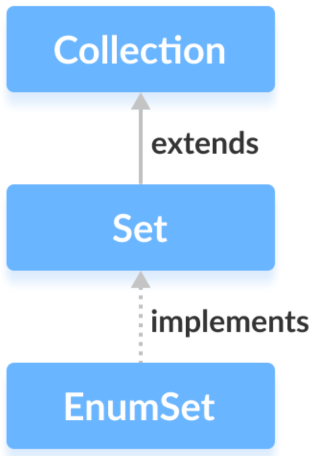
Java EnumSet

In this tutorial, we will learn about the Java EnumSet class and its various methods with the help of examples.

The `EnumSet` class of the Java collections framework provides a set implementation of elements of a single enum.

Before you learn about EnumSet, make sure to know about Java Enums.

It implements the Set interface.



Creating EnumSet

In order to create an enum set, we must import the `java.util.EnumSet` package first.

Unlike other set implementations, the enum set does not have public constructors. We must use the predefined methods to create an enum set.

1. Using allOf(Size)

The `allof()` method creates an enum set that contains all the values of the specified enum type `Size`. For example,

```
import java.util.EnumSet;

class Main {
    // an enum named Size
    enum Size {
        SMALL, MEDIUM, LARGE, EXTRALARGE
    }

    public static void main(String[] args) {

        // Creating an EnumSet using allOf()
        EnumSet<Size> sizes = EnumSet.allOf(Size.class);

        System.out.println("EnumSet: " + sizes);
    }

}
```

Output

```
EnumSet: [SMALL, MEDIUM, LARGE, EXTRALARGE]
```

Notice the statement,

```
EnumSet<Size> sizes = EnumSet.allOf(Size.class);
```

Here, `Size.class` denotes the `Size` enum that we have created.

2. Using noneOf(Size)

The `noneOf()` method creates an empty enum set. For example,

```
import java.util.EnumSet;

class Main {

     // an enum type Size
    enum Size {
        SMALL, MEDIUM, LARGE, EXTRALARGE
    }

    public static void main(String[] args) {

        // Creating an EnumSet using noneOf()
        EnumSet<Size> sizes = EnumSet.noneOf(Size.class);

        System.out.println("Empty EnumSet: " + sizes);
    }
}
```

Output

```
Empty EnumSet : []
```

Here, we have created an empty enum named `sizes`.

> **Note**: We can only insert elements of enum type `Size` in the above program. It's because we created our empty enum set using `Size` enum.

3. Using range(e1, e2) Method

The `range()` method creates an enum set containing all the values of an enum between `e1` and `e2` including both values. For example,

```java
import java.util.EnumSet;

class Main {

    enum Size {
        SMALL, MEDIUM, LARGE, EXTRALARGE
    }

    public static void main(String[] args) {

        // Creating an EnumSet using range()
        EnumSet<Size> sizes = EnumSet.range(Size.MEDIUM, Size.EXTRALARGE);

        System.out.println("EnumSet: " + sizes);
    }
}
```

Output

```
EnumSet: [MEDIUM, LARGE, EXTRALARGE]
```

4. Using of() Method

The `of()` method creates an enum set containing the specified elements. For example,

```
import java.util.EnumSet;

class Main {

    enum Size {
        SMALL, MEDIUM, LARGE, EXTRALARGE
    }

    public static void main(String[] args) {

        // Using of() with a single parameter
        EnumSet<Size> sizes1 = EnumSet.of(Size.MEDIUM);
        System.out.println("EnumSet1: " + sizes1);

        EnumSet<Size> sizes2 = EnumSet.of(Size.SMALL, Size.LARGE);
        System.out.println("EnumSet2: " + sizes2);
    }
}
```

Output

```
EnumSet1: [MEDIUM]
EnumSet2: [SMALL, LARGE]
```

Methods of EnumSet

The `EnumSet` class provides methods that allow us to perform various elements on the enum set.

Insert Elements to EnumSet

- `add()` - inserts specified enum values to the enum set
- `addAll()` inserts all the elements of the specified collection to the set

For example,

```java
import java.util.EnumSet;

class Main {

    enum Size {
        SMALL, MEDIUM, LARGE, EXTRALARGE
    }

    public static void main(String[] args) {

        // Creating an EnumSet using allOf()
        EnumSet<Size> sizes1 = EnumSet.allOf(Size.class);

        // Creating an EnumSet using noneOf()
        EnumSet<Size> sizes2 = EnumSet.noneOf(Size.class);

        // Using add method
        sizes2.add(Size.MEDIUM);
        System.out.println("EnumSet Using add(): " + sizes2);

        // Using addAll() method
        sizes2.addAll(sizes1);
        System.out.println("EnumSet Using addAll(): " + sizes2);
    }
}
```

Output

```
EnumSet using add(): [MEDIUM]
EnumSet using addAll(): [SMALL, MEDIUM, LARGE, EXTRALARGE]
```

In the above example, we have used the `addAll()` method to insert all the elements of an enum set `sizes1` to an enum set `sizes2`.

It's also possible to insert elements from other collections such as `ArrayList`, `LinkedList`, etc. to an enum set using `addAll()`. However, all collections should be of the same enum type.

Access EnumSet Elements

To access elements of an enum set, we can use the `iterator()` method. In order to use this method, we must import the `java.util.Iterator` package. For example,

```java
import java.util.EnumSet;
import java.util.Iterator;

class Main {

    enum Size {
        SMALL, MEDIUM, LARGE, EXTRALARGE
    }

    public static void main(String[] args) {

        // Creating an EnumSet using allOf()
        EnumSet<Size> sizes = EnumSet.allOf(Size.class);

        Iterator<Size> iterate = sizes.iterator();

        System.out.print("EnumSet: ");
        while(iterate.hasNext()) {
            System.out.print(iterate.next());
            System.out.print(", ");
        }
    }
}
```

Output

```
EnumSet: SMALL, MEDIUM, LARGE, EXTRALARGE,
```

Note:

- `hasNext()` returns `true` if there is a next element in the enum set

- `next()` returns the next element in the enum set

Remove EnumSet Elements

- `remove()` - removes the specified element from the enum set
- `removeAll()` - removes all the elements from the enum set

For example,

```java
import java.util.EnumSet;

class Main {

    enum Size {
        SMALL, MEDIUM, LARGE, EXTRALARGE
    }

    public static void main(String[] args) {

        // Creating EnumSet using allOf()
        EnumSet<Size> sizes = EnumSet.allOf(Size.class);
        System.out.println("EnumSet: " + sizes);

        // Using remove()
        boolean value1 = sizes.remove(Size.MEDIUM);
        System.out.println("Is MEDIUM removed? " + value1);

        // Using removeAll()
        boolean value2 = sizes.removeAll(sizes);
        System.out.println("Are all elements removed? " + value2);
    }
}
```

Output

```
EnumSet: [SMALL, MEDIUM, LARGE, EXTRALARGE]
Is MEDIUM removed? true
Are all elements removed? true
```

Other Methods

Method	Description
`copyOf()`	Creates a copy of the `EnumSet`
`contains()`	Searches the `EnumSet` for the specified element and returns a boolean result
`isEmpty()`	Checks if the `EnumSet` is empty
`size()`	Returns the size of the `EnumSet`
`clear()`	Removes all the elements from the `EnumSet`

Clonable and Serializable Interfaces

The `EnumSet` class also implements `Cloneable` and `Serializable` interfaces.

Cloneable Interface

It allows the `EnumSet` class to make a copy of instances of the class.

Serializable Interface

Whenever Java objects need to be transmitted over a network, objects need to be converted into bits or bytes. This is because Java objects cannot be transmitted over the network.

The `Serializable` interface allows classes to be serialized. This means objects of the classes implementing `Serializable` can be converted into bits or bytes.

Why EnumSet?

The `EnumSet` provides an efficient way to store enum values than other set implementations (like `HashSet`, `TreeSet`).

An enum set only stores enum values of a specific enum. Hence, the JVM already knows all the possible values of the set.

This is the reason why enum sets are internally implemented as a sequence of bits. Bits specifies whether elements are present in the enum set or not.

The bit of a corresponding element is turned on if that element is present in the set.