EnumMap class in Java

Difficulty Level : Easy   Last Updated : 06 Dec, 2021

EnumMap is a specialized implementation of the Map interface for enumeration types. It extends AbstractMap and implements the Map interface in Java. It belongs to java.util package. A few important features of EnumMap are as follows:

- EnumMap class is a member of the Java Collections Framework & is not synchronized.
- EnumMap is an ordered collection and they are maintained in the natural order of their keys(the natural order of keys means the order on which enum constants are declared inside enum type)
- It's a high-performance map implementation, much faster than HashMap.
- All keys of each EnumMap instance must be keys of a single enum type.
- EnumMap doesn't allow null key and throws NullPointerException when we attempt to insert the null key.
- Iterators returned by the collection views are weakly consistent: they will never throw ConcurrentModificationException and they may or may not show the effects of any modifications to the map that occur while the iteration is in progress.
- EnumMap is internally represented as arrays. This representation is extremely compact and efficient.

Syntax: Declaration

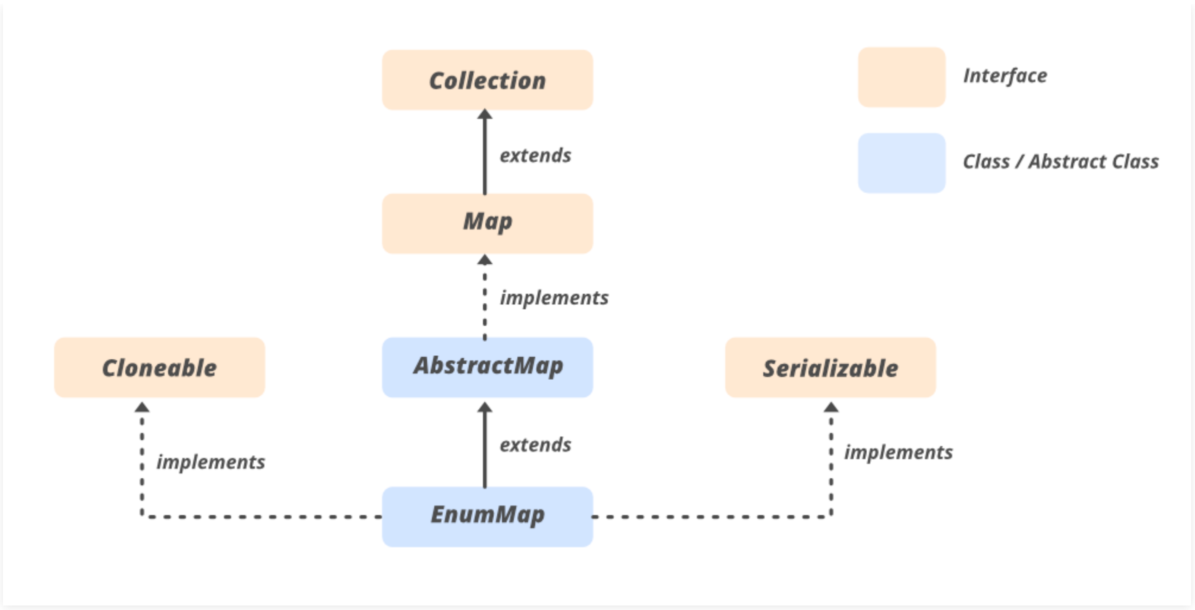
```
public class EnumMap<K extends Enum<K>,V> extends AbstractMap<K,V> implements
```

Parameters:

- Key object type
- Value object type

__K__ must extend Enum, which enforces the requirement that the keys must be of the s pecified enum type.

EnumMap Hierarchy



Constructors of EnumMap

1. **EnumMap(Class keyType):** The constructor is used to create an empty EnumMap with the specified *keyType*.
2. **EnumMap(EnumMap m):** The constructor is used to create an enum map with the same keyType as the specified enum map, with initial mappings being the same as the EnumMap
3. **EnumMap(Map m):** The constructor is used to create an enum map with initialization from the specified map in the parameter.

Example

```
// Java Program to illustrate Working of EnumMap class
// and its functions

// Importing EnumMap class
import java.util.EnumMap;

// Main class
public class EnumMapExample {

    // Enum
    public enum GFG {
        CODE,
        CONTRIBUTE,
        QUIZ,
        MCQ;
    }

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

        // Java EnumMap
        // Creating an empty EnumMap with key
        // as enum type state
        EnumMap<GFG, String> gfgMap
            = new EnumMap<GFG, String>(GFG.class);

        // Putting values inside EnumMap in Java
        // Inserting Enum keys different from
        // their natural order
        gfgMap.put(GFG.CODE, "Start Coding with gfg");
        gfgMap.put(GFG.CONTRIBUTE, "Contribute for others");
        gfgMap.put(GFG.QUIZ, "Practice Quizes");
        gfgMap.put(GFG.MCQ, "Test Speed with Mcqs");

        // Printing size of EnumMap
        System.out.println("Size of EnumMap in java: "
                           + gfgMap.size());

        // Printing Java EnumMap
        // Print EnumMap in natural order
        // of enum keys (order on which they are declared)
        System.out.println("EnumMap: " + gfgMap);

        // Retrieving value from EnumMap
        System.out.println("Key : " + GFG.CODE + " Value: "
                           + gfgMap.get(GFG.CODE));

        // Checking if EnumMap contains a particular key
        System.out.println(
            "Does gfgMap has " + GFG.CONTRIBUTE + ": "
            + gfgMap.containsKey(GFG.CONTRIBUTE));

        // Checking if EnumMap contains a particular value
        System.out.println(
            "Does gfgMap has :" + GFG.QUIZ + " : "
            + gfgMap.containsValue("Practice Quizes"));
        System.out.println("Does gfgMap has :" + GFG.QUIZ
                           + " : "
                           + gfgMap.containsValue(null));
    }
}
```

Output

```
Size of EnumMap in java: 4
```

```
EnumMap: {CODE=Start Coding with gfg, CONTRIBUTE=Contribute for others, QUIZ=P
Key : CODE Value: Start Coding with gfg
Does gfgMap has CONTRIBUTE: true
Does gfgMap has :QUIZ : true
Does gfgMap has :QUIZ : false
```

Basic Operations on EnumMap

Operation 1: Adding Elements

In order to add elements to the EnumMap, we can use put() or putAll() method as shown below.

```java
// Java Program to Add Elements to the EnumMap

// Importing EnumMap class
import java.util.EnumMap;

// Main class
// AddingElementsToEnumMap
class GFG {

    enum Color { RED, GREEN, BLUE, WHITE }
    public static void main(String[] args)
    {

        // Creating an EnumMap of the Color enum
        EnumMap<Color, Integer> colors1
            = new EnumMap<>(Color.class);

        // Insert elements in Map
        // using put() method
        colors1.put(Color.RED, 1);
        colors1.put(Color.GREEN, 2);

        // Printing mappings to the console
        System.out.println("EnumMap colors1: " + colors1);

        // Creating an EnumMap of the Color Enum
        EnumMap<Color, Integer> colors2
            = new EnumMap<>(Color.class);

        // Adding elements using the putAll() method
        colors2.putAll(colors1);
        colors2.put(Color.BLUE, 3);

        // Printing mappings to the console
        System.out.println("EnumMap colors2: " + colors2);
    }
}
```

Output

```
EnumMap colors1: {RED=1, GREEN=2}
EnumMap colors2: {RED=1, GREEN=2, BLUE=3}
```

Operation 2: Accessing the Elements

We can access the elements of EnumMap using entrySet(), keySet(), values(), get(). The example below explains these methods.

```
// Java Program to Access the Elements of EnumMap

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

// Main class
// AccessElementsOfEnumMap
class GFG {

    // Enum
    enum Color { RED, GREEN, BLUE, WHITE }

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

        // Creating an EnumMap of the Color enum
        EnumMap<Color, Integer> colors
            = new EnumMap<>(Color.class);

        // Inserting elements using put() method
        colors.put(Color.RED, 1);
        colors.put(Color.GREEN, 2);
        colors.put(Color.BLUE, 3);
        colors.put(Color.WHITE, 4);
        System.out.println("EnumMap colors : " + colors);

        // Using the entrySet() method
        System.out.println("Key/Value mappings: "
                           + colors.entrySet());

        // Using the keySet() method
        System.out.println("Keys: " + colors.keySet());

        // Using the values() method
        System.out.println("Values: " + colors.values());

        // Using the get() method
        System.out.println("Value of RED : "
                           + colors.get(Color.RED));
    }
}
```

Output

```
EnumMap colors : {RED=1, GREEN=2, BLUE=3, WHITE=4}
Key/Value mappings: [RED=1, GREEN=2, BLUE=3, WHITE=4]
Keys: [RED, GREEN, BLUE, WHITE]
Values: [1, 2, 3, 4]
Value of RED : 1
```

Operation 3: Removing elements

In order to remove the elements, EnumMap provides two variations of the remove() method.

Example

```
// Java program to Remove Elements of EnumMap

// Importing EnumMap class
import java.util.EnumMap;

// Main class
class GFG {

    // Enum
    enum Color {

        // Custom elements
        RED,
        GREEN,
        BLUE,
        WHITE
    }

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

        // Creating an EnumMap of the Color enum
        EnumMap<Color, Integer> colors
            = new EnumMap<>(Color.class);

        // Inserting elements in the Map
        // using put() method
        colors.put(Color.RED, 1);
        colors.put(Color.GREEN, 2);
        colors.put(Color.BLUE, 3);
        colors.put(Color.WHITE, 4);

        // Printing colors in the EnumMap
        System.out.println("EnumMap colors : " + colors);

        // Removing a mapping
        // using remove() Method
        int value = colors.remove(Color.WHITE);

        // Displaying the removed value
        System.out.println("Removed Value: " + value);

        // Removing specific color and storing boolean
        // if removed or not
        boolean result = colors.remove(Color.RED, 1);

        // Printing the boolean result whether removed or
        // not
        System.out.println("Is the entry {RED=1} removed? "
                           + result);

        // Printing the updated Map to the console
        System.out.println("Updated EnumMap: " + colors);
    }
}
```

Output

```
EnumMap colors : {RED=1, GREEN=2, BLUE=3, WHITE=4}
Removed Value: 4
Is the entry {RED=1} removed? true
Updated EnumMap: {GREEN=2, BLUE=3}
```

Operation 4: Replacing elements

Map interface provides three variations of the replace() method to change the mappings of EnumMap.

Example

```java
// Java Program to Replace Elements of EnumMap

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

// Main class
class GFG {

    // Enum
    enum Color {

        RED,
        GREEN,
        BLUE,
        WHITE
    }

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

        // Creating an EnumMap of the Color enum
        EnumMap<Color, Integer> colors
            = new EnumMap<>(Color.class);

        // Inserting elements to Map
        // using put() method
        colors.put(Color.RED, 1);
        colors.put(Color.GREEN, 2);
        colors.put(Color.BLUE, 3);
        colors.put(Color.WHITE, 4);

        // Printing all elements inside above Map
        System.out.println("EnumMap colors " + colors);
        // Replacing certain elements depicting colors
        // using the replace() method
        colors.replace(Color.RED, 11);
        colors.replace(Color.GREEN, 2, 12);

        // Printing the updated elements (colors)
        System.out.println("EnumMap using replace(): "
                           + colors);

        // Replacing all colors using the replaceAll()
        // method
        colors.replaceAll((key, oldValue) -> oldValue + 3);

        // Printing the elements of above Map
        System.out.println("EnumMap using replaceAll(): "
                           + colors);
    }
}
```

Output

```
EnumMap colors {RED=1, GREEN=2, BLUE=3, WHITE=4}
EnumMap using replace(): {RED=11, GREEN=12, BLUE=3, WHITE=4}
EnumMap using replaceAll(): {RED=14, GREEN=15, BLUE=6, WHITE=7}
```

Synchronized EnumMap

The implementation of an EnumMap is not synchronized. This means that if multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it must be synchronized externally. This is typically accomplished by using the synchronizedMap() method of the Collections class. This is best done at the creation time, to prevent accidental unsynchronized access.

```
Map<EnumKey, V> m = Collections.synchronizedMap(new EnumMap<EnumKey, V>(...)
```

Methods of EnumMap

- K – Type of the key object
- V – Type of the value object

Method	Action Performed
clear()	Removes all mappings from this map.
clone()	Returns a shallow copy of this enum map.
containsKey(Object key)	Returns true if this map contains a mapping for the specified key.
containsValue(Object value)	Returns true if this map maps one or more keys to the specified value.
entrySet()	Returns a Set view of the mappings contained in this map.
equals(Object o)	Compares the specified object with this map for equality.
get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
hashCode()	Returns the hash code value for this map.
keySet()	Returns a Set view of the keys contained in this map.
put(K key, V value)	Associates the specified value with the specified key in this map.
putAll(Map<? extends K,? extends V> m)	Copies all of the mappings from the specified map to this map.
remove(Object key)	Removes the mapping for this key from this map if present.
size()	Returns the number of key-value mappings in this map.
values()	Returns a Collection view of the values contained in this map.

Methods Declared in AbstractMap Class

Method	Description
isEmpty()	Returns true if this map contains no key-value mappings.

Method	Description
toString()	Returns a string representation of this map.

Methods Declared in Interface java.util.Map

Method	Descriptionenter
compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
forEach(BiConsumer<? super K,? super V> action)	Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
getOrDefault(Object key, V defaultValue)	Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
putIfAbsent(K key, V value)	If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current value.
remove(Object key, Object value)	Removes the entry for the specified key only if it is currently mapped to the specified value.

Method	Descriptionenter
replace(K key, V value)	Replaces the entry for the specified key only if it is currently mapped to some value.
replace(K key, V oldValue, V newValue)	Replaces the entry for the specified key only if currently mapped to the specified value.
replaceAll(BiFunction<? super K,? super V,? extends V> function)	Replaces each entry’s value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.

EnumMap vs EnumSet

Property	EnumMap	EnumSet
Internal Representation	EnumMap is internally represented as arrays. The representation is compact and efficient.	EnumSet is internally represented as BitVector or sequence of bits.
Permits Null Elements?	Null keys are not allowed but Null values are allowed.	Null elements are not permitted.
Is the *Abstract Class?*	No	Yes
Instantiation	Since EnumMap is not an abstract class, it can be instantiated using the new operator.	It is an abstract class, it does not have a constructor. Enum set is created using its predefined methods like allOf(), noneOf(), of(), etc.
Implementation	EnumMap is a specialized Map implementation for use with enum type keys.	EnumSet is a specialized Set implementation for use with enum types.

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.