The **EnumMap** is a special kind of Map in which the keys are only of the **Enum** type. Although it is possible to use the Enum type as a key in other Map implementations such as a HashMap or TreeMap, but a separate Map implementation was created for performance reasons.

The **EnumMap** is highly efficient, and we will discuss more when we will look at how it works internally. Let's look at some of the features of **EnumMap**.

1. **EnumMap** does not allow null keys, but it allows null values.
2. The keys are stored in their natural order. In the case of an **Enum**, the natural order of keys means the order where enum constant is declared inside Enum type.
3. The **EnumMap** is not synchronized.
4. All keys of each **EnumMap** instance must be keys of a single Enum type.
5. Iterators returned by the collection views are inconsistent. 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.
6. Java **EnumMap** implementation provides constant-time performance for the basic operations (like get and put).

Internal workings of an EnumMap

When an **EnumMap** instance is created, it is mandatory to pass the type of Enum that will be stored in this EnumMap. When we create the instance, then two arrays, **keyUniverse** and **vals**, are initialized. The size of both these arrays is equal to the number of elements in the Enum. The **keyUniverse** array stores the elements of the Enum, and **vals** array stores the values corresponding to the keys. The *ith* element is the value to which universe[i] is currently mapped, it's null if it isn't mapped to anything, or NULL if it's mapped to null.

When an element is inserted into an **EnumMap**, the index of the element that is being inserted is calculated. Let's say we have an Enum as shown below:

```
public enum FRUITS {
       APPLE, BANANA, ORANGE
}
```

If we try to store **BANANA** in the **EnumMap**, the index of **BANANA** is 1. So the value will be stored at `vals[1]`.

Similarly, if we need to get a value for a key, then we will get the index of that key and then get the value using the `vals[index]` method.

As you can see, there is no need to calculate any hashcode in an **EnumMap**, and each bucket contains only one element. This makes it very efficient compared to a HashMap.

Creating an EnumMap

There are three constructors available to create an instance of an **EnumMap**. Let's discuss all three of them.

Constructor taking the key type as input

There is no no-arg constructor in **EnumMap** because while creating an **EnumMap**, the type of the Enum that will be stored must be known. The reason is that based on the number of fields in the Enum, the array size is decided while. So, this constructor takes the type of Enum as input. The syntax of creating an **EnumMap**. So this constructor takes the type of Enum as input. The syntax of creating an **EnumMap** using this constructor is as seen below:

```
EnumMap<DayOfWeek, Integer> enumMap = new EnumMap<>(DayOfWeek.class);
```

Constructor taking another **EnumMap** as parameter

When an **EnumMap** is created using another **EnumMap**, then an enum map is created with the same key type as the specified **EnumMap**, initially containing the same mappings (if any). The **EnumMap** that is passed as a parameter can be empty.

Constructor taking another Map as a parameter

This creates an enum map initialized from the specified map. If the specified map is an **EnumMap** instance, this constructor behaves identically to the constructor discussed above. Otherwise, the specified map must contain at least one mapping (in order to determine the new enum map's key type).

Since all other operations such as insert, fetch, update, etc., are the same as the other Map implementations, we will just look at an example here instead of discussing each of them again.

In the below example, we have created a DayOfWeek Enum that will be used to store the week's day and how many hours a person worked on that day.

```
import java.util.EnumMap;
import java.time.DayOfWeek;

public class EnumMapDemo {

    public static void main(String args[]) {
        EnumMap<DayOfWeek, Integer> enumMap = new EnumMap<>(DayOfWeek.class);

        // Putting records in EnumMap
        enumMap.put(DayOfWeek.MONDAY, 5);
        enumMap.put(DayOfWeek.WEDNESDAY, 23);

        // Fetching the value from MONDAY
        System.out.println(enumMap.get(DayOfWeek.MONDAY));

        // Removing MONDAY from the Map
        enumMap.remove(DayOfWeek.MONDAY);
    }
}

```

Run Save Reset