# Iterating a HashSet

Below are the different methods to iterate over a HashSet.

## Using `for` loop

A **HashSet** can be easily iterated using an enhanced `for` loop as shown below.

```
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);

        for(int i : set) {
            System.out.println(i);
        }
    }
}
```

Run Save Reset

## Using `Iterator`

HashSet can also be iterated using an iterator as shown in the below example.

```
import java.util.HashSet;
import java.util.Iterator;
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);

        Iterator<Integer> itr = set.iterator();

        while(itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}
```

Run Save Reset

## Using `forEach()` method

We can use the `forEach(Consumer<? super T> action)` method defined in the `Iterable` class. This method was introduced in Java 8. It accepts an **action** that needs to be performed for each element as a parameter.

```
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);

        set.forEach(System.out::println);
    }
}
```

Run Save Reset

# Sorting a HashSet

Since a **HashSet** stores the elements in random order, it is not possible to store the elements in a **HashSet** in sorted order. If we want to sort the elements of a **HashSet**, then we should convert it into some other Collection such as a List, TreeSet, or LinkedHashSet. We will discuss TreeSet and LinkedHashSet in upcoming lessons.

Here we will see how we can convert a **HashSet** to an ArrayList, and then we can use the elements from the List. As discussed here, we can create an ArrayList by sending another collection to its constructor. We can sort this ArrayList using the `sort()` method of the Collections class.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
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);

        // Creating an ArrayList from existing set.
        List<Integer> list = new ArrayList<>(set);
        // Sorting the list.
        Collections.sort(list);

        list.forEach(System.out::println);
    }
}
```

Run Save Reset