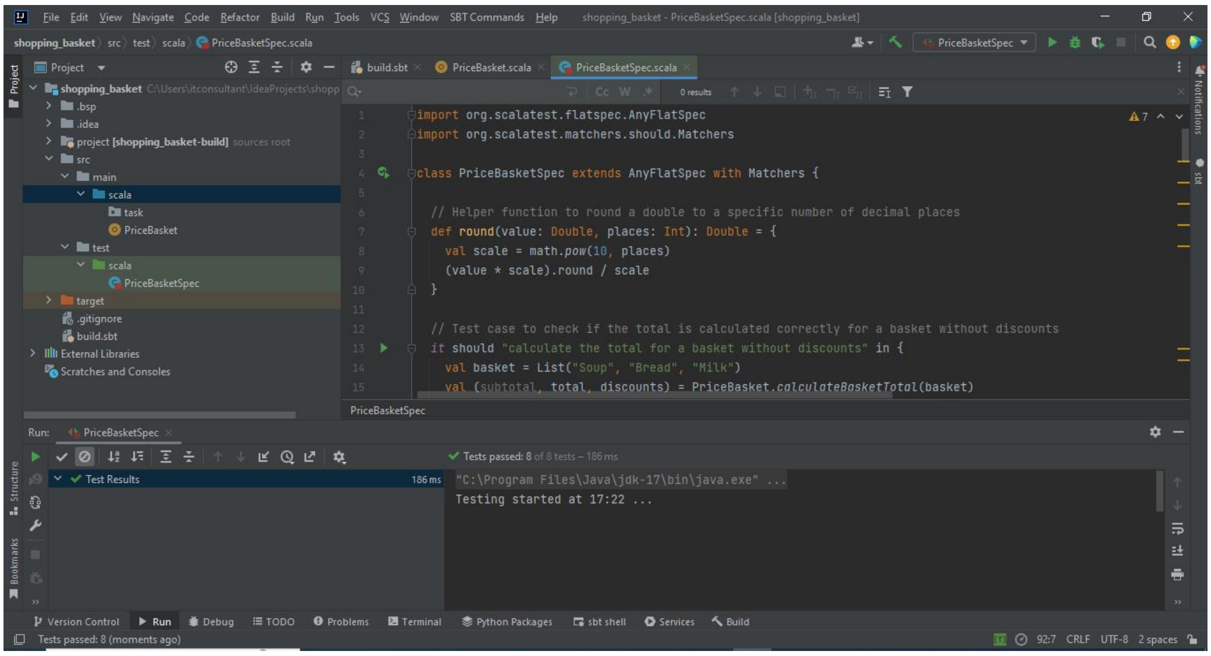
```scala
if (args.isEmpty) {
  println("Basket is Empty, Provide an input")
//    println(unavailableItems)
}
else {
  // Convert command-line arguments to a list of items
  val basket = args.toList
  val unavailableItems = basket.filterNot(itemPrices.contains)
  if (unavailableItems.isEmpty) {

    // Calculate the total price and applicable discounts
    val (subtotal, total, discounts) = calculateBasketTotal(basket)
    // Print the receipt with the subtotal, discounts, and final total
    printReceipt(subtotal, total, discounts)
  } else {
```

```
C:\Users\itconsultant\Downloads\amazon-corretto-8.382.05.1-windows-x64-jdk\jdk1.8.0_382\bin\java.exe ...
Basket is Empty, Provide an input

Process finished with exit code 0
```

Fig11. Empty Basket

7. Unit Test Cases

Unit tests have been included to ensure the correctness and reliability of the PriceBasket program by creating unit test file i.e., 'PriceBasketSpec.scala' file. To run unit tests, use the run option in IntelliJ IDE. The below figure shows us the program had successfully passed all test cases.



```
import org.scalatest.flatspec.AnyFlatSpec
import org.scalatest.matchers.should.Matchers

class PriceBasketSpec extends AnyFlatSpec with Matchers {

  // Helper function to round a double to a specific number of decimal places
  def round(value: Double, places: Int): Double = {
    val scale = math.pow(10, places)
    (value * scale).round / scale
  }

  // Test case to check if the total is calculated correctly for a basket without discounts
  it should "calculate the total for a basket without discounts" in {
    val basket = List("Soup", "Bread", "Milk")
    val (subtotal, total, discounts) = PriceBasket.calculateBasketTotal(basket)
```

```
Tests passed: 8 of 8 tests – 186 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Testing started at 17:22 ...
```

Fig12. Unit tests

8. Troubleshooting

If there are any errors or issues encountered, then make sure to check and pass correct command line arguments.

9. Reference

- https://www.tutorialspoint.com/scala/scala_pattern_matching.htm
- https://github.com/minalmahajan3/task_price_basket.git

10. Appendix

a. You can follow the readme file for executing the project step by step.