EnumSet in Java

Difficulty Level : Medium Last Updated : 17 Jan, 2022

Enumerations or popularly known as enum serve the purpose of representing a group of named constants in a programming language. For example, the 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit.

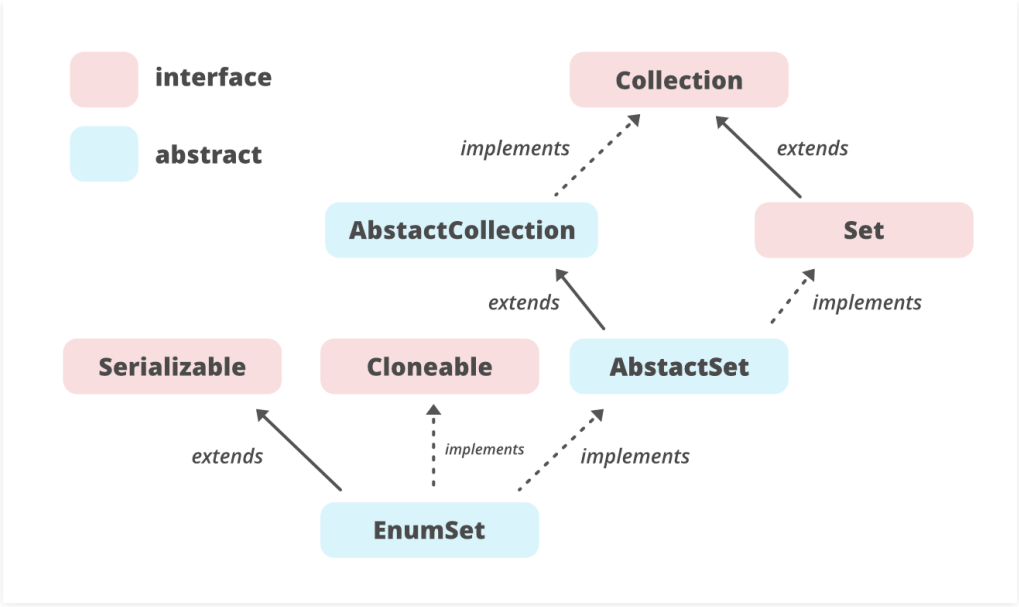
The **EnumSet** is one of the specialized implementations of the Set interface for use with the enumeration type. A few important features of EnumSet are as follows:

- It extends AbstractSet class and implements Set Interface in Java.
- EnumSet class is a member of the Java Collections Framework & is not synchronized.
- It's a high-performance set implementation, much faster than HashSet.
- All of the elements in an EnumSet must come from a single enumeration type that is specified when the set is created either explicitly or implicitly.
- It does not allow **null Objects** and throws NullPointerException if we do so.
- It uses a **fail-safe** iterator, so it won't throw ConcurrentModificationException if the collection is modified while iterating.

The Hierarchy of EnumSet is as follows:

```
java.lang.Object
   ↳ java.util.AbstractCollection<E>
        ↳ java.util.AbstractSet<E>
             ↳ java.util.EnumSet<E>
```

Here, **E** is the type of elements stored.



Syntax: Declaration

```
public abstract class EnumSet<E extends Enum<E>>
```

Here, **E** specifies the elements. E must extend Enum, which enforces the requirement that the elements must be of the specified enum type.

Benefits of using EnumSet

- Due to its implementation using **RegularEnumSet** and **JumboEnumSet,** all the methods in an EnumSet are implemented using bitwise arithmetic operations.
- EnumSet is faster than HashSet because we no need to compute any hashCode to find the right bucket.
- The computations are executed in constant time and the space required is very little.

Methods of EnumSet

Method	Action Performed
allOf (Class<E> elementType)	Creates an enum set containing all of the elements in the specified element type.
clone()	Returns a copy of this set.

Method	Action Performed
complementOf(EnumSet<E> s)	Creates an enum set with the same element type as the specified enum set, initially containing all the elements of this type that are not contained in the specified set.
copyOf(Collection<E> c)	Creates an enum set initialized from the specified collection.
copyOf(EnumSet<E> s)	Creates an enum set with the same element type as the specified enum set, initially containing the same elements (if any).
noneOf(Class<E> elementType)	Creates an empty enum set with the specified element type.
of(E e)	Creates an enum set initially containing the specified element.
of(E e1, E e2)	Creates an enum set initially containing the specified elements.
of(E first, E… rest)	Creates an enum set initially containing the specified elements.
of(E e1, E e2, E e3)	Creates an enum set initially containing the specified elements.
of(E e1, E e2, E e3, E e4)	Creates an enum set initially containing the specified elements.
of(E e1, E e2, E e3, E e4, E e5)	Creates an enum set initially containing the specified elements.
range(E from, E to)	Creates an enum set initially containing all of the elements in the range defined by the two specified endpoints.

Implementation:

```
// Java Program to Illustrate Working
// of EnumSet and its functions

// Importing required classes
import java.util.EnumSet;

// Enum
enum Gfg { CODE, LEARN, CONTRIBUTE, QUIZ, MCQ };

// Main class
// EnumSetExample
public class GFG {

    // Main driver method
    public static void main(String[] args) {

        // Creating a set
        EnumSet<Gfg> set1, set2, set3, set4;

        // Adding elements
        set1 = EnumSet.of(Gfg.QUIZ, Gfg.CONTRIBUTE,
                          Gfg.LEARN, Gfg.CODE);
        set2 = EnumSet.complementOf(set1);
        set3 = EnumSet.allOf(Gfg.class);
        set4 = EnumSet.range(Gfg.CODE, Gfg.CONTRIBUTE);

        // Printing corresponding elements in Sets
        System.out.println("Set 1: " + set1);
        System.out.println("Set 2: " + set2);
        System.out.println("Set 3: " + set3);
        System.out.println("Set 4: " + set4);
    }
}
```

Output

```
Set 1: [CODE, LEARN, CONTRIBUTE, QUIZ]
Set 2: [MCQ]
Set 3: [CODE, LEARN, CONTRIBUTE, QUIZ, MCQ]
Set 4: [CODE, LEARN, CONTRIBUTE]
```

Operation 1: Creating an EnumSet Object

Since EnumSet is an abstract class, we cannot directly create an instance of it. It has many static factory methods that allow us to create an instance. There are *two different implementations of EnumSet* provided by JDK

- RegularEnumSet
- JumboEnumSet

Note: *These are package-private and backed by a bit vector.*

RegularEnumSet uses a single long object to store elements of the EnumSet. Each bit of the long element represents an Enum value. Since the size of the long is 64 bits, it can store up to 64 different elements.

JumboEnumSet uses an array of long elements to store elements of the EnumSet. The only difference from RegularEnumSet is JumboEnumSet uses a long array to store the bit vector thereby allowing more than 64 values.

The factory methods create an instance based on the number of elements, EnumSet does not provide any public constructors, instances are created using static factory methods as listed below as follows:

- allOf(size)
- noneOf(size)
- range(e1, e2)
- of()

Illustration:

```
if (universe.length <= 64)
    return new RegularEnumSet<>(elementType, universe);
else
    return new JumboEnumSet<>(elementType, universe);
```

Example:

```
// Java Program to illustrate Creation an EnumSet

// Importing required classes
import java.util.*;
```

```
// Main class
// CreateEnumSet
class GFG {

    // Enum
    enum Game { CRICKET, HOCKEY, TENNIS }

    // Main driver method
    public static void main(String[] args)
    {

        // Creating an EnumSet using allOf()
        EnumSet<Game> games = EnumSet.allOf(Game.class);

        // Printing EnumSet elements to the console
        System.out.println("EnumSet: " + games);
    }
}
```

Output

```
EnumSet: [CRICKET, HOCKEY, TENNIS]
```

Operation 2: Adding Elements

We can add elements to an EnumSet using add() and addAll() methods.

Example:

```
// Java program to illustrate Addition of Elements
// to an EnumSet

// Importing required classes
import java.util.EnumSet;

// Main class
class AddElementsToEnumSet {

    // Enum
    enum Game { CRICKET, HOCKEY, TENNIS }

    // Main driver method
    public static void main(String[] args)
```

```
    {

        // Creating an EnumSet
        // using allOf()
        EnumSet<Game> games1 = EnumSet.allOf(Game.class);

        // Creating an EnumSet
        // using noneOf()
        EnumSet<Game> games2 = EnumSet.noneOf(Game.class);

        // Using add() method
        games2.add(Game.HOCKEY);

        // Printing the elements to the console
        System.out.println("EnumSet Using add(): "
                           + games2);

        // Using addAll() method
        games2.addAll(games1);

        // Printing the elements to the console
        System.out.println("EnumSet Using addAll(): "
                           + games2);
    }
}
```

Output

```
EnumSet Using add(): [HOCKEY]
EnumSet Using addAll(): [CRICKET, HOCKEY, TENNIS]
```

Operation 3: Accessing Elements

We can access the EnumSet elements by using the *iterator() method*.

Example:

```
// Java program to Access Elements of EnumSet

// Importing required classes
import java.util.EnumSet;
import java.util.Iterator;

// Main class
```

```
// AccessingElementsOfEnumSet
class GFG {

    // Enum
    enum Game { CRICKET, HOCKEY, TENNIS }

    // MAin driver method
    public static void main(String[] args)
    {
        // Creating an EnumSet using allOf()
        EnumSet<Game> games = EnumSet.allOf(Game.class);

        // Creating an iterator on games
        Iterator<Game> iterate = games.iterator();

        // Display message
        System.out.print("EnumSet: ");

        while (iterate.hasNext()) {

            // Iterating and printing elements to
            // the console using next() method
            System.out.print(iterate.next());
            System.out.print(", ");
        }
    }
}
```

Output

```
EnumSet: CRICKET, HOCKEY, TENNIS,
```

Operation 4: Removing elements

We can remove the elements using remove() and removeAll() methods.

Example:

```
// Java program to Remove Elements
// from a EnumSet

// Importing required classes
import java.util.EnumSet;

// Main class
```

```
// RemovingElementsOfEnumSet
class GFG {

    // Enum
    enum Game { CRICKET, HOCKEY, TENNIS }

    // Main driver method
    public static void main(String[] args)
    {
        // Creating EnumSet using allOf()
        EnumSet<Game> games = EnumSet.allOf(Game.class);

        // Printing the EnumSet
        System.out.println("EnumSet: " + games);

        // Using remove()
        boolean value1 = games.remove(Game.CRICKET);

        // Printing elements to the console
        System.out.println("Is CRICKET removed? " + value1);

        // Using removeAll() and storing the boolean result
        boolean value2 = games.removeAll(games);

        // Printing elements to the console
        System.out.println("Are all elements removed? "
                           + value2);
    }
}
```

Output

```
EnumSet: [CRICKET, HOCKEY, TENNIS]
Is CRICKET removed? true
Are all elements removed? true
```

Some other methods of classes or interfaces are defined below that somehow helps us to get a better understanding of AbstractSet Class as follows:

Methods declared in class java.util.AbstractSet

METHOD	DESCRIPTION
equals(Object o)	Compares the specified object with this set for equality.

METHOD	DESCRIPTION
hashCode()	Returns the hash code value for this set.
removeAll (Collection<?> c)	Removes from this set all of its elements that are contained in the specified collection (optional operation).

Methods declared in interface java.util.Collection

METHOD	DESCRIPTION
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
removeIf (Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
stream()	Returns a sequential Stream with this collection as its source.
toArray (IntFunction<T[]> generator)	Returns an array containing all of the elements in this collection, using the provided generator function to allocate the returned array.

Methods declared in interface java.lang.Iterable

METHOD	DESCRIPTION
forEach (Consumer<? super T> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

Methods declared in interface java.util.Set

METHOD	DESCRIPTION

METHOD	DESCRIPTION
add(E e)	Adds the specified element to this set if it is not already present (optional operation).
addAll (Collection<? extends E> c)	Adds the specified element to this set if it is not already present (optional operation).
clear()	Removes all of the elements from this set (optional operation).
contains(Object o)	Returns true if this set contains the specified element.
containsAll (Collection<?> c)	Returns true if this set contains all of the elements of the specified collection.
isEmpty()	Returns true if this set contains no elements.
iterator()	Returns an iterator over the elements in this set.
remove(Object o)	Removes the specified element from this set if it is present (optional operation).
retainAll (Collection<?> c)	Retains only the elements in this set that are contained in the specified collection (optional operation).
size()	Returns the number of elements in this set (its cardinality).
spliterator()	Creates a Spliterator over the elements in this set.
toArray()	Returns an array containing all of the elements in this set.
toArray(T[] a)	Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

Methods declared in class java.util.AbstractCollection

METHOD	DESCRIPTION

METHOD	DESCRIPTION
add(E e)	Ensures that this collection contains the specified element (optional operation).
addAll (Collection<? extends E> c)	Adds all of the elements in the specified collection to this collection (optional operation).
clear()	Removes all of the elements from this collection (optional operation).
contains(Object o)	Returns true if this collection contains the specified element.
containsAll (Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
isEmpty()	Returns true if this collection contains no elements.
iterator()	Returns an iterator over the elements contained in this collection.
remove(Object o)	Removes a single instance of the specified element from this collection, if it is present (optional operation).
retainAll (Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
toArray()	Returns an array containing all of the elements in this collection.
toArray(T[] a)	Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.
toString()	Returns a string representation of this collection.

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on GeeksforGeek's main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.