Collections

Beside List, there are two other common collections.

Set: like a mathematical set, unordered, no repetitions.

Map: Indexes collections by arbitrary keys.

HashSet<E>

```
import java.util.HashSet;  // Or TreeSet
...
HashSet<String> set = new HashSet<String>();  // Or TreeSet
set.add(value);           // Add an item to the set
set.remove(value);        // Remove value from set
set.contains(value);      // returns true if value is in set
set.isEmpty();            // returns true if set is empty
set.addAll(other);        // set = set union other
set.retainAll(other);     // set = set intersection other
set.removeAll(other);     // set = set - other
set.containsAll(other); // Is other a subset of set
set.equals(other);        // Do set and other have same elements?
set.size();               // Returns number of elements in set
```

Set Example

Write a program that reads tokens until a token is seen twice and reports the token.

Pseudocode:

```
set up scanner
create empty set
read token
while token not in set
    add token to set
    read token
report token
```

Set Example

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    HashSet<String> tokens = new HashSet<String>();
    String tmp = in.next();
    while ( ! tokens.contains(tmp) ) {
        tokens.add(tmp);
        tmp = in.next();
    }
    System.out.println(tmp + " was seen twice");
}
```

HashMap<K,V>

```
import java.util.HashMap;  // Or TreeMap
...
HashMap<String,Integer> map = new HashMap<>();  // May leave <> empty
map.put(key,value);      // Add/replace map from key to value
map.get(key);            // Returns value key maps to, or null
map.containsKey(key);    // Returns true if map has this key
map.keySet();            // Returns a Set of all the keys in map
map.remove(key);         // Removes key and its mapping
map.size();              // Returns number of keys in map
```

Map Example

Write a program that reads tokens until a token is seen three times and reports the token.

```
set up scanner
create empty map from strings to integers
repeat
   read token
   if token is already a key
      increment its integer
      if 3 report it
   else
      map new key to 1
```

Ugly! Report is deep.

Map Example

Write a program that reads tokens until a token is seen three times and reports the token.

```
set up scanner
create empty map from strings to integers
seen = 0
while seen != 3
   read token
   if token is already a key
      increment its integer
      seen = new integer
   else
      map new key to 1
      seen = 1
report token
```

Map Example

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    HashMap<String,Integer> tokens = new HashMap<String,Integer>();
    String tmp;
    int seen = 0;
    while ( seen != 3 ) {
        tmp = in.next();
        if (tokens.containsKey(tmp)) {
            seen = tokens.get(tmp)+1;
        } else {
            seen=1;
        }
        tokens.put(tmp,seen);
    }
    System.out.println(tmp + " was seen thrice");
}
```

Interfaces

In `Foo.java`:

```
public interface Foo {
    public void add(int number);
    public void remove(int number);
}
```

In `Bar.java`:

```
public class Bar implements Foo {  // A promise to have methods
    ...
}
```

Interface Usage

```
Foo x = <any instance of a class that "implements Foo">
```

Including, assignments of newly created objects.

```
List<Integer> list = new ArrayList<Integer>();
Set<Integer> set = new HashSet<Integer>();
Map<String,Integer> map = new HashMap<String,Integer>();
```

Advantage: Can write code that works with any kind of List/Set/Map not just particular implementations.

Practice-It

Write a method `listToSet` that takes a List of integers as a parameter and returns a Set with the same elements.

```
public static Set<Integer> listToSet(List<Integer> source) {
    Set<Integer> dest = new HashSet<Integer>();
    Iterator<Integer> itr = source.iterator();
    while (itr.hasNext()) {
        dest.add(itr.next());
    }
    return dest;
}
```

Iterators

To visit each element of a collection, use an iterator.

```
Iterator<String> itr = set.iterator();  // Asks set for an iterator
while (itr.hasNext()) {
   ...
   String s = itr.next();
   ...
   itr.remove();  // remove the last thing returned via next()
   ...
}
```

Do not manipulate the collection or any of its items during iteration. You could confuse the iterator.

One exception: `itr.remove()` is allowed.