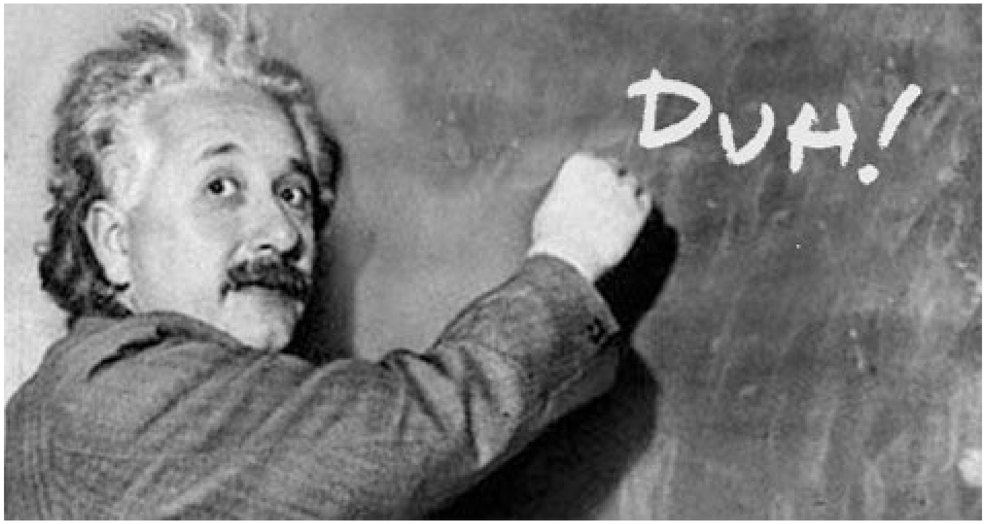
Introduction to Java Collections

…

What are collections?

The most generic
“collection” of elements



A collection — sometimes called a container — is simply an object that groups multiple elements into a single unit

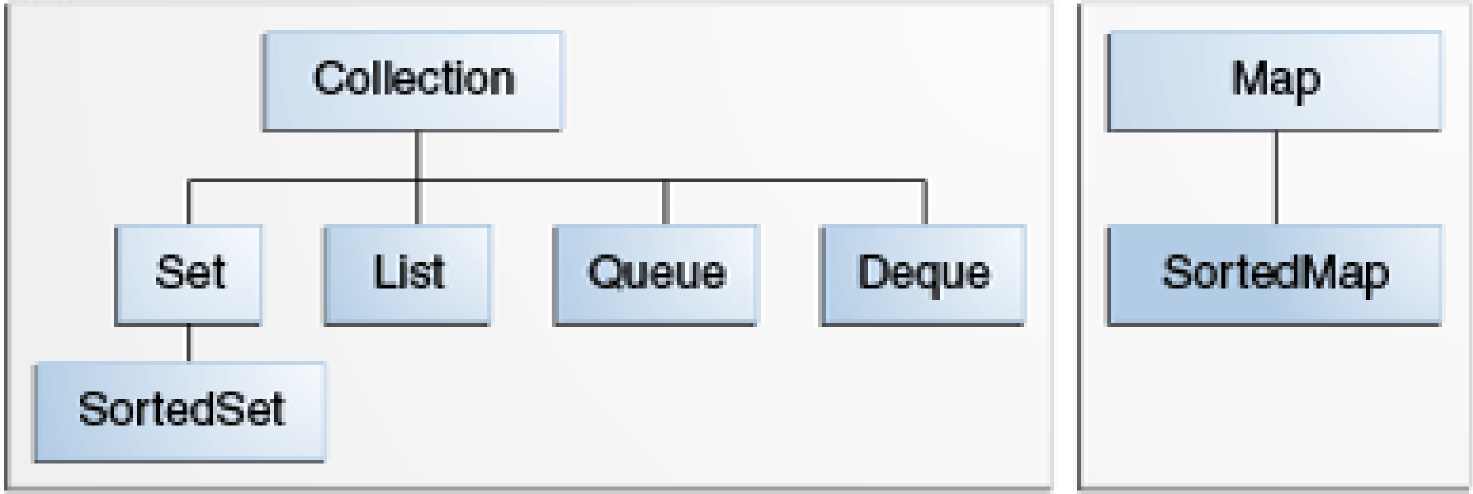
Collections are used to store, retrieve, manipulate, and communicate aggregate data

A collections framework is a unified architecture for representing and manipulating collections which contain the following 3 components;

#1

Interfaces

- abstract data types that represent collections
- allow collections to be manipulated independently of the details of their representation
- generally form a hierarchy



#2 Implementations

- concrete implementations of the collection interfaces;
- i.e. reusable data structures

#3 Algorithms

- reusable functionality / methods on objects that implement collection interfaces
- polymorphic: 1 method for many different implementations of collection interface

Using Constructors to convert collections

```
// create & populate a List / Set / Collection
Collection<String> c = new …

// Java SE 7 & sooner
List<String> list = new ArrayList<String>(c);

// Java SE 8 "diamond" operator
List<String> list = new ArrayList<>(c);
```

Available Methods – basic stuff

```
int            size()
boolean        isEmpty()
boolean        contains(Object element)
Iterator<E>    iterator()

boolean        add(E element)
boolean        remove(Object element)
```

- We'll talk more about this one in next slides

- Alright with both collections that allow or do not allow duplicates
- Makes sure the element is removed
- Returns true if the collection was modified

Why does remove() return a Boolean?

```
// remove all instances of an element
while(col.remove(anObject));

// e.g. remove all null elements
while(col.remove(null));
```

```
// Allows to simplify this...
for(Object obj : col) {
    if(obj != null){
doSomethingWithObject(obj);
    }
}

// ...with this...
while(col.remove(null));

for(Object obj : col) {
    doSomethingWithObject(obj);
}
```

stack overflow https://stackoverflow.com/questions/18895124/why-does-java-util-collection-remove-return-a-boolean

Why does add() return a Boolean?

Arrays?

- Not really useful

Sets?

- Element might already be in there

Bounded Collections

- Collection might be full

```java
// we could do these checks by hand…
if (!set.contains(item)) {
    set.add(item);
    itemWasAdded(item);
}
// … but the version below…
if (set.add(item)) {
    itemWasAdded(item);
}
// … is both shorter AND thread-safe!!!
```

stack overflow https://stackoverflow.com/questions/24173117/why-does-list-adde-return-boolean-while-list-addint-e-returns-void

Available Methods – whole collections

```
boolean containsAll(Collection<?> col)
```

- returns true if target Collection contains all of the elements in col

```
boolean addAll(Collection<? extends E> col)
```

- adds all of the elements in col to target Collection
- Returns true if collection was modified

```
boolean removeAll(Collection<?> col)
```

- removes from target Collection all elements also in col
- Returns true if collection was modified

```
boolean retainAll(Collection<?> col)
```

- i.e. retains only those elements in target Collection that are also in col
- Returns true if collection was modified

```
void clear()
```

- removes all elements from the Collection

Available Methods – Array Conversions

```
Object[] a = c.toArray(); // simple form
String[] a = c.toArray(new String[0]);
//Returned array has type of parameter array
```

```
IF      list fits in array specified as parameter
THEN    it is returned therein
        IF      size(array parameter) > size(list)
        THEN    array element immediately following end of collection is set to null
ELSE    return new array w/ runtime type of the parameter array and w/ size of list
```

http://www.tutorialspoint.com/java/util/arraylist_toarray.htm

How to traverse Collections – 3 ways

#1 – Aggregate Operations

- Not now
- When we learn about functional programming

#2 – For-each

#3 – Iterators

#2

How to traverse Collections – ForEach

```
// Prepare for some VERY intricate Java stuff…

for (Object o : collection){

    System.out.println(o);

}
```

- AKA Enhanced For Loop
- Do not confuse with forEach() method we will study when we look at Streams

#3

How to traverse Collections – Iterators

```
// this is what the iterator interface offers

public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

- returns true if iterator has more elements

- returns the next element in the iteration

- Removes last element returned by next()
- It may be called **only once** per call to next
- Throws an exception if this rule is violated

- Iterator = only safe way to modify a collection during iteration
- Behavior is unspecified if the underlying collection is modified in any other way while the iteration is in progress

Example of Iterator-based Collection Filter

```
static void filter(Collection<?> c) {
     for (Iterator<?> it = c.iterator(); it.hasNext(); ) {
          if (!cond(it.next())) {
               it.remove();
          }
     }
}
```

- Whatever conditions based on which you want to filter elements out

When should I use an Iterator vs. ForEach?

1. Do you ever need to remove the current element?

    - for-each construct hides the iterator, so you cannot call remove(…)
    - Therefore, the for-each construct is not usable for filtering

2. Do you need to Iterate over multiple collections in parallel?

    - More about this when we discuss concurrent programming

Simple example of when things may go wrong

WARNING

Demotivation.us
DON'T TRY THIS AT HOME !
They say don't try this at home
So I'm going to my friend's house to try it!
Demotivation.us

Simple example of when things may go wrong

WARNING

```
public class IterationsGoneWrong{

   public static void main (String[] args){
      Integer[] data = {1,1,1,1,1};

      ArrayList<Integer> myList = new ArrayList<>(Arrays.asList(data));

      removeDuplicate(myList);

      System.out.print("The distinct integers are ");
      for (int number: myList) {
         System.out.print(number + " ");
      }
   }
   …
}
```

NOTE
This slide uses a bit of ArrayList syntax from the next section

? Side Note – Why using both the constructor and asList()

! …asList() returned does not allow add / rm but writes through to the ArrayList object

Do not mix .remove() and index-based accessing

WARNING

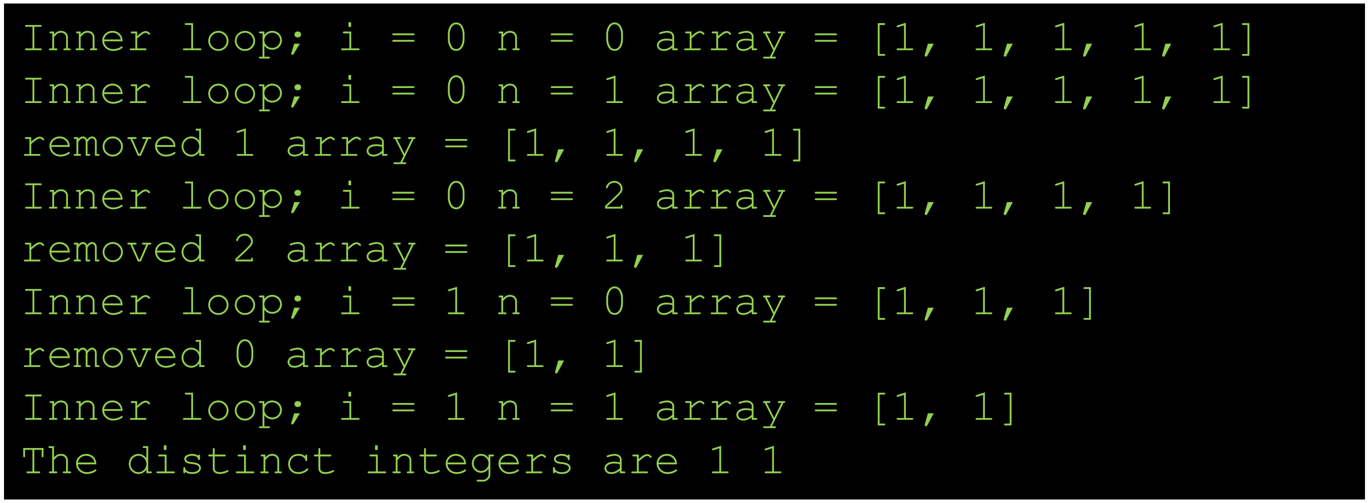
```
public static void removeDuplicate(ArrayList<Integer> list){

      for (int i=0;i<list.size();i++){
         for (int n=0; n<list.size(); n++){

            System.out.println("Inner loop; i = "+ i
                                    + " n = "+n
                                    + " array = " + list);

            if (n!=i){
               if (list.get(n)==list.get(i)){
                  list.remove(n);
                  System.out.println("removed "+n + " array = " + list);
               }
            }
         }
      }
}
```

Punch it…



```
Inner loop; i = 0 n = 0 array = [1, 1, 1, 1, 1]
Inner loop; i = 0 n = 1 array = [1, 1, 1, 1, 1]
removed 1 array = [1, 1, 1, 1]
Inner loop; i = 0 n = 2 array = [1, 1, 1, 1]
removed 2 array = [1, 1, 1]
Inner loop; i = 1 n = 0 array = [1, 1, 1]
removed 0 array = [1, 1]
Inner loop; i = 1 n = 1 array = [1, 1]
The distinct integers are 1 1
```

```
for (int i=0;i<list.size();i++)
      for (int n=0; n<list.size(); n++){

            System.out.println("Inner loop; i = "+ i
                                    + " n = "+n
                                    + " array = " + list);
            if (n!=i)
               if (list.get(n)==list.get(i)){
                  list.remove(n);
                  System.out.println("removed "+n + " array = " + list);
               }
      }
```