CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

Set interface

Overview

1. Introduction
2. Set interface
3. Hierarchy of Set interface
4. Features of Set
5. Set Implementation
6. Set Methods
7. Generic Set objects

Introduction

A **set in Java** is an unordered collection of unique elements or objects. In other words, a set is a collection of elements that are not stored in a particular order.

Elements are simply added to the set without any control over where they go. For example, in many applications, we do not care about the order of elements in a collection.

Introduction

Consider the figure below where a set of books by topics is not arranged. A collection of elements is not stored in a particular order into a Set. Such a collection of elements based without order but uniqueness is called Set in Java.

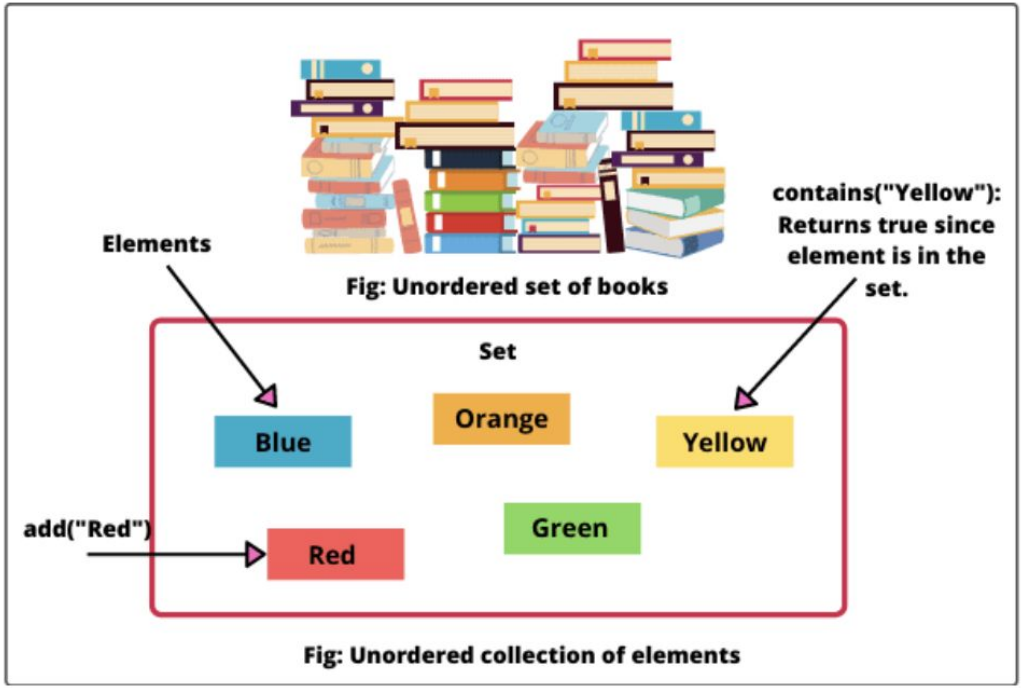


Fig: Unordered set of books

Fig: Unordered collection of elements

Introduction

We can add elements in a set and iterate over all elements in a set. It will grow dynamically when elements are stored in it. We can also check whether a particular element is contained in the set.

A set will **not** allow duplicate elements. That means an element can exist only once in the set. If we try to add the same element that is already present in the set, then it is not stored in the set. That's why each element in a set is unique.

Set in Java can be useful to remove duplicate elements from the collection. Set holds a single reference to an object. It does not provide two references to the same object, two references to `null`, or references to two objects a and b such that `a.equals(b)`.

Set interface

`Set` is an interface that was introduced in Java 1.2 version. It is a generic interface that is declared as:

```
interface Set<E>
```

Here, `E` defines the type of Elements that the set will hold.

The java `Set` interface does not provide any additional methods of its own. It uses methods defined by the `collection` interface but places additional restrictions on those methods.

Set interface - restrictions

For example, when `Set` uses `add()` or `addAll()` methods defined by the `Collection` interface, it does not add an element to the Set if the Set already contains that element.

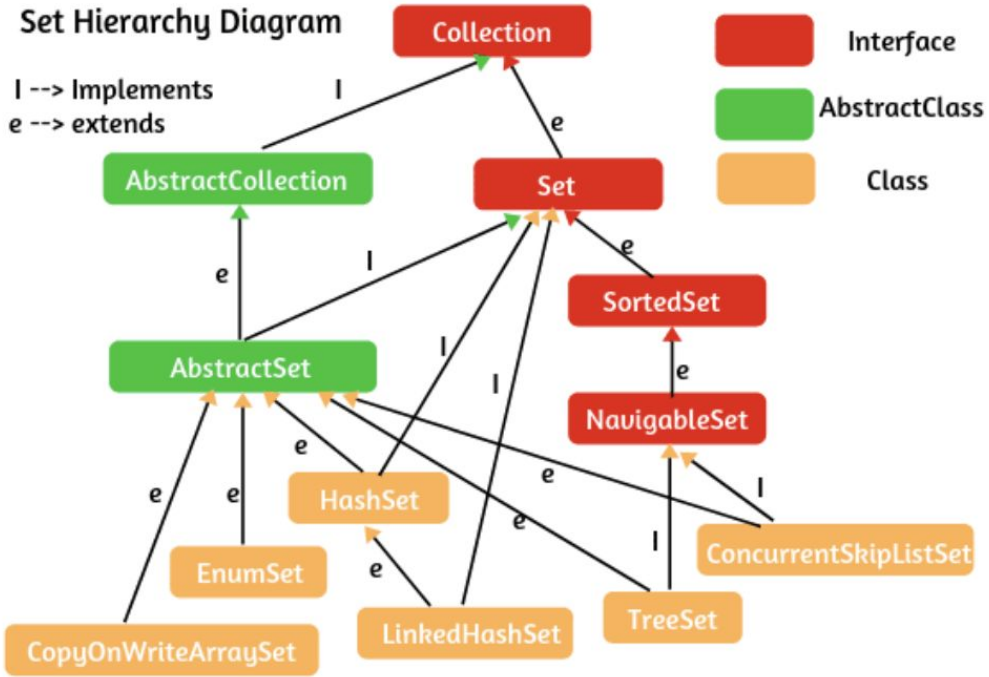
`Set` interface does not provide any `get()` method like `List` to retrieve elements. Therefore, the only way to get elements from the Set is to do so using `iterator()` method. But this method does not return elements from the set in any particular order.

Using the `Iterator`, we can traverse only in the forward direction from the first to last element. We cannot traverse over elements in the backward direction using iterator method.

Hierarchy of Set interface

- The java `Set` interface `extends` `java.util.collection` interface.
- The `java.util.SortedSet` interface `extends` the `Set` interface to provide the sorted set of elements.
- Three classes such as `HashSet`, `LinkedHashSet`, and `TreeSet` implement `Set` interface.
- `ConcurrentSkipListSet` and `EnumSet` classes also implements `Set` interface. The hierarchy diagram of the `Set` interface in Java is shown in next figure.

Hierarchy of Set Interface in Java



- The java `Set` interface `extends` `java.util.collection` interface.
- The `java.util.SortedSet` interface `extends` the `Set` interface to provide the sorted set of elements.
- Three classes such as `HashSet`, `LinkedHashSet`, and `TreeSet` implement `Set` interface.
- `ConcurrentSkipListSet` and `EnumSet` classes also implements `Set` interface. The hierarchy diagram of the `Set` interface in Java is shown in next figure.

Features of Set

1. Set is an unordered collection of elements. That means the order is not maintained while storing elements. While retrieving we may not get the same order as that we put in elements.

2. It is used to store a collection of elements without duplicate. That means it contains only unique elements.

3. Set uses map based structure for its implementation.

4. It can be iterated by using `Iterator` but cannot be iterated by using `ListIterator`.

5. Most of the `Set` implementations allow adding only one `null` element. TreeSet does not allow to add `null` element.

6. Set is not an indexed-based structure like a list in Java. Therefore, we cannot access elements by their index position.

8. It does not provide any get method like a list.

Set Implementation

Java collections framework provides three general-purpose Set implementations: `HashSet, TreeSet,` and `LinkedHashSet.`

1. **`HashSet`** is a concrete class that implements set interface. It uses a hash table mechanism to store its elements. It is the best-performing implementation.

2. **`TreeSet`** is a concrete class that implements SortedSet interface (a subinterface of set). It uses a binary search tree mechanism to store its elements. It orders its elements based on their values. It is considerably slower than HashSet.

3. **`LinkedHashSet`** is a concrete class that extends HashSet class. It stores its elements using hash table mechanism with a linked list implementation.

Set Implementation

It orders its elements based on the order in which they were inserted into the set. That is, elements in the HashSet are not ordered but elements in the LinkedHashSet can be retrieved in the order in which they were inserted into the set.

This is the brief idea of set implementation classes. We will learn more detail about HashSet, TreeSet, and LinkedHashSet classes in further tutorials.

Set Methods

A set interface has the following various useful methods such as add, remove, clear, size, etc to enhance the usage of a set interface in the collections framework. They are listed in the table:

Set Methods

Method	Description
boolean add(Object o)	It is used to add the specified element in this set.
boolean addAll(Collection c)	This method adds all the elements in the given collection.
int size()	It is used to get the number of elements in the set.
boolean isEmpty()	This method checks that the set is empty or not.
void clear()	It is used to remove all the elements from the set.
boolean contains(Object o)	This method returns true if this set contains the given element.

Set Methods

Method	Description
boolean containsAll(Collection c)	This method returns true if this set contains all the elements of the given collection.
boolean remove(Object o)	It is used to remove the specified element from this set.
boolean removeAll(Collection c)	It removes all the elements in the given collection from the set.
Iterator iterate()	It returns an Iterator over the elements in this set.
boolean equals(Object o)	It is used to compare the given element for equality in this set.
int hashCode()	It is used to get the hashCode value for this set.

Generic Set objects

We can create a generic `Set` object by using one of its three concrete classes: `HashSet`, `LinkedHashSet`, and `TreeSet`. The syntax to create set objects is as follows:

Syntax:

```
Set<T> set = new HashSet<T>(); where T is type of generic.
Set<T> set = new LinkedHashSet<T>();
Set<T> set = new TreeSet<T>();
```

For example:

```
Set<Integer> set1 = new HashSet<Integer>(); // Creates an empty set of Integer objects.
Set<String>  set2 = new LinkedHashSet<String>(); // Creates an empty set of String objects.
```

Set-based Example Programs based on basic Operations

1. Adding Elements to Set: The add() method returns true if set does not contain the specified element. If set already contains the specified element then it will return false. Let’s take an example program based on it.

Example 1: Add elements to Set

```
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;
public class SetTester1 {
   public static void main(String[] args) {
       // Create a generic set object of type String.
       Set<String> s = new HashSet<String>();
       int size = s.size();
       System.out.println("Size before adding elements: " +size);

       // Adding elements to set.
       s.add("Orange"); // s.size()is 1.
       s.add("Red"); // s.size() is 2.
       s.add("Blue"); // s.size() is 3.
       s.add("Yellow"); // s.size() is 4.
       s.add("Green"); // Now s.size() is 5.

       // Add duplicate elements in the set. These elements will be ignored by set due to not taking duplicate elements.
       s.add("Red"); // s.size() is still 5 because we cannot add duplicate element.
       s.add("Blue"); // s.size() is still 5 because we cannot add duplicate element.
       System.out.println("Unordered Set Elements");
       System.out.println(s);

       // Check 'Black' element is present in the above set or not.
       if(s.equals("Black")) {
           System.out.println("Black element is not present in set.");
       }
       else {
           s.add("Black");
           System.out.println("Black is added successfully.");
           System.out.println("Set Elements after adding black element");
           System.out.println(s);
       }
       // Create another set object to add collection of elements to the above set.
       Set<String> s2 = new LinkedHashSet<String>();
       s2.add("White");
       s2.add("Brown");
       s2.add("Red"); // Duplicate element.

       // Call addAll() method to add all the elements of the given collection.
       s.addAll(s2);
       System.out.println("Set Elements after adding elements from given collection");
       System.out.println(s);
   }
}
```

Example 1: Add elements to Set

Output:

```
Size before adding elements: 0
Unordered Set Elements
[Red, Blue, Yellow, Orange, Green]
Black is added successfully.
Set Elements after adding black element
[Red, Blue, Yellow, Black, Orange, Green]
Set Elements after adding elements from given collection
[Red, Brown, White, Blue, Yellow, Black, Orange, Green]
```

- In this example program, you can observe in the output, Set does not maintain the order of elements while storing. It gives an unordered collection of elements in HashSet.
- To impose an order on them, we need to use the LinkedHashSet class, which we will learn in the next section.
- When we add duplicate elements in the set, it rejected duplicate elements because the set does not allow duplicate elements (keep in mind these points).

Set Example Programs based on basic Operations

2. Removing an Element from Set: Let's take a program where we will remove an element from set. We will also check the set is empty or not before adding elements in the list.

Example 2: Remove elements from Set

```
import java.util.HashSet;
import java.util.Set;
public class SetTester2 {
   public static void main(String[] args) {
       // Create a generic set object of type String.
       Set<String> s = new HashSet<String>();

       // Check set is empty or not.
       boolean check = s.isEmpty(); // Return type of this method is an boolean.
       System.out.println("Is Set empty: " +check);

       // Adding elements to set.
       s.add("Orange");
       s.add("Red");
       s.add("Blue");
       s.add("Yellow");
       s.add("Green");
       if(s.isEmpty()) {
           System.out.println("Set is empty.");
       }
       else {
           System.out.println("Set is not empty.");
           System.out.println("Elements in the Set");
           System.out.println(s);
       }
       // Remove an element from set.
       s.remove("Blue");
       System.out.println("Set elements after removing");
       System.out.println(s);

       // Get the total number of set elements.
       int size = s.size();
       System.out.println("Total number of elements: " +size);
   }
}
```

Example 2: Remove elements from Set

Output:

```
Is Set empty: true
Set is not empty.
Elements in the Set
[Red, Blue, Yellow, Orange, Green]
Set elements after removing
[Red, Yellow, Orange, Green]
Total number of elements: 4
```

Set Example Programs based on basic Operations

3. Searching an Element in Set: Let’s make a program where we will search an element in set. The contains() method checks whether an element is present in the set. If it is present in the set, the method returns true, otherwise false.

Example 3: Searching an element in Set

```
import java.util.HashSet;
import java.util.Set;
public class SetTester3 {
   public static void main(String[] args) {
       Set<Character> s = new HashSet<>();
       s.add('D');
       s.add('F');
       s.add('H');
       s.add('P');
       s.add('K');
       s.add(null);
       s.add(null); // Duplicate null element. Therefore, set allow only one null element.
       System.out.println("Unordered Set Elements");
       System.out.println(s);

       // Call contains() method to search an element.
       boolean search = s.contains('A'); // Returns false because A is not present in the set.
       System.out.println("Is Element A present in set: " +search);
       if(s.contains('K')) {
           System.out.println("K is present in set.");
       }
       else {
           System.out.println("K is not present in set.");
       }
       int hashcode = s.hashCode();
       System.out.println("HashCode value: " +hashcode);
   }
}
```

Example 3: Searching an element in Set

Output:

```
Unordered Set Elements
[P, null, D, F, H, K]
Is Element A present in set: false
K is present in set.
HashCode value: 365
```

When to use Set?

1. If you want to represent a group of individual elements as a single entity where duplicates are not allowed and insertion order is not preserved then we should go for the Set.

2. If your requirement is to get unique elements, set is the best choice for this.

3. If you want to remove duplicate elements without maintaining the insertion order from the non-set collection, you should go for set.

Let’s take a simple example program where we will remove duplicate elements from the ArrayList. Look at the source code.

Example 4: Use Set to remove duplicates from Array

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
public class SetTester4 {
   public static void main(String[] args) {
       // Create a generic list object of type Integer.
       List<Integer> list = new ArrayList<Integer>();
       int size = list.size();
       System.out.println("Size before adding elements: " +size);
       list.add(5);
       list.add(10);
       list.add(5);
       list.add(15);
       list.add(20);
       list.add(10);
       list.add(20);
       list.add(30);
       System.out.println("Original order of List Elements");
       System.out.println(list);

       Set<Integer> s = new HashSet<Integer>(list);
       System.out.println("Unordered List Elements after removing duplicates.");
       System.out.println(s);
   }
}
```

Example 4: Use Set to remove duplicates from Array

Output:

```
Size before adding elements: 0
Original order of List Elements
[5, 10, 5, 15, 20, 10, 20, 30]
Unordered List Elements after removing duplicates.
[20, 5, 10, 30, 15]
```

Set - Pop Quiz

Question:
Assume that `set1` is a set that contains string elements such as "Banana", "Orange", and "Apple". `set2` is another set that contains string elements such as "Banana", "Orange", and "Mango".

Answer the following questions:

a) What are the elements of set1 and set2 after executing set1.addAll(set2)?

b) What are the elements of set1 and set2 after executing set1.add(set2)?

c) What are the elements of set1 and set2 after executing set1.removeAll(set2)?

d) What are the elements of set1 and set2 after executing set1.remove(set2)?

e) What are elements of set1 and set2 after executing set1.retainAll(set2)?

f) What is the size of set1 after executing set1.clear()?

g) What is the size of set2 after executing set2.add(“Mango”)?

Set - Pop Quiz

Q	Answer
a)	`set1`: `["Apple", "Mango", "Orange", "Banana"]`, `set2`: `["Mango", "Orange", "Banana"]`
b)	Compilation error: The method `add(String)` in the type `Set<String>` is not applicable for the arguments (`Set<String>`)
c)	`set1: ["Apple"], set2: ["Mango", "Orange", "Banana"]`
d)	`set1: ["Apple", "Orange", "Banana"], set2: ["Mango", "Orange", "Banana"]`
e)	`set1: ["Orange", "Banana"], set2: ["Mango", "Orange", "Banana"]`
f)	After executing `set1.clear()`, size of `set1` is `0`.
g)	After executing `set2.add("Mango")`, the size of `set2` is `3` because duplicate elements cannot be added to the set

END