Using Set

We'll cover the following

- Creating sets in Kotlin

Creating sets in Kotlin

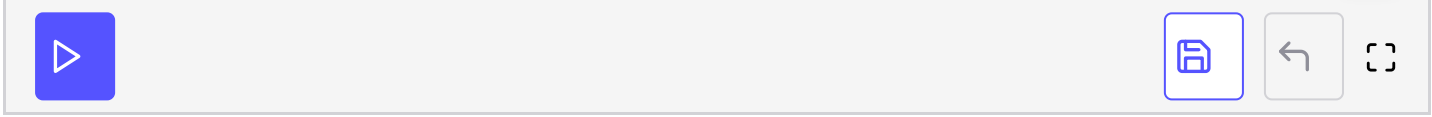
Sets are unordered collections of elements. Like the methods for creating `List<T>`, which has both immutable/read-only and mutable/read-write versions, you may create instances of `Set<T>` using `setOf()` or instances of `MutableSet<T>` using `mutableSetOf()`. You may also use `hashSetOf()` to get a reference of type `java.util.HashSet<T>`: `linkedSetOf()` for `LinkedHashSet`, and `sortedSetOf()` for `TreeSet<T>`.

Here's a set of fruits, with a duplicate element:

```
// sets.kts
val fruits: Set<String> = setOf("Apple", "Banana", "Apple")
```

Since sets guarantee uniqueness of elements, the duplicate is discarded in the set created:

```
println(fruits) //[Apple, Banana]
```



sets.kts

The instance created by `setOf()` is of the type `Set<T>` interface, but the underlying implementation, the class, is from the JDK:

```
// sets.kts
println(fruits::class) //class java.util.LinkedHashSet
println(fruits.javaClass) //class java.util.LinkedHashSet
```

Just like on `List<T>`, there are plenty of functions on `Set<T>` and `MutableSet<T>`: operations like `+`, `-`, `contains` or `in`, and so on. The chances are the library already contains a method to accomplish the operation you'd like to carry out on a set. Take time to familiarize with the methods of Set and its mutable counterpart.

Instead of keeping a collection of values or objects, often we want a collection of key-value pairs, and `kotlin.collections.Map<K, V>` is exactly for that, which we'll cover in the next lesson.