SCALA SETS

Scala Set is a collection of pairwise different elements of the same type. In other words, a Set is a collection that contains no duplicate elements. There are two kinds of Sets, the **immutable** and the **mutable**. The difference between mutable and immutable objects is that when an object is immutable, the object itself can't be changed.

By default, Scala uses the immutable Set. If you want to use the mutable Set, you'll have to import **scala.collection.mutable.Set** class explicitly. If you want to use both mutable and immutable sets in the same, then you can continue to refer to the immutable Set as **Set** but you can refer to the mutable Set as **mutable.Set**. Following is the example to declare immutable Sets as follows:

```
// Empty set of integer type
var s : Set[Int] = Set()

// Set of integer type
var s : Set[Int] = Set(1,3,5,7)

or

var s = Set(1,3,5,7)
```

While defining empty set, the type annotation is necessary as the system needs to assign a concrete type to variable.

Basic Operations on Set:

All operations on sets can be expressed in terms of the following three methods:

Methods	Description
head	This method returns the first element of a set.
tail	This method returns a set consisting of all elements except the first.
isEmpty	This method returns true if the set is empty otherwise false.

Following is the example showing usage of the above methods:

```
object Test {
   def main(args: Array[String]) {
      val fruit = Set("apples", "oranges", "pears")
      val nums: Set[Int] = Set()

      println( "Head of fruit : " + fruit.head )
      println( "Tail of fruit : " + fruit.tail )
      println( "Check if fruit is empty : " + fruit.isEmpty )
      println( "Check if nums is empty : " + nums.isEmpty )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Head of fruit : apples
Tail of fruit : Set(oranges, pears)
Check if fruit is empty : false
Check if nums is empty : true
```

```
C:/>
```

Concatenating Sets:

You can use either **++** operator or **Set.++** method to concatenate two or more sets, but while adding sets it will remove duplicate elements. Following is the example to concatenate two sets:

```
object Test {
   def main(args: Array[String]) {
      val fruit1 = Set("apples", "oranges", "pears")
      val fruit2 = Set("mangoes", "banana")

      // use two or more sets with ++ as operator
      var fruit = fruit1 ++ fruit2
      println( "fruit1 ++ fruit2 : " + fruit )

      // use two sets with ++ as method
      fruit = fruit1.++(fruit2)
      println( "fruit1.++(fruit2) : " + fruit )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
fruit1 ++ fruit2 : Set(banana, apples, mangoes, pears, oranges)
fruit1.++(fruit2) : Set(banana, apples, mangoes, pears, oranges)

C:/>
```

Find max, min elements in Set:

You can use **Set.min** method to find out the minimum and **Set.max** method to find out the maximum of the elements available in a set. Following is the example to show the usage:

```
object Test {
   def main(args: Array[String]) {
      val num = Set(5,6,9,20,30,45)

      // find min and max of the elements
      println( "Min element in Set(5,6,9,20,30,45) : " + num.min )
      println( "Max element in Set(5,6,9,20,30,45) : " + num.max )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Min element in Set(5,6,9,20,30,45) : 5
Max element in Set(5,6,9,20,30,45) : 45

C:/>
```

Find common values in Sets:

You can use either **Set.&** method or **Set.intersect** method to find out the common values between two sets. Following is the example to show the usage:

```
object Test {
   def main(args: Array[String]) {
      val num1 = Set(5,6,9,20,30,45)
      val num2 = Set(50,60,9,20,35,55)
```

```
      // find common elements between two sets
      println( "num1.&(num2) : " + num1.&(num2) )
      println( "num1.intersect(num2) : " + num1.intersect(num2) )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
num1.&(num2) : Set(20, 9)
num1.intersect(num2) : Set(20, 9)

C:/>
```

Scala Set Methods:

Following are the important methods which you can use while playing with Sets. For a complete list of methods available, please check official documentation of Scala.

SN	Methods with Description
1	**def +*elem*: *A*: Set[A]** Creates a new set with an additional element, unless the element is already present.
2	**def -*elem*: *A*: Set[A]** Creates a new set with a given element removed from this set.
3	**def contains*elem*: *A*: Boolean** Returns true if elem is contained in this set, false otherwise.
4	**def &*that*: *Set*[*A*]: Set[A]** Returns a new set consisting of all elements that are both in this set and in the given set that.
5	**def &~*that*: *Set*[*A*]: Set[A]** Returns the difference of this set and another set.
6	**def +*elem*1: *A*, *elem*2: *A*, *elems*: *A* ∗ : Set[A]** Creates a new immutable set with additional elements from the passed sets
7	**def ++*elems*: *A*: Set[A]** Concatenates this immutable set with the elements of another collection to this immutable set.
8	

def -*elem1*: *A*, *elem2*: *A*, *elems*: *A* * : Set[A]

Returns a new immutable set that contains all elements of the current immutable set except one less occurrence of each of the given argument elements.

9

def addString*b*: *StringBuilder*: StringBuilder

Appends all elements of this immutable set to a string builder.

10

def addString*b*: *StringBuilder*, *sep*: *String*: StringBuilder

Appends all elements of this immutable set to a string builder using a separator string.

11

def apply*elem*: *A*

Tests if some element is contained in this set.

12

def count*p*: (*A* => Boolean): Int

Counts the number of elements in the immutable set which satisfy a predicate.

13

def copyToArray*xs*: *Array*[*A*], *start*: *Int*, *len*: *Int*: Unit

Copies elements of this immutable set to an array.

14

def diff*that*: *Set*[*A*]: Set[A]

Computes the difference of this set and another set.

15

def drop*n*: *Int*: Set[A]]

Returns all elements except first n ones.

16

def dropRight*n*: *Int*: Set[A]

Returns all elements except last n ones.

17

def dropWhile*p*: (*A* => Boolean): Set[A]

Drops longest prefix of elements that satisfy a predicate.

18

def equals*that*: *Any*: Boolean

The equals method for arbitrary sequences. Compares this sequence to some other object.

19

def exists*p*: (*A* => Boolean): Boolean

Tests whether a predicate holds for some of the elements of this immutable set.

20

def filter$p$**: (**A **=> Boolean): Set[A]**

Returns all elements of this immutable set which satisfy a predicate.

21

def find$p$**: (**A **=> Boolean): Option[A]**

Finds the first element of the immutable set satisfying a predicate, if any.

22

def forall$p$**: (**A **=> Boolean): Boolean**

Tests whether a predicate holds for all elements of this immutable set.

23

def foreach$f$**: (**A **=> Unit): Unit**

Applies a function f to all elements of this immutable set.

24

def head: A

Returns the first element of this immutable set.

25

def init: Set[A]

Returns all elements except the last.

26

def intersect$that$**:** $Set[A]$**: Set[A]**

Computes the intersection between this set and another set.

27

def isEmpty: Boolean

Tests if this set is empty.

28

def iterator: Iterator[A]

Creates a new iterator over all elements contained in the iterable object.

29

def last: A

Returns the last element.

30

def map[B]f**: (**A **=> B): immutable.Set[B]**

Builds a new collection by applying a function to all elements of this immutable set.

31

def max: A

Finds the largest element.

32

def min: A

Finds the smallest element.

33

def mkString: String

Displays all elements of this immutable set in a string.

34

def mkString*sep*: *String*: String

Displays all elements of this immutable set in a string using a separator string.

35

def product: A

Returns the product of all elements of this immutable set with respect to the * operator in num.

36

def size: Int

Returns the number of elements in this immutable set.

37

def splitAt*n*: *Int*: *Set*[*A*], *Set*[*A*]

Returns a pair of immutable sets consisting of the first n elements of this immutable set, and the other elements.

38

def subsetOf*that*: *Set*[*A*]: Boolean

Returns true if this set is a subset of that, i.e. if every element of this set is also an element of that.

39

def sum: A

Returns the sum of all elements of this immutable set with respect to the + operator in num.

40

def tail: Set[A]

Returns a immutable set consisting of all elements of this immutable set except the first one.

41

def take*n*: *Int*: Set[A]

Returns first n elements.

42

def takeRight*n*: *Int*:Set[A]

Returns last n elements.

43

def toArray: Array[A]

Returns an array containing all elements of this immutable set.

44

def toBuffer[B >: A]: Buffer[B]

Returns a buffer containing all elements of this immutable set.

45

def toList: List[A]

Returns a list containing all elements of this immutable set.

46

def toMap[T, U]: Map[T, U]

Converts this immutable set to a map

47

def toSeq: Seq[A]

Returns a seq containing all elements of this immutable set.

48

def toString: String

Returns a String representation of the object.