Java 9 Immutable Collections : List.of(), Set.of() And Map.of()

pramodbablad September 3, 2021 0

Immutable collections are the collections which can not be modified once they are created. Java 9 has introduced some static factory methods to easily create immutable collections like List, Set and Map. Before Java 9, wrapper methods of Collections class are used to create, not immutable, but unmodifiable collections. In this post, we will see how unmodifiable collections are used to created before Java 9? how to create Java 9 immutable collections? difference between immutable vs unmodifiable collections and characteristics of Java 9 immutable collections.

Before Java 9 : Creating Unmodifiable Collections

Before Java 9, `Collections.unmodifiableXXX()` methods are used to create unmodifiable collections. These methods just behave like wrapper methods which return unmodifiable view or read-only view of the original collection. i.e you can't perform modifying operations like add, remove, replace, clear etc through the references returned by these wrapper methods. But, you can modify original collection if you have other references to it and those modifications will be reflected in the view returned by these methods.

For example, in the below program, `unModifiableSportList` is created from `sportList` through `Collections.unmodifiableList()`. `unModifiableSportList` just acts as a read-only view of the original `sportList`. You can't add elements to `unModifiableSportList`. If you try to add, it will give `UnsupportedOperationException`. But, you can add elements to original `sportList` and those elements will be reflected in `unModifiableSportList`.

```
import java.util.ArrayL
import java.util.Collec
import java.util.List;

public class Java9Immut
{
    public static void
    {
        List<String> sp

        sportList.add('
        sportList.add('
        sportList.add('

        List<String> un

        System.out.prir

        System.out.prir

        unModifiableSpo

        sportList.add('

        System.out.prir

        System.out.prir

    }
}
```

There are other wrapper methods available in `Collections` class to create unmodifiable collections like `Collections.unmodifiableSet` to create unmodifiable set and `Collections.unmodifiableMap` to create unmodifiable map.

Java 9 Immutable Collections

From Java 9, static factory methods are introduced to create immutable collections.

1) Immutable List

Immutable List is created by calling `List.of()` method. This method has other overloaded forms to facilitate the creation of immutable list with desired number of elements. They are as follows.

//Returns immutable list with zero element.
of()

//Returns immutable list with one element.
of(E e1)

//Returns immutable list with two elements.
of(E e1, E e2)

//Returns immutable list with three elements.
of(E e1, E e2, E e3)

//Returns immutable list with four elements.
of(E e1, E e2, E e3, E e4)

//Returns immutable list with five elements.
of(E e1, E e2, E e3, E e4, E e5)

//Returns immutable list with six elements.
of(E e1, E e2, E e3, E e4, E e5, E e6)

//Returns immutable list with seven elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7)

//Returns immutable list with eight elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8)

//Returns immutable list with nine elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9)

//Returns immutable list with ten elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9, E e10)

//Returns immutable list with arbitrary number of elements.
of(E… elements)

Examples :

```
//Creating immutable lis
List<String> immutableSp

//Creating immutable lis
List<String> immutableNa

//Creating immutable lis
List<Integer> immutaleNu
```

2) Immutable Set

Immutable set is created by invoking `Set.of()` method. This method also has several overloaded versions to create immutable set with desired number of elements.

```
//Returns immutable set with zero element.
of()

//Returns immutable set with one element.
of(E e1)

//Returns immutable set with two elements.
of(E e1, E e2)

//Returns immutable set with three elements.
of(E e1, E e2, E e3)

//Returns immutable set with four elements.
of(E e1, E e2, E e3, E e4)

//Returns immutable set with five elements.
of(E e1, E e2, E e3, E e4, E e5)

//Returns immutable set with six elements.
of(E e1, E e2, E e3, E e4, E e5, E e6)

//Returns immutable set with seven elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7)

//Returns immutable set with eight elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8)
```

//Returns immutable set with nine elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9)

//Returns immutable set with ten elements.
of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9, E e10)

//Returns immutable set with arbitrary number of elements.
of(E… elements)

Examples :

```
//Creating immutable set
Set<String> immuatbleCit
 
//Creating immutable set
Set<Double> immutableNum
```

3) Immutable Map

Immutable map is created by calling either `Map.of()` or `Map.ofEntries()` in which `Map.of()` has several overloaded forms.

//Returns immutable map with zero mapping.
of()

//Returns immutable map with one mapping.
of(K k1, V v1)

//Returns immutable map with two mappings.
of(K k1, V v1, K k2, V v2)

//Returns immutable map with three mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3)

//Returns immutable map with four mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3, K k4, V v4)

//Returns immutable map with five mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3, K k4, V v4, K k5, V v5)

//Returns immutable map with six mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3, K k4, V v4, K k5, V v5, K k6, V v6)

//Returns immutable map with seven mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3, K k4, V v4, K k5, V v5, K k6, V v6, K k7, V

v7)

//Returns immutable map with eight mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3, K k4, V v4, K k5, V v5, K k6, V v6, K k7, V v7, K k8, V v8)

//Returns immutable map with nine mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3, K k4, V v4, K k5, V v5, K k6, V v6, K k7, V v7, K k8, V v8, K k9, V v9)

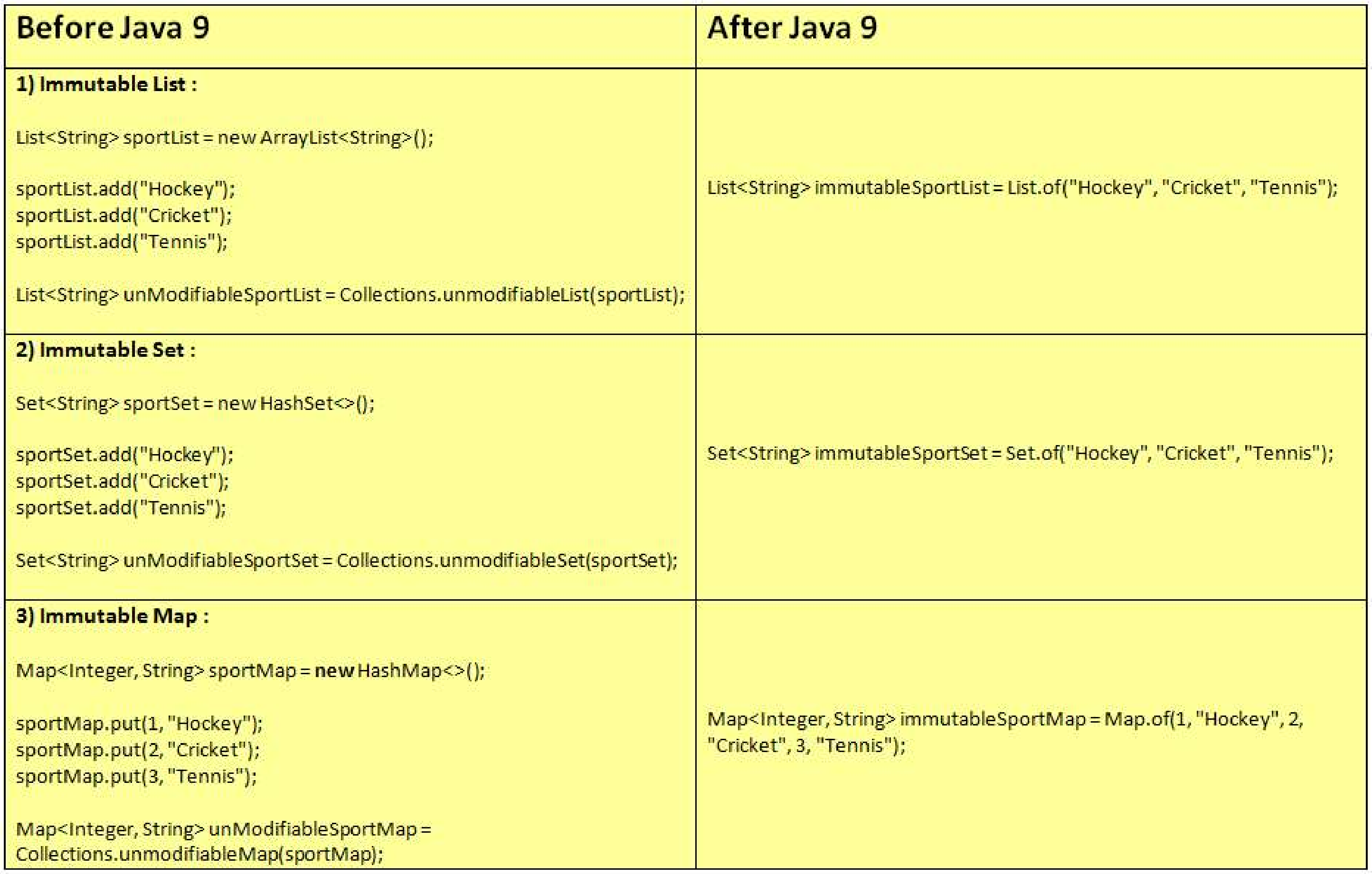
//Returns immutable map with ten mappings.
of(K k1, V v1, K k2, V v2, K k3, V v3, K k4, V v4, K k5, V v5, K k6, V v6, K k7, V v7, K k8, V v8, K k9, V v9, K k10, V v10)

//Returns immutable map with arbitrary number of mappings.
ofEntries(Entry<K k, V v>… entries)

Examples :

```
//Creating immutable ma
Map<Integer, String> im

//Creating immutable ma
Map<Integer, String> im





```

Below table shows some of before and after Java 9 code snippets.



Before Java 9	After Java 9
1) Immutable List : `List<String> sportList = new ArrayList<String>();` `sportList.add("Hockey");` `sportList.add("Cricket");` `sportList.add("Tennis");` `List<String> unModifiableSportList = Collections.unmodifiableList(sportList);`	`List<String> immutableSportList = List.of("Hockey", "Cricket", "Tennis");`
2) Immutable Set : `Set<String> sportSet = new HashSet<>();` `sportSet.add("Hockey");` `sportSet.add("Cricket");` `sportSet.add("Tennis");` `Set<String> unModifiableSportSet = Collections.unmodifiableSet(sportSet);`	`Set<String> immutableSportSet = Set.of("Hockey", "Cricket", "Tennis");`
3) Immutable Map : `Map<Integer, String> sportMap = new HashMap<>();` `sportMap.put(1, "Hockey");` `sportMap.put(2, "Cricket");` `sportMap.put(3, "Tennis");` `Map<Integer, String> unModifiableSportMap = Collections.unmodifiableMap(sportMap);`	`Map<Integer, String> immutableSportMap = Map.of(1, "Hockey", 2, "Cricket", 3, "Tennis");`

Immutable Vs Unmodifiable :

Java 9 Immutable collections and unmodifiable collections returned by the `Collections.unmodifiableXXX()` wrapper methods are not the same. Unmodifiable collections are just the read-only views of the original collection. You can perform modifying operations on the original collection and those modifications will be reflected in the collections returned by these methods. But, immutable collections returned by Java 9 static factory methods are 100% immutable. You can't modify them once they are created.

Characteristics Of Java 9 Immutable Collections :

1. Modifying operations on immutable collections are not allowed. If you try to modify them, `UnsupportedOperationException` will be thrown.
2. Null elements are not allowed. They will give `NullPointerException` at run time.
3. Java 9 immutable collections are thread safe. You can use them in a multi threaded environment without synchronization.
4. Immutable collections returned by Java 9 static factory methods are space efficient. They consume less memory than the mutable collections.

Where to use them?

You can consider using Java 9 immutable collections if you have lots of known values and those values never change throughout the execution and those values are retrieved frequently. In such scenarios, immutable collections give better performance than mutable collections.

Also Read :

- Java 9 Interface Private Methods
- Java 9 JShell
- Java 9 Immutable Collections Oracle Doc