JAVA EXP – 12

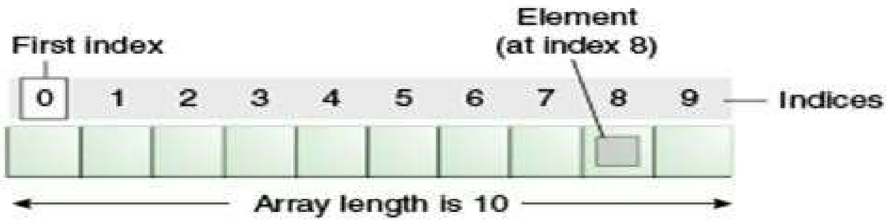
NAME: DIVYESH KHUNT **SAPID:60009210116** **BATCH:D12**

Aim:-Implement Program to demonstrate Array, List, Sets, Tuple, Maps, iterator and String in Scala

Theory: -

1. **Array in Scala**

Array is a collection of mutable values. It is an index based data structure which starts from 0 index to n-1 where n is length of array.
Scala arrays can be generic. It means, you can have an Array[T], where T is a type parameter.
Following image represents the structure of array where first index is 0, last index is 9 and array length is 10.



Scala Types of array

1. Single dimensional array
2. Multidimensional array

Scala Single Dimensional Array
Single dimensional array is used to store elements in linear order. Array elements are stored in contiguous memory space. So, if you have any index of an array, you can easily traverse all the elements of the array.

Syntax for Single Dimensional Array

```
var arrayName: Array[arrayType] = new Array[arrayType](arraySize); or
var arrayName = new Array[arrayType](arraySize) or
var arrayName: Array[arrayType] = new Array(arraySize); or
var arrayName = Array (element1, element2 ... elementN)
```

Scala Array Example: Single Dimensional

```
class ArrayExample {
  var arr = Array (1,2,3,4,5)      // Creating single dimensional array
  def show (){
    for(a<-arr)                    // Traversing array elements
      println(a)
    println("Third Element  = "+ arr(2))       // Accessing elements by using index
  }
}

object MainObject{
    def main(args:Array[String]){
        var a = new ArrayExample()
        a.show()
    }
}
```

Scala Example 2: Single Dimensional

In this example, we have created an array by using new keyword which is used to initialize memory for array. The entire array elements are set to default value, you can assign that later in your code.

```
import scala.io.StdIn._
class ArrayExample {
    print ("Enter Size of Array")
    var size=readInt()
    var arr = new Array[Int](size)        // Creating single dimensional array
    def show(){
        for(a<-arr){                      // Traversing array elements
            println(a)
        }
        println("Third Element before assignment = "+ arr(2))// Accessing elements by using
index
        arr(2) = 10 // Assigning new element at 2 index
        println("Third Element after assignment = "+ arr(2))
    }
}

object MainObject{
    def main(args:Array[String]){
        var a = new ArrayExample()
        a.show()
    }
}
```

```
Output
Enter Size of Array5
0
0
0
0
0
Third Element before assignment = 0
Third Element after assignment = 10
```

Scala Passing Array into Function

You can pass array as an argument to function during function call. Following example illustrate the process how we can pass an array to the function.

```
class ArrayExample {
    def show(arr: Array[Int]){
        for(a<-arr)                // Traversing array elements
            println(a)
        println("Third Element = "+ arr(2)) // Accessing elements by using index
    }
}

object MainObject{
    def main(args:Array[String]){
        var arr = Array(1,2,3,4,5,6)    // creating single dimensional array
        var a = new ArrayExample()
        a.show(arr)                     // passing array as an argument in the function
    }
}
```

Output: -

```
1
2
3
4
5
6
Third Element = 3
```

Scala Array Example: Iterating by using Foreach Loop

You can also iterate array elements by using foreach loop. Let's see an example.

```
class ArrayExample {
    var arr = Array (1,2,3,4,5)      // Creating single dimensional array
    arr. foreach ((element: Int) =>println(element))      // Iterating by using foreach loop
}

object MainObject {
    def main (args: Array[String]){
        new ArrayExample ()
    }
}
```

Output

```
1
2
3
4
5
```

Scala Multidimensional Array

Multidimensional array is an array which store data in matrix form. You can create from two dimensional to three, four and many more dimensional array according to your need. Below we have mentioned array syntax. Scala provides an ofDim method to create Multidimensional array.

Multidimensional Array Syntax

var arrayName = Array.ofDim[ArrayType](NoOfRows, NoOfColumns) or

var arrayName = Array(Array(element...), Array(element...), ...)

Scala Multidimensional Array Example by using ofDim

In This example, we have created array by using ofDim method.

Experiment No 12

```
class ArrayExample {
  var arr = Array.ofDim[Int](2,2)         // Creating multidimensional array
  arr(1)(0) = 15                          // Assigning value
  def show(){
    for(i<- 0 to 1){                      // Traversing elements by using loop
      for(j<- 0 to 1){
        print(" "+arr(i)(j))
      }
      println()
    }
    println("Third Element = "+ arr(1)(1))        // Accessing elements by using index
  }
}
object MainObject{
  def main(args:Array[String]){
    var a = new ArrayExample()
    a.show()
  }
}
```

Output

```
0 0
15 0
Third Element = 0
```

Scala Multidimensional Array by using Array of Array

Apart from ofDim you can also create multidimensional array by using array of array. In this example, we have created multidimensional array by using array of array.

```
class ArrayExample{
  var arr = Array(Array(1,2,3,4,5), Array(6,7,8,9,10))  // Creating multidimensional array
  def show(){
    for(i<- 0 to 1){              // Traversing elements using loop
      for(j<- 0 to 4){
        print(" "+arr(i)(j))
      }
      println()
    }
  }
}

object MainObject{
  def main(args:Array[String]){
    var a = new ArrayExample()
    a.show()
  }
}
```

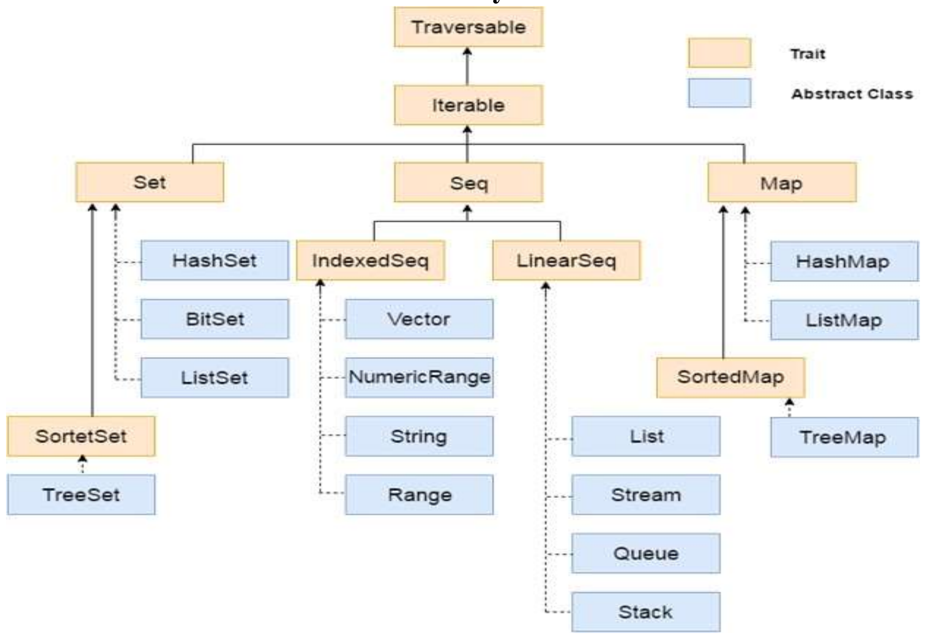
Experiment No 12

Scala Collection

Scala provides rich set of collection library. It contains classes and traits to collect data. These collections can be mutable or immutable. You can use them according to your requirement. **Scala.collection.mutable** package contains all the mutable collections. You can add, remove and update data while using this package.

Scala.collection.immutable contains all the immutable collections. It does not allow you to modify data. Scala imports this package by default. If you want mutable collection, you must import **scala.collection.mutable** package in your code.

Scala Immutable Collections Hierarchy



Scala Traversable

It is a trait and used to traverse collection elements. It is a base trait for all Scala collections.

Some Significant Methods of Traversable Trait

Method	**Description**
def head: A	It returns the first element of collection.
def init: Traversable[A]	It returns all elements except last one.
def isEmpty: Boolean	It checks whether the collection is empty or not. It returns either true or false.
def last: A	It returns the last element of this collection.

def max: A	It returns the largest element of this collection.
def min: A	It returns smallest element of this collection
def size: Int	It is used to get size of this traversable and returns a number of elements present in this traversable.
def sum: A	It returns sum of all elements of this collection.
def tail: Traversable[A]	It returns all elements except first.
def toArray: Array[A]	It converts this collection to an array.
def toList: List[A]	It converts this collection to a list.
def toSeq: Seq[A]	It converts this collection to a sequence.
def toSet[B >: A]: immutable.Set[B]	It converts this collection to a set.

Scala Iterable
It is a next trait from the top of the hierarchy and a base trait for iterable collections. It extends traversable trait and provides important methods to concrete classes.

Scala List
List is used to store ordered elements. It is a class for immutable linked lists.It maintains order of elements and can contain duplicates elements also.
Scala List Example
In this example, we have created two lists. Here, both lists have different syntax to create list.

```
import scala.collection.immutable._
object MainObject{
   def main(args:Array[String]){
      var list = List(1,8,5,6,9,58,23,15,4)
       var list2:List[Int] = List(1,8,5,6,9,58,23,15,4)
       println(list)
       println(list2)
   }
}
```

Scala List Example: Applying Predefined Methods

```
import scala.collection.immutable._
object MainObject{
   def main(args:Array[String]){
       var list = List(1,8,5,6,9,58,23,15,4)
       var list2 = List(88,100)
       print("Elements: ")
       list.foreach((element:Int) => print(element+" "))        // Iterating using foreach loop
       print("\nElement at 2 index: "+list(2))             // Accessing element of 2 index
```

```
    var list3 = list ++ list2                                  // Merging two list
    print("\nElement after merging list and list2: ")
    list3.foreach((element:Int)=>print(element+" "))
    var list4 = list3.sorted                                 // Sorting list
    print("\nElement after sorting list3: ")
    list4.foreach((element:Int)=>print(element+" "))
    var list5 = list3.reverse                                // Reversing list elements
    print("\nElements in reverse order of list5: ")
    list5.foreach((element:Int)=>print(element+" "))

  }
}
```

Output:

```
Elements: 1 8 5 6 9 58 23 15 4
Element at 2 index: 5
Element after merging list and list2: 1 8 5 6 9 58 23 15 4 88 100
Element after sorting list3: 1 4 5 6 8 9 15 23 58 88 100
Elements in reverse order of list5: 100 88 4 15 23 58 9 6 5 8 1
```

Scala Set
It is used to store unique elements in the set. It does not maintain any order for storing elements. You can apply various operations on them. It is defined in the Scala.collection.immutable package.

Scala Set Syntax
val variableName:Set[Type] = Set(element1, element2,... elementN) or
val variableName = Set(element1, element2,... elementN)

Scala Set Example
In this example, we have created a set. You can create an empty set also. Let's see how to create a set.

```
import scala.collection.immutable._
object MainObject{
  def main(args:Array[String]){
    val set1 = Set()                          // An empty set
    val games = Set("Cricket","Football","Hocky","Golf")    // Creating a set with elements
    println(set1)
    println(games)
  }
}
```

Output:

```
Set()    // an empty set
Set(Cricket,Football,Hocky,Golf)
```

Experiment No 12

Scala Set Example 2

In Scala, Set provides some predefined properties to get information about set. You can get first or last element of Set and many more. Let's see an example.

```
import scala.collection.immutable._
object MainObject{
    def main(args:Array[String]){
      val games = Set("Cricket","Football","Hocky","Golf")
      println(games.head)          // Returns first element present in the set
      println(games.tail)        // Returns all elements except first element.
      println(games.isEmpty)         // Returns either true or false
    }
  }
```

Scala Set Example: Merge two Set

You can merge two sets into a single set. Scala provides a predefined method to merge sets. In this example, ++ method is used to merge two sets.

```
import scala.collection.immutable._
object MainObject{
    def main(args:Array[String]){
      val games = Set("Cricket","Football","Hocky","Golf")
      val alphabet = Set("A","B","C","D","E")
      val mergeSet = games ++ alphabet          // Merging two sets
      println("Elements in games set: "+games.size)  // Return size of collection
      println("Elements in alphabet set: "+alphabet.size)
      println("Elements in mergeSet: "+mergeSet.size)
      println(mergeSet)
    }
}
```

Scala Set Example 2

You can check whether element is present in the set or not. The following example describe the use of contains() method.

```
import scala.collection.immutable._
object MainObject{
    def main(args:Array[String]){
      val games = Set("Cricket","Football","Hocky","Golf")
      println(games)
      println("Elements in set: "+games.size)
      println("Golf exists in the set : "+games.contains("Golf"))
      println("Racing exists in the set : "+games.contains("Racing"))

    }
  }
```

Experiment No 12

Scala Set Example: Adding and Removing Elements

You can add or remove elements from the set. You can add only when your code is mutable. In this example, we are adding and removing elements of the set.

```
import scala.collection.immutable._
object MainObject{
        def main(args:Array[String]){
            var games = Set("Cricket","Football","Hocky","Golf")
            println(games)
            games += "Racing"              // Adding new element
            println(games)
            games += "Cricket"             // Adding new element, it does not allow duplicacy.
            println(games)
            games -= "Golf"            // Removing element
            println(games)
        }
    }
```

Scala Set Example: Iterating Set Elements using for loop

You can iterate set elements either by using for loop or foreach loop. You can also filter elements during iteration. In this example have used for loop to iterate set elements.

```
import scala.collection.immutable._
object MainObject{
        def main(args:Array[String]){
            var games = Set("Cricket","Football","Hocky","Golf")
            for(game <- games){
                println(game)
            }
        }
    }
```

Scala Set Example: Set Operations

In scala Set, you can also use typical math operations like: intersection and union. In the following example we have used predefined methods to perform set operations.

```
import scala.collection.immutable._
object MainObject{
    def main(args:Array[String]){
        var games = Set("Cricket","Football","Hocky","Golf","C")
        var alphabet = Set("A","B","C","D","E","Golf")
        var setIntersection = games.intersect(alphabet)
        println("Intersection by using intersect method: "+setIntersection)
        println("Intersection by using & operator: "+(games & alphabet))
        var setUnion = games.union(alphabet)
        println(setUnion)
    }
}
```

Scala SortedSet

Experiment No 12

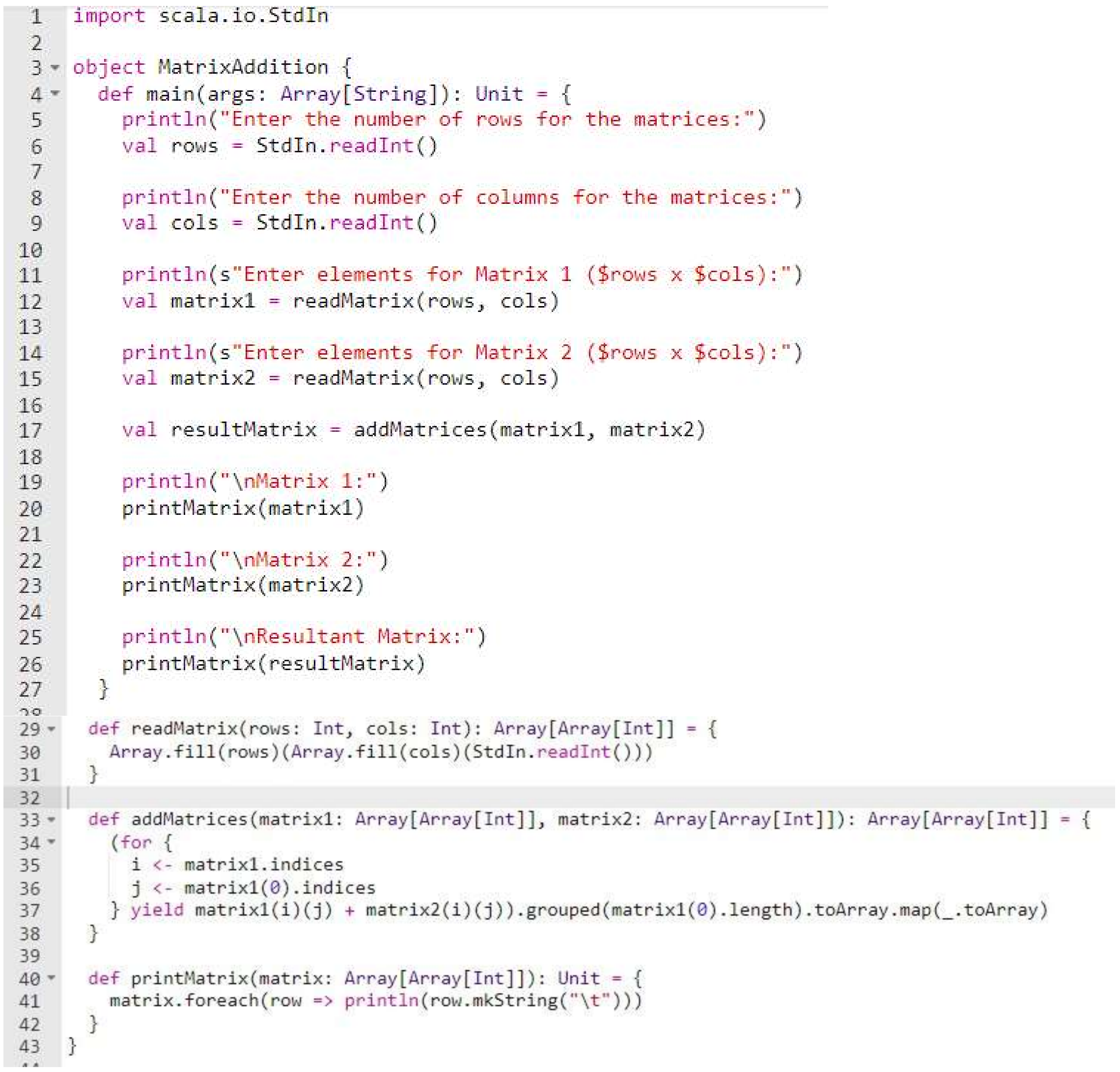
In scala, SortedSet extends Set trait and provides sorted set elements. It is useful when you want sorted elements in the Set collection. You can sort integer values and string as well.
It is a trait and you can apply all the methods defined in the traversable trait and Set trait.
Scala SortedSet Example
In the following example, we have used SortedSet to store integer elements. It returns a Set after sorting elements.

```
import scala.collection.immutable.SortedSet
object MainObject{
  def main(args:Array[String]){
    var numbers: SortedSet[Int] = SortedSet(5,8,1,2,9,6,4,7,2)
    numbers.foreach((element:Int)=> println(element))
  }
}
```

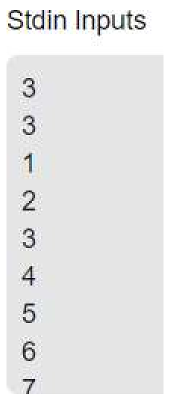
Lab Exercise to be performed in this session

1. Write a Scala Program to perform Addition of two matrix example

Experiment No 12



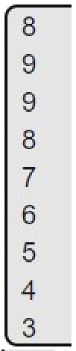
```scala
import scala.io.StdIn

object MatrixAddition {
  def main(args: Array[String]): Unit = {
    println("Enter the number of rows for the matrices:")
    val rows = StdIn.readInt()

    println("Enter the number of columns for the matrices:")
    val cols = StdIn.readInt()

    println(s"Enter elements for Matrix 1 ($rows x $cols):")
    val matrix1 = readMatrix(rows, cols)

    println(s"Enter elements for Matrix 2 ($rows x $cols):")
    val matrix2 = readMatrix(rows, cols)

    val resultMatrix = addMatrices(matrix1, matrix2)

    println("\nMatrix 1:")
    printMatrix(matrix1)

    println("\nMatrix 2:")
    printMatrix(matrix2)

    println("\nResultant Matrix:")
    printMatrix(resultMatrix)
  }

  def readMatrix(rows: Int, cols: Int): Array[Array[Int]] = {
    Array.fill(rows)(Array.fill(cols)(StdIn.readInt()))
  }

  def addMatrices(matrix1: Array[Array[Int]], matrix2: Array[Array[Int]]): Array[Array[Int]] = {
    (for {
      i <- matrix1.indices
      j <- matrix1(0).indices
    } yield matrix1(i)(j) + matrix2(i)(j)).grouped(matrix1(0).length).toArray.map(_.toArray)
  }

  def printMatrix(matrix: Array[Array[Int]]): Unit = {
    matrix.foreach(row => println(row.mkString("\t")))
  }
}
```



Stdin Inputs

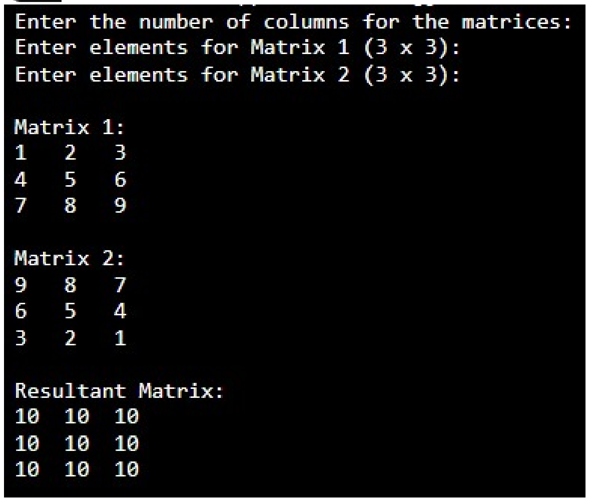
```
3
3
1
2
3
4
5
6
7
```

Experiment No 12



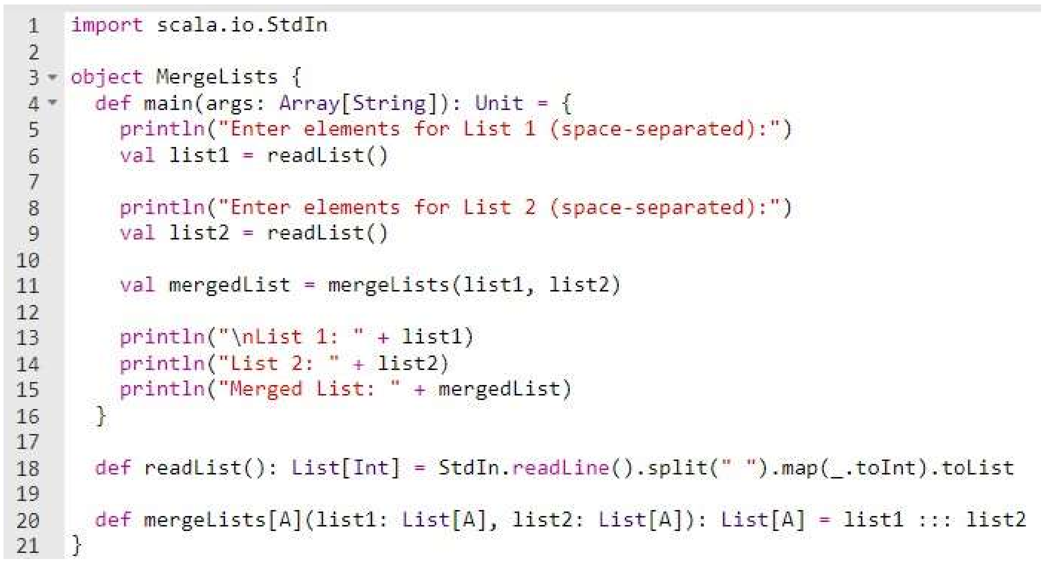
```
8
9
9
8
7
6
5
4
3
```

```
2
1
```



```
Enter the number of columns for the matrices:
Enter elements for Matrix 1 (3 x 3):
Enter elements for Matrix 2 (3 x 3):

Matrix 1:
1   2   3
4   5   6
7   8   9

Matrix 2:
9   8   7
6   5   4
3   2   1

Resultant Matrix:
10  10  10
10  10  10
10  10  10
```

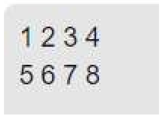
2. Write Scala Program to merge to list



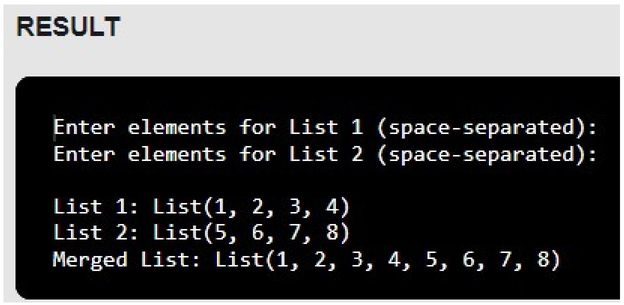
```scala
import scala.io.StdIn

object MergeLists {
  def main(args: Array[String]): Unit = {
    println("Enter elements for List 1 (space-separated):")
    val list1 = readList()

    println("Enter elements for List 2 (space-separated):")
    val list2 = readList()

    val mergedList = mergeLists(list1, list2)

    println("\nList 1: " + list1)
    println("List 2: " + list2)
    println("Merged List: " + mergedList)
  }

  def readList(): List[Int] = StdIn.readLine().split(" ").map(_.toInt).toList

  def mergeLists[A](list1: List[A], list2: List[A]): List[A] = list1 ::: list2
}
```

Experiment No 12

Stdin Inputs



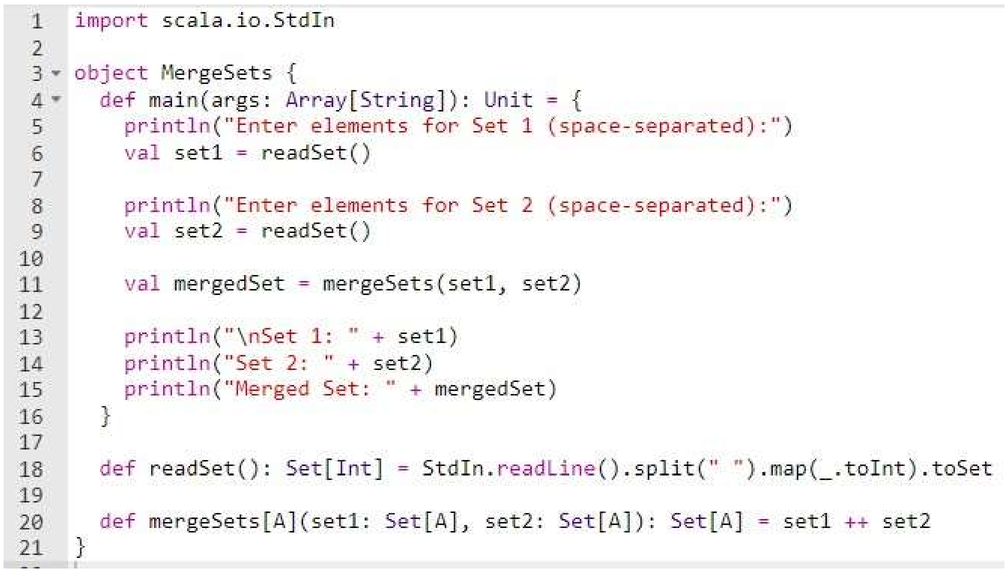
```
1 2 3 4
5 6 7 8
```



RESULT

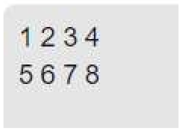
```
Enter elements for List 1 (space-separated):
Enter elements for List 2 (space-separated):

List 1: List(1, 2, 3, 4)
List 2: List(5, 6, 7, 8)
Merged List: List(1, 2, 3, 4, 5, 6, 7, 8)
```

3. Write a Scala Program to merge to sets

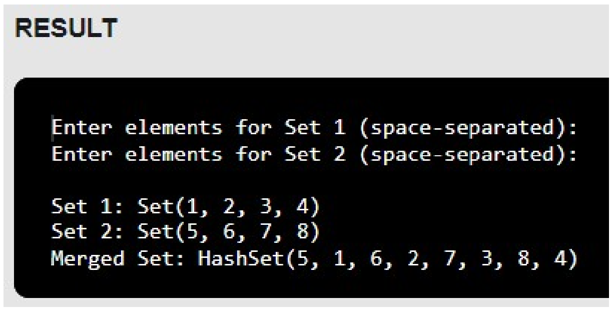


```scala
import scala.io.StdIn

object MergeSets {
  def main(args: Array[String]): Unit = {
    println("Enter elements for Set 1 (space-separated):")
    val set1 = readSet()

    println("Enter elements for Set 2 (space-separated):")
    val set2 = readSet()

    val mergedSet = mergeSets(set1, set2)

    println("\nSet 1: " + set1)
    println("Set 2: " + set2)
    println("Merged Set: " + mergedSet)
  }

  def readSet(): Set[Int] = StdIn.readLine().split(" ").map(_.toInt).toSet

  def mergeSets[A](set1: Set[A], set2: Set[A]): Set[A] = set1 ++ set2
}
```

Stdin Inputs



```
1 2 3 4
5 6 7 8
```



RESULT

```
Enter elements for Set 1 (space-separated):
Enter elements for Set 2 (space-separated):

Set 1: Set(1, 2, 3, 4)
Set 2: Set(5, 6, 7, 8)
Merged Set: HashSet(5, 1, 6, 2, 7, 3, 8, 4)
```