



Sets

The Set Interface, HashSets, & TreeSet

This document contains ...

What is a Set?

A set is a collection of distinct elements. In a set, there are no duplicate elements. Sets are unordered (from our reference point, at least). This means we cannot expect to access items by position. The implementations of the Set interface will order elements in ways out of our control such that they are more efficient. In Java, Set is the name of an interface (`java.util.Set<E>`). Two library classes, `HashSet` (`java.util.HashSet<E>`) and `TreeSet` (`java.util.TreeSet<E>`) implement the interface.



When Might You Use a Set?

- If you have a set of strings to implement a spelling dictionary. You don't really care the order in which they are maintained, so long as you can access them quickly to look for a match.
- Hold items that have already been processed by some system – when you only want to process an item once, and not process duplicates. You can put the item into the set the first time you process it, then check if the set contains the item before processing it again.

What is in the Set Interface (Java 5)?

- `boolean add(E x)`
 - Adds an element `x`, if not already in the set.
 - Returns `true` if successfully added.
 - Returns `false` if element is already in the set and cannot be added again.
- `boolean contains(Object x)`
 - Returns `true` if the set contains `Object x`.
- `boolean remove(Object x)`
 - Removes `Object x`, returns `true` if it was, in fact, in the set prior to removal.
- `int size()`
 - Returns the number of elements in the set.
- `Iterator<E> iterator()`
 - Creates an iterator for use with the set that can be used to iterate over the entire set.
 - Elements are NOT visited in the order in which you inserted them. They are visited in the order in which the library implementation keeps them for rapid execution of its methods.
 - Because iterator exists, you can also use a `for..each` loop to do this.

Using the Library Implementations

It is considered good style to store a reference to a `HashSet` or `TreeSet` in a variable of type `Set`, so that if you decide to switch which is implementing it later, you need only change one line of code. This will allow you to program with independence on the implementation of `Set`, because you can just use the methods specified in the interface. Your declaration should look like one of the following two lines:

```
Set<Type> setName=new TreeSet<Type>();
```

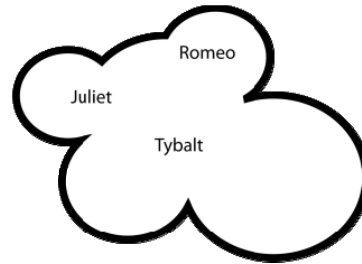
```
Set<Type> setName=new HashSet<Type>();
```

You would replace `Type` with the data type to be stored in the `Set`.

Some Example Code

Note – everything except the constructor will look identical for either TreeSet or HashSet.

```
//construct a set
Set <String> names = new HashSet<String>();
//add elements to the set
names.add("Romeo");
names.add("Juliet");
names.add("Tybalt");
```



```
//check containment
if(names.contains("Juliet"))
{
    System.out.println("Present!");
}

//remove
names.remove("Tybalt");
```

Two ways to iterate over the set:

```
//iterate through using an iterator
Iterator<String> iter=names.iterator();
while(iter.hasNext())
{
    String name=iter.next();
    //process however
}
}
```

```
//using for each loop
for(String name: names)
{
    //do something with name
}
}
```

About HashSet...

- Stores elements in a hash table.
- Provides constant time performance for basic operations (add, remove, and contains).
- If set is modified after the iterator is created, except through iterator's own remove method, code will throw a ConcurrentModificationException.
- Items are not stored in any particular order, so they do not need to be Comparable (implement Comparable interface).
- Use for fast access and removal of items when you do not care about whether items are sorted.
- If you use this, must make sure to override default hashCode and equals methods for user-defined classes to avoid accidentally adding duplicates to a set.

About TreeSet...

- Stores elements in a balanced binary search tree.
- Provides guaranteed $O(\log(n))$ time for basic operations (add, remove, and contains).
- If set is modified after the iterator is created, except through iterator's own remove method, code will throw a `ConcurrentModificationException`.
- Maintains elements in sorted order. Guarantees that set will be in ascending order.
- Items must be `Comparable` (implement the `Comparable` interface).
- Use if elements must be sorted.