An Introduction to the Collection Library

In the following lesson, you will be introduced to Scala's collection library.

We'll cover the following

- Introduction
- Mutable Collections
- Immutable Collections
- Sequences, Sets, and Maps
  - Sequences
  - Sets
  - Maps
  - The apply Method
    - Seq
    - Set
  - Map
- Sequences

Introduction

You can think of collections as vessels used for collecting data and Scala has a vast library of them. Scala's collection library is made up of classes and traits which provide you a plethora of built-in data structures and methods for collecting and manipulating data.

> Traits is a topic for another course. For the scope of this course, you can safely assume that a trait is simply a class with its own members. An instance of a trait is identical to saying an instance of a class. Hence, we will be omitting the use of *trait* from this point onwards and just use the term *class* for both.

The focus of this chapter will be to simply introduce Scala's collection library and focus on the syntax and implementation of the well-known/most used collections.

Each Scala collection is one of two types: **mutable** collection or **immutable** collection.

Mutable Collections

Mutable collections are collections which can be updated. Elements can be added to the collection and can be removed or manipulated. In this case, the collection itself will be getting modified.

Immutable Collections

Immutable collections cannot be updated. When you add, remove, or manipulate an element in an immutable collection, you are creating a new collection and leaving the old one unchanged.

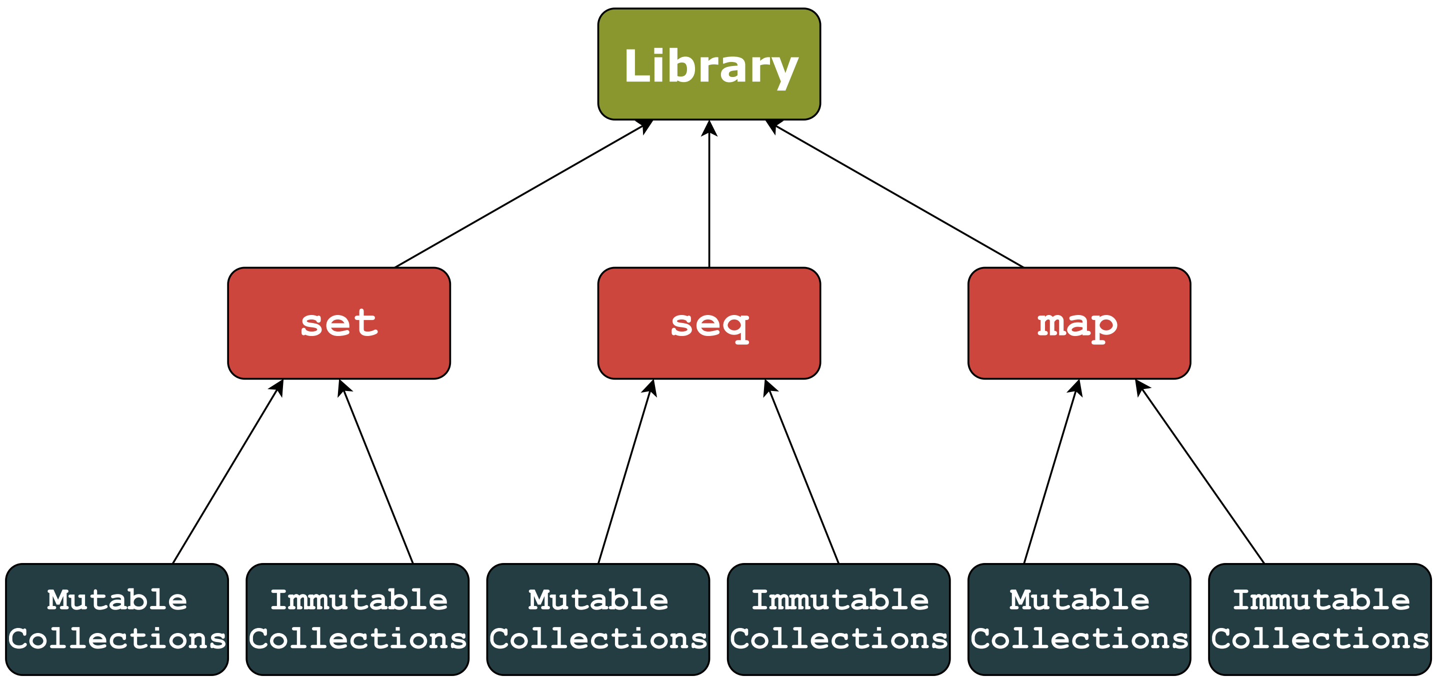
Sequences, Sets, and Maps

The collection library takes on a hierarchical structure. At the top of the library, there are three main categories of collection classes under which different collections lie:

- Sequences - `Seq`
- Sets - `Set`
- Maps - `Map`

All three classes contain both mutable and immutable collections.

> Remember, `Seq` here would be acting as a blueprint with which you can create objects that represent *sequences*.



Sequences

Collections which are part of the `Seq` class, store elements at fixed index positions, with the index starting at **0**. Each element has a specified location in the sequence and therefore, can be located very easily.

```
Seq(2,4,6,8,10)
```

Sets

Collections which are the `Set` class contain sets of elements with no element existing more than once, i.e, no duplicates.

```
Set("apple","orange","banana","grape")
```

Maps

Collections which are of the `Map` class consist of pairs of keys and values with each value being associated with a unique key.

```
(key,value)
```

```
Map((“a”,25) (“b”,50) (“c”,75))
```

The `apply` Method

To better understand the difference between each of the three types of collections, let’s look at how each collection implements the `apply` method. `apply` is a method which has a single parameter and is available to all the collections in the collection library.

`Seq`

For sequence collections, the argument passed to `apply` specifies an index. `apply` returns the element at the specified index.

> In Scala, indexing starts at **0**.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
val seqCollection = Seq(2,4,6,8)

val result = seqCollection.apply(1)

// Driver Code
print(result)
```

`Set`

For set collections, the argument passed to `apply` is an element in the specified collection. `apply` returns `true` if the element is in the specified collection and `false` if it isn’t.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
val setCollection = Set("apple", "orange", "banana", "grape")

val result = setCollection.apply("orange")

// Driver Code
print(result)
```

Map

For map collections, the argument passed to `apply` specifies a key. `apply` returns the value of the specified key.

This code requires the following environment variables to execute:

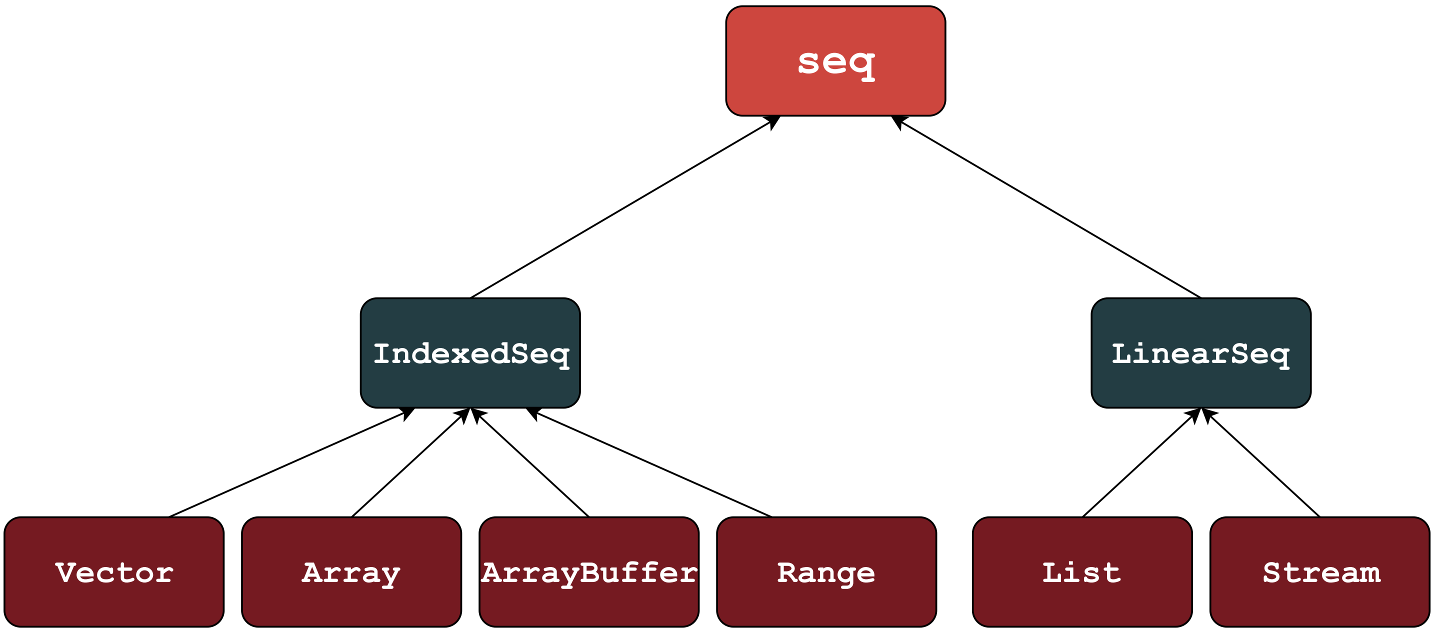
LANG C.UTF-8

```
val mapCollection = Map(("a",25),("b",50),("c",75))

val result = mapCollection.apply("c")

// Driver Code
print(result)
```

Sequence collections are the more commonly used collections and will be the focus of this chapter.

Sequences

The sequence class is further divided into two classes `IndexedSeq` and `LinearSeq`.



Collections we will be covering in this course

`IndexSeq` and `LinearSeq` do not add new operations to the `Seq` class, but each offers different performance characteristics. Collections of the `LinearSeq` class have efficient `head` and `tail` operations. What this means is that the collection is structured in such a way that it is not computationally difficult to access the head (first element) of the collection or the tail (last element) of the collection. Furthermore, `IndexedSeq` collections have efficient `apply` and `length` operations.

Before we move on to the actual collection, we will learn about the `foreach` method in the next lesson.