

ASSIGNMENT 16.1

Problem Statement:

Given a list of numbers - List[Int] (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

1. Find the sum of all numbers
2. Find the total elements in the list
3. Calculate the average of the numbers in the list
4. Find the sum of all the even numbers in the list
5. Find the total number of elements in the list divisible by both 5 and 3

Solution:

Let's create a spark context and then convert the given list into a Spark parallelized RDD (Resilient Distributed Dataset). A **parallelized RDD** is a collection of distributed data elements and is a basic unit of data abstraction on which various map/reduce operations can be performed.

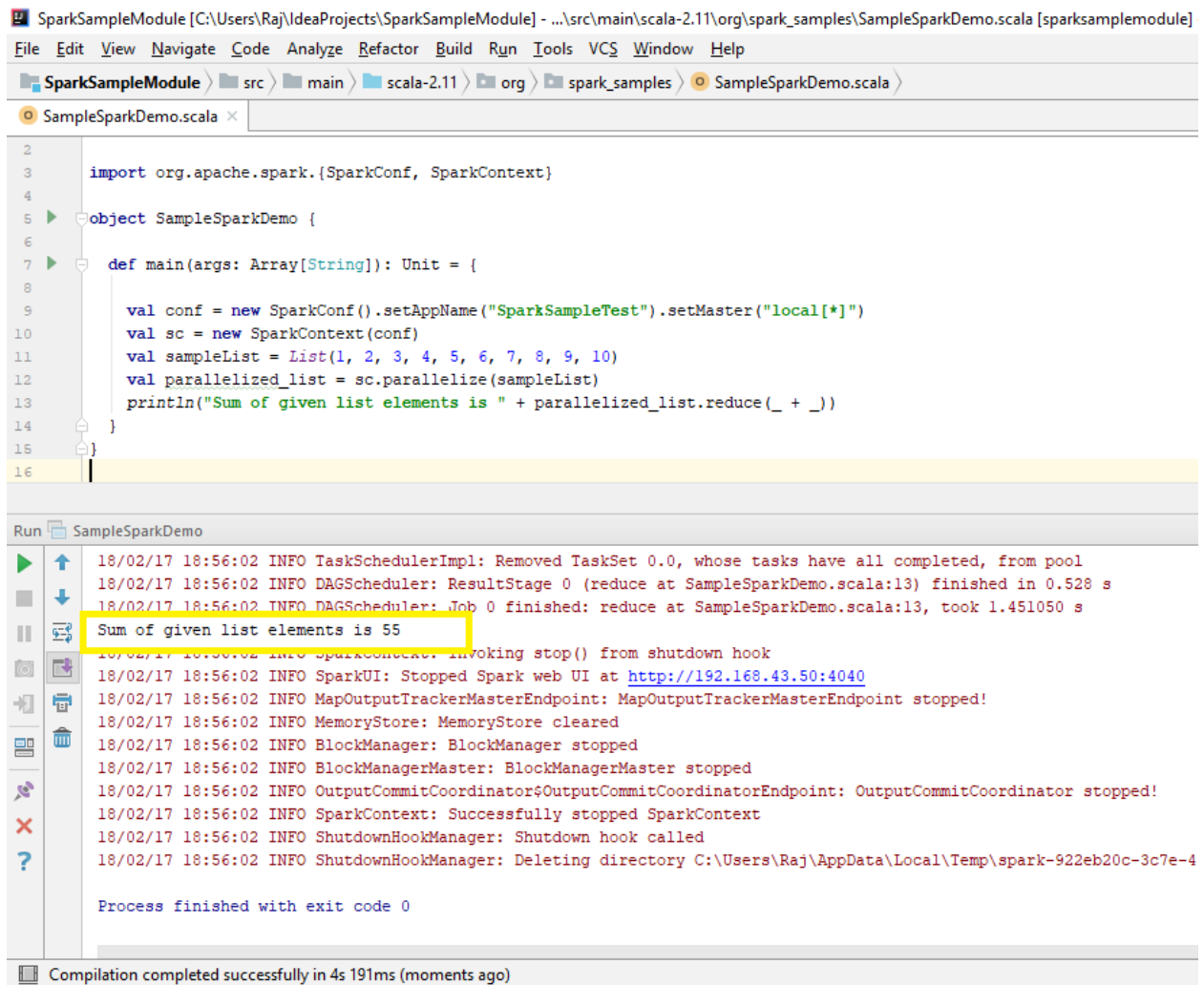
```
object SampleSparkDemo {  
  
  def main(args: Array[String]): Unit = {  
  
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")  
  
    // 'conf' value is used to configure Spark environment  
  
    val sc = new SparkContext(conf)    //'sc' is a Spark context variable created using SparkConf  
  
    val sampleList = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) //create and initialize a list with given values  
  
    val parallelized_list = sc.parallelize(sampleList) //create a parallelized RDD from sampleList  
  
  }  
}
```

1. Scala code to find the sum of all numbers:

```
println("Sum of given list elements is " + parallelized_list.reduce(_ + _))    // reduce() method  
    // is an action that takes two parameters of given RDD and applies '+' on them
```

Output:

Sum of given list elements is 55



The screenshot shows an IDE window for a project named 'SparkSampleModule'. The file 'SampleSparkDemo.scala' is open, showing the following Scala code:

```
2
3 import org.apache.spark.{SparkConf, SparkContext}
4
5 object SampleSparkDemo {
6
7   def main(args: Array[String]): Unit = {
8
9     val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")
10    val sc = new SparkContext(conf)
11    val sampleList = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
12    val parallelized_list = sc.parallelize(sampleList)
13    println("Sum of given list elements is " + parallelized_list.reduce(_ + _))
14  }
15}
16
```

The Run console shows the following output:

```
18/02/17 18:56:02 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
18/02/17 18:56:02 INFO DAGScheduler: ResultStage 0 (reduce at SampleSparkDemo.scala:13) finished in 0.528 s
18/02/17 18:56:02 INFO DAGScheduler: Job 0 finished: reduce at SampleSparkDemo.scala:13, took 1.451050 s
Sum of given list elements is 55
18/02/17 18:56:02 INFO SparkContext: Invoking stop() from shutdown hook
18/02/17 18:56:02 INFO SparkUI: Stopped Spark web UI at http://192.168.43.50:4040
18/02/17 18:56:02 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
18/02/17 18:56:02 INFO MemoryStore: MemoryStore cleared
18/02/17 18:56:02 INFO BlockManager: BlockManager stopped
18/02/17 18:56:02 INFO BlockManagerMaster: BlockManagerMaster stopped
18/02/17 18:56:02 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
18/02/17 18:56:02 INFO SparkContext: Successfully stopped SparkContext
18/02/17 18:56:02 INFO ShutdownHookManager: Shutdown hook called
18/02/17 18:56:02 INFO ShutdownHookManager: Deleting directory C:\Users\Raj\AppData\Local\Temp\spark-922eb20c-3c7e-4

Process finished with exit code 0

Compilation completed successfully in 4s 191ms (moments ago)
```

2. Scala code to find the total elements in the list:

```
println("Total number of elements in the list is " + parallelized_list.count())
```

// count() method is an action that returns count of elements in an input RDD

Output:

Total number of elements in the list is 10

The screenshot shows an IDE window for a project named 'SparkSampleModule'. The code editor displays the following Scala code:

```
1 package org.spark_samples
2
3 import org.apache.spark.{SparkConf, SparkContext}
4
5 object SampleSparkDemo {
6
7   def main(args: Array[String]): Unit = {
8
9     val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")
10    val sc = new SparkContext(conf)
11    val sampleList = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
12    val parallelized_list = sc.parallelize(sampleList)
13    println("Sum of given list elements is " + parallelized_list.reduce(_ + _))
14    println("Total number of elements in the list is " + parallelized_list.count())
15  }
16 }
```

The Run console shows the following output:

```
18/02/17 18:59:38 INFO DAGScheduler: Job 1 finished: count at SampleSparkDemo.scala:14, took 0.065771 s
Total number of elements in the list is 10
18/02/17 18:59:38 INFO SparkContext: invoking stop() from shutdown hook
18/02/17 18:59:38 INFO SparkUI: Stopped Spark web UI at http://192.168.43.50:4040
18/02/17 18:59:38 INFO BlockManagerInfo: Removed broadcast_0_piece0 on 192.168.43.50:55085 in memory (size: 940.0 B,
18/02/17 18:59:38 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
18/02/17 18:59:38 INFO MemoryStore: MemoryStore cleared
18/02/17 18:59:38 INFO BlockManager: BlockManager stopped
18/02/17 18:59:38 INFO BlockManagerMaster: BlockManagerMaster stopped
18/02/17 18:59:38 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
18/02/17 18:59:38 INFO SparkContext: Successfully stopped SparkContext
18/02/17 18:59:38 INFO ShutdownHookManager: Shutdown hook called
18/02/17 18:59:38 INFO ShutdownHookManager: Deleting directory C:\Users\Raj\AppData\Local\Temp\spark-ccd89eaa-0dff-4

Process finished with exit code 0

Compilation completed successfully in 5s 149ms (a minute ago)
```

3. Scala code to calculate the average of the numbers in the list:

```
println("Average of numbers in the list is " + parallelized_list.reduce(_ + _).toFloat /
parallelized_list.count())    //This is the calculates the sum via reduce(), gets number of elements
                               // in the RDD and finally divides the sum by number of items to get
                               // the average value as result
```

Note: The reason for including the computation inside println() statement is **to demonstrate the simplicity of coding in Spark**. To make it more understandable, we can separate each computational statement and print resulting value in the end.

Output:

Average of numbers in the list is 5.5

The screenshot shows an IDE window for a project named 'SparkSampleModule'. The file 'SampleSparkDemo.scala' is open, showing the following Scala code:

```
4  
5 object SampleSparkDemo {  
6  
7 def main(args: Array[String]): Unit = {  
8  
9     val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")  
10    val sc = new SparkContext(conf)  
11    val sampleList = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
12    val parallelized_list = sc.parallelize(sampleList).cache()  
13    println("Sum of given list elements is " + parallelized_list.reduce(_ + _))  
14    println("Total number of elements in the list is " + parallelized_list.count())  
15    println("Average of numbers in the list is " + parallelized_list.reduce(_ + _).toFloat / parallelized_list.count())  
16  }  
17 }  
18
```

The 'Run' console shows the following output:

```
18/02/17 19:38:30 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 12) in 20 ms on localhost (1/4)  
18/02/17 19:38:30 INFO TaskSetManager: Finished task 1.0 in stage 3.0 (TID 13) in 20 ms on localhost (2/4)  
18/02/17 19:38:30 INFO Executor: Finished task 2.0 in stage 3.0 (TID 14). 954 bytes result sent to driver  
18/02/17 19:38:30 INFO TaskSetManager: Finished task 3.0 in stage 3.0 (TID 15) in 18 ms on localhost (3/4)  
18/02/17 19:38:30 INFO TaskSetManager: Finished task 2.0 in stage 3.0 (TID 14) in 20 ms on localhost (4/4)  
18/02/17 19:38:30 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool  
18/02/17 19:38:30 INFO DAGScheduler: ResultStage 3 (count at SampleSparkDemo.scala:15) finished in 0.023 s  
Average of numbers in the list is 5.5  
18/02/17 19:38:30 INFO DAGScheduler: Job 3 finished: count at SampleSparkDemo.scala:15, took 0.040473 s  
18/02/17 19:38:30 INFO SparkContext: Invoking stop() from shutdown hook  
18/02/17 19:38:30 INFO SparkUI: Stopped Spark web UI at http://192.168.43.50:4040  
18/02/17 19:38:30 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!  
18/02/17 19:38:30 INFO MemoryStore: MemoryStore cleared  
18/02/17 19:38:30 INFO BlockManager: BlockManager stopped  
18/02/17 19:38:30 INFO BlockManagerMaster: BlockManagerMaster stopped  
18/02/17 19:38:30 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!  
18/02/17 19:38:30 INFO SparkContext: Successfully stopped SparkContext  
18/02/17 19:38:30 INFO ShutdownHookManager: Shutdown hook called
```

At the bottom, a status bar indicates: 'Compilation completed successfully in 3s 577ms (moments ago)'.

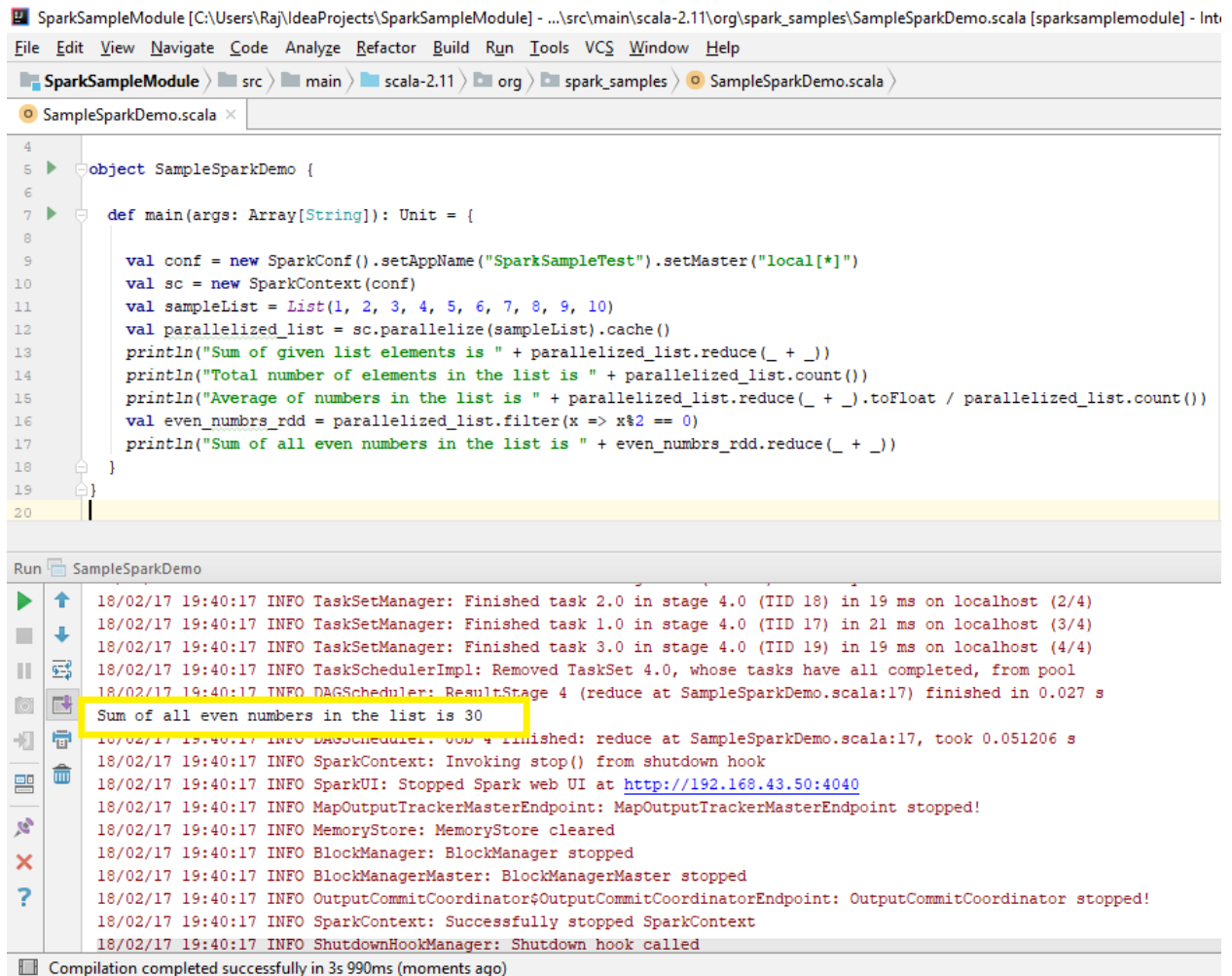
4. Scala code to find the sum of all the even numbers in the list:

```
val even_numbrs_rdd = parallelized_list.filter(x => x%2 == 0)    //return an RDD of even numbers
```

```
println("Sum of all even numbers in the list is " + even_numbrs_rdd.reduce(_ + _)) // print sum
```

Output:

Sum of all even numbers in the list is 30



The screenshot shows an IDE window for a Spark application. The top part displays the source code for `SampleSparkDemo.scala`. The code defines a `main` function that creates a `SparkConf`, a `SparkContext`, and a list of numbers from 1 to 10. It then parallelizes this list and performs several operations: printing the sum of elements, the total number of elements, the average of numbers, and the sum of even numbers. The bottom part of the screenshot shows the console output of the application. The output includes various Spark logs and a final line that states "Sum of all even numbers in the list is 30", which is highlighted with a yellow box.

```
object SampleSparkDemo {  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")  
    val sc = new SparkContext(conf)  
    val sampleList = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
    val parallelized_list = sc.parallelize(sampleList).cache()  
    println("Sum of given list elements is " + parallelized_list.reduce(_ + _))  
    println("Total number of elements in the list is " + parallelized_list.count())  
    println("Average of numbers in the list is " + parallelized_list.reduce(_ + _).toFloat / parallelized_list.count())  
    val even_nums_rdd = parallelized_list.filter(x => x%2 == 0)  
    println("Sum of all even numbers in the list is " + even_nums_rdd.reduce(_ + _))  
  }  
}
```

Run SampleSparkDemo

```
18/02/17 19:40:17 INFO TaskSetManager: Finished task 2.0 in stage 4.0 (TID 18) in 19 ms on localhost (2/4)  
18/02/17 19:40:17 INFO TaskSetManager: Finished task 1.0 in stage 4.0 (TID 17) in 21 ms on localhost (3/4)  
18/02/17 19:40:17 INFO TaskSetManager: Finished task 3.0 in stage 4.0 (TID 19) in 19 ms on localhost (4/4)  
18/02/17 19:40:17 INFO TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool  
18/02/17 19:40:17 INFO DAGScheduler: ResultStage 4 (reduce at SampleSparkDemo.scala:17) finished in 0.027 s  
Sum of all even numbers in the list is 30  
18/02/17 19:40:17 INFO DAGScheduler: Job 4 finished: reduce at SampleSparkDemo.scala:17, took 0.051206 s  
18/02/17 19:40:17 INFO SparkContext: Invoking stop() from shutdown hook  
18/02/17 19:40:17 INFO SparkUI: Stopped Spark web UI at http://192.168.43.50:4040  
18/02/17 19:40:17 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!  
18/02/17 19:40:17 INFO MemoryStore: MemoryStore cleared  
18/02/17 19:40:17 INFO BlockManager: BlockManager stopped  
18/02/17 19:40:17 INFO BlockManagerMaster: BlockManagerMaster stopped  
18/02/17 19:40:17 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!  
18/02/17 19:40:17 INFO SparkContext: Successfully stopped SparkContext  
18/02/17 19:40:17 INFO ShutdownHookManager: Shutdown hook called
```

Compilation completed successfully in 3s 990ms (moments ago)

5. Scala code to find the total number of elements in the list divisible by both 5 and 3:

```
val divisible_by_5_and_3_rdd = parallelized_list.filter(x => (x % 5 == 0) && (x % 3 == 0))
```

```
println("Total number of elements divisible by both 5 and 3 is " + divisible_by_5_and_3_rdd.count())
```

The first statement applies `filter()` function to get only those numbers that can be divisible by both 5 and 3. The next statement prints the count of elements in that RDD.

Output:

Total number of elements divisible by both 5 and 3 is 0

Note: As there are no numbers in the given list which are divisible by both 5 and 3, we got the count as zero.

The screenshot displays an IDE window for 'SparkSampleModule'. The top pane shows the source code for 'SampleSparkDemo.scala'. The code defines a 'main' function that creates a SparkConf, SparkContext, and a list of numbers (1 to 10). It then parallelizes this list and performs several operations: calculating the sum, counting elements, calculating the average, filtering even numbers, and filtering numbers divisible by both 5 and 3. The bottom pane shows the execution logs for 'SampleSparkDemo', which include task completion messages, DAG scheduling information, and SparkContext shutdown logs. The logs confirm that the job finished successfully and the SparkContext was stopped.

```
object SampleSparkDemo {  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")  
    val sc = new SparkContext(conf)  
    val sampleList = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
    val parallelized_list = sc.parallelize(sampleList).cache()  
    println("Sum of given list elements is " + parallelized_list.reduce(_ + _))  
    println("Total number of elements in the list is " + parallelized_list.count())  
    println("Average of numbers in the list is " + parallelized_list.reduce(_ + _).toFloat / parallelized_list.count())  
    val even_nums_rdd = parallelized_list.filter(x => x%2 == 0)  
    println("Sum of all even numbers in the list is " + even_nums_rdd.reduce(_ + _))  
    val divisible_by_5_and_3_rdd = parallelized_list.filter(x => (x % 5 == 0) && (x % 3 == 0))  
    println("Total number of elements divisible by both 5 and 3 is " + divisible_by_5_and_3_rdd.count())  
  }  
}
```

Run SampleSparkDemo

```
18/02/17 22:31:54 INFO TaskSetManager: Finished task 0.0 in stage 5.0 (TID 20) in 19 ms on localhost (2/4)  
18/02/17 22:31:54 INFO TaskSetManager: Finished task 2.0 in stage 5.0 (TID 22) in 19 ms on localhost (3/4)  
18/02/17 22:31:54 INFO TaskSetManager: Finished task 1.0 in stage 5.0 (TID 21) in 26 ms on localhost (4/4)  
18/02/17 22:31:54 INFO TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all completed, from pool  
Total number of elements divisible by both 5 and 3 is 0  
18/02/17 22:31:54 INFO DAGScheduler: ResultStage 5 (count at SampleSparkDemo.scala:19) finished in 0.030 s  
18/02/17 22:31:54 INFO DAGScheduler: Job 5 finished: count at SampleSparkDemo.scala:19, took 0.050415 s  
18/02/17 22:31:54 INFO SparkContext: Invoking stop() from shutdown hook  
18/02/17 22:31:54 INFO SparkUI: Stopped Spark web UI at http://192.168.43.50:4040  
18/02/17 22:31:54 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!  
18/02/17 22:31:54 INFO MemoryStore: MemoryStore cleared  
18/02/17 22:31:54 INFO BlockManager: BlockManager stopped  
18/02/17 22:31:54 INFO BlockManagerMaster: BlockManagerMaster stopped  
18/02/17 22:31:54 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!  
18/02/17 22:31:54 INFO SparkContext: Successfully stopped SparkContext  
Compilation completed successfully in 7s 239ms (a minute ago)
```

P. S.: If you have noticed the code in few of the screenshots above, I made use of `cache()` function while creating a parallelized collection. This function is useful when you want to perform the same operation repeatedly on an RDD. For example, we have applied `count()` on 'parallelized_list' RDD couple of times. In case we don't use `cache()` on it, the count operation will read contents from source each time, whereas in the application of `cache()`, first time when `count()` is used, the data will be read from this RDD, cached and then counted. Second time when `count()` is used, the data will be read from cache and returns the count almost immediately. This function is recommended to be used while reading large amount of data from source so as to reduce data read time.