## Collection Interface

Collection interface, like the Object class, is sort of the highest level interface. It is the parent of both the List interface and the Set interface.

## The .split() method for String

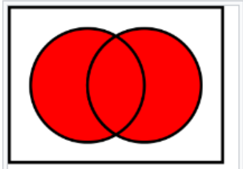
This method will split up a String into a String array. The splits are based on the parameter you pass into the method. Most commonly you would split by spaces (“ ”).

```
Ex1:
String a = "1-2-3---5-7";
String[] b = a.split("-");
System.out.println(Arrays.toString(b));
Output: [1, 2, 3, , , 5, 7]
System.out.println(Arrays.toString(("wefwefwfefee").split("e")));
Output: [w, fw, fwf, f]
```

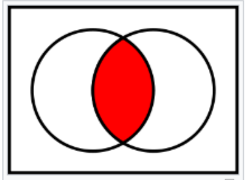
This is just a generally useful method that may show up later.

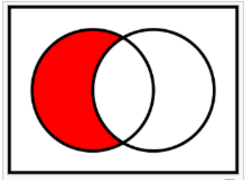
## Introduction to Sets

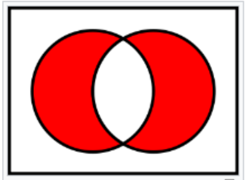
**A set** is a group of items, all of the same type, and none of them are duplicates.
Common methods for sets:


The **union** of $A$ and $B$, denoted $A \cup B$

- Union: combining two sets so that the resulting set has no duplicate
  - $[1, 3] \cup [3, 1, 11, 7, 5] = [1, 3, 11, 7, 5]$
- Intersection: making a new set with all the common elements between two sets.
  - $[5, 7, 9] \cap [9, 5, 13] = [9, 5]$
- Difference or Relative Complement: making a new set with all the elements in Set1 but not in set2. Like subtraction, order matters here.
  - The symbol is \ or $-$
  - $[9, 5, 7] \setminus [2, 9, 4] = [5, 7]$
  - $[2, 9, 4] - [9, 5, 7] = [2, 4]$
- Size of the (Cardinality). Number of elements in it.
  - Size/Cardinality of $[1, 2, -7, 80, 900] = 5$


The **intersection** of $A$ and $B$, denoted $A \cap B$


The **set difference** $A \setminus B$


The **symmetric difference** of $A$ and $B$

## The Set Interface

Any class that implements the set interface must have the following methods:
`boolean add(E obj)`

- Adds the obj to the set if it is not present and returns true. If the obj is already present in the set, it will return false.

`Boolean remove(Object o)`

If the object is present in the set, remove it and return true. If the object is not in the set, return false.

The Set interface is implemented by two classes: Hash Set and TreeSet. Both are generic and can accept any kind of object in the angle brackets.

| | Hash Set | Tree Set |
|---|---|---|
| Ordering | Elements are not ordered | Elements are ordered, always in ascending order. ( It sorts based on the `Comparable/Comparator` method of the type). |
| Null Object | Allows for a null element in the set. | Does not allow null object, since that would mess with the `Comparable/Comparator` (`NullPointer Exception`) |

## Traversing a Set

Please note there is no get method, because the set is not indexed -- it's just a group of the items. Therefore, there is also no set method.

Since there is no index, you can traverse all the elements of a set via Enhanced-for loop or an Iterator.

Notice that if you need to *remove* objects from the set while you are iterating, you must use an iterator. Otherwise, with the for-each loop you would get concurrent modification exceptions!

## Set Examples

Example 1:  Try to predict the output!

```
Set<String> words = new TreeSet<String>();
//notice we instantiate this so that we can switch the implementation
//of words to HashSet later if we wanted to.
words.add("apple");
words.add("bee");
words.add("yak");
words.add("zebra");
words.add("apple");
words.remove("bee");
System.out.println(words);
Iterator x = words.iterator();
//this is okay since TreeSet implements iterable so it has the
//iterator method.
while(x.hasNext()) {
      if(x.next().substring(0,1).compareTo("m") < 0) {
        x.remove();
}
}
System.out.println(words);
```

The output is:

```
[apple, yak, zebra]
[yak, zebra]
```

Notice how since the iterator interface doesn't require a `get()` method, we need to use our `next()` method, which returns an Object reference of the element we just got to in the collection. In order to use String Methods, we'll need to cast the object to String.
Also notice that because it's a tree set, it comes in order!

Example 2: Try to predict the output!

```
Set<Double> Eric = new HashSet<Double>();
for(int i = 0;i < 10; i++) {
     Eric.add(i + .5);
}
Eric.add(null);
Eric.add(5.5);
Eric.remove(9.5);
for(Double num: Eric) {
     System.out.print(num + " ");
}
```

The output is:
`0.5 null 8.5 4.5 2.5 5.5 1.5 6.5 3.5 7.5` , or some other order of these values.

## **Other Useful Methods for Sets**

`boolean retainAll(Collection<?> c)`
- Turn the set you called the method on, call it `a`, into the intersection between `a` and `c`.

`boolean addAll(Collection<? extends E> c)`
- Turn the set you called the method on, call it `a`, into the union between `a` and `c`.

`void removeAll(Collection<?> c)`
- Turn the set you called the method on, call it `a`, into $a - c$.

Ex: Return a new set as the Symmetric difference of X and Y. In the process of making Z, do not alter X or Y.

We can solve this by doing a union between the two differences. In other words, $Z = (X \setminus Y) \cup (Y \setminus X)$

```
public static Set<String> symDif (Set<String> X, Set<String> Y) {
        Set<String> Z1 = new HashSet<String>();
        for(String a: X)
              {Z1.add(a);} //Z1 and X are the same now.
        Z1.removeAll(Y);// X - Y
        Set<String> Z2 = new HashSet<String>();
        for(String a: Y)
              {Z2.add(a);} //Z2 and Y are the same now.
        Z2.removeAll(X); // Y - X
        Z1.addAll(Z2); //Z1 is now their union
        return Z1;
  }
```