HashSet is a class in the `java.utils` package which implements the **Set** interface. Some of the features of HashSet are:

1. **HashSet** does not allow duplicate elements.
2. **HashSet** allows only one null element.
3. The elements are inserted in random order in a **HashSet**.
4. A **HashSet** is internally backed by a **HashMap**.

Creating a HashSet

There are four different constructors available to create a **HashSet** in Java:

Using the no-arg constructor

The simplest way to create a **HashSet** is by using the no-arg constructor. This constructor creates a **HashSet** with an initial capacity of 16 and a **load factor** of 0.75.

Below is the code syntax to create a **HashSet**.

```
Set<Integer> set= new HashSet<>();
```

> Load factor is a number that defines when a Set should be resized. If the load factor is 0.75, then the Set should be resized when it is 75% full.

Using the Constructor that takes initial capacity

We can also provide the initial capacity while creating the **HashSet**. If we are already aware that our **HashSet** will contain more than 16 elements, then it is better to provide some initial capacity as it reduces the number of resizes. In this case, also, the default load factor (0.75) is used.

Using the constructor that takes initial capacity and load factor

We can also provide initial capacity along with the load factor while creating the **HashSet**. If we don't want frequent resizing, we can set the load factor to a higher number.

Using the constructor that takes another Set as a parameter

We can also create a **HashSet** using another Set by passing it to the constructor. The newly created **HashSet** will have the same size as that of the passed Set, whereas the load factor will default to **0.75**.

Inserting an element into a HashSet

There is an `add(E e)` method available that inserts an element into the **HashSet**. If the element is not already present, then this method puts the element and returns `true`. If the element is already present, then it returns `false`.

```
import java.util.HashSet;
import java.util.Set;

public class HashSetDemo {
    public static void main(String args[]) {
        Set<Integer> set = new HashSet<>();

        System.out.println("Inserting 23 in the HashSet:  " + set.add(23));
        System.out.println("Inserting 34 in the HashSet:  " + set.add(34));
        System.out.println("Inserting 23 in the HashSet:  " + set.add(23));

        System.out.println(set);

    }
}
```

Run Save Reset

Fetching an element from a HashSet

Unlike **ArrayList**, there is no `get()` method in a **HashSet** because a **HashSet** is not backed by an array. The elements are stored in random order in a **HashSet**, and we can't get a particular element. If we want to check whether a particular element is in the **HashSet** or not, then we can use the `contains()` method.

```
import java.util.HashSet;
import java.util.Set;

public class HashSetDemo {
    public static void main(String args[]) {
        Set<Integer> set = new HashSet<>();

        set.add(23);
        set.add(34);
        set.add(56);

        System.out.println(set.contains(23));

    }
}
```

Run Save Reset