

# Collections

---

## Collections

Built-in classes for common storage problems (like the C++ Standard Template Library (STL))

We'll look at basic features of the Collection classes now, and see more detail later on

Collection type (sequence/set) -- ArrayList is the most useful

Map type (hash table/dictionary)-- HashMap is the most useful

They only store pointers

See the Sun docs: <http://java.sun.com/docs/books/tutorial/collections/>

The collection classes use inheritance and interfaces, but we are ignoring that for now. We will just look at the basic uses of a collection classes.

## Collection Design

As much as possible, all the various classes implement the same interface so they use the same method names (e.g. add(), iterator(), ...), so you can substitute one type of collection for another. This also makes it easier to learn, since the method names are all consistent.

Only store pointers to elements

Can store pointers to objects such as Strings, but cannot store a primitive like an int. If you need to store an int, use the wrapper class (Integer, Float, ...).

In the source code, collections store all pointers as type Object (like void\*), so you cast the pointer back to what it really is, such as String, when extracting the pointer from the collection.

At runtime, java checks all casts, so a bad cast will be caught at that time.

## Collection Messages

There are a few basic methods, and everything else is built on them.

constructor() -- collection with no elements

Actually, a Java interface cannot specify a ctor or static method, but all the collection classes implement the default ctor at a minimum.

int size() -- number of elements in the collection

This could have been called getSize() or getLength(), but they kept the name size() to remain compatible with the old Vector class.

boolean add(Object ptr)

Add a new pointer/element to the collection. Adds to the "end" for collections that have an ordering. Returns true if the collection is modified (it might not be when adding to a Set type collection).

iterator()

Return a new Iterator object set up to iterate through the collection and possibly remove elements.

The iterator responds to...

-hasNext() -- returns true if more elements,

-next() returns the next element

-remove() -- removes the element returned by the previous call to next()

It is not valid to modify the collection directly while the iterator is iterating --

e.g. calling collection.add(), collection.remove(). It is valid to modify the collection through the iterator -- iterator.remove().

Utilities -- these convenience methods are built on top of the basic methods above  
boolean isEmpty()

boolean contains(Object o) -- iterative search -- uses equals() on the elements

boolean remove(Object o) -- iterative remove -- uses equals() on the elements

boolean addAll(Collection c) -- true if receiver changed

... and so on

## ArrayList

Replaces the old "Vector" class -- like an array, but it can grow over time

add() -- add pointer to end of the collection

int size() -- number of elements

Object get(int index) -- retrieve the elem pointer, indexed (0..len-1) (not all collections support this efficiently, but ArrayList does)

iterator() -- return an iterator object to iterate over the array list

Responds to hasNext(), next() and remove() as above

## ArrayList Demo Code

```

/*
The ArrayList is replaces the old Vector class.
ArrayList implements the Collection interface, and also
the more powerful List interface features as well.
Main methods:add(), size(), get(i), iterator()
See the "Collection" and "List" interfaces.
*/
public static void demoArrayList() {
    ArrayList strings = new ArrayList();

    // add things...
    for (int i= 0; i<10; i++) {
        // Make a String object out of the int
        String numString = Integer.toString(i);
        strings.add(numString); // add pointer to collection
    }

    // access the length
    System.out.println("size:" + strings.size());

    // ArrayList supports a for-loop access style...
    // (the more general Collection does not support this)
    for (int i=0; i<strings.size(); i++) {
        String string = (String) strings.get(i);
        // Note: cast the pointer to its actual class
        System.out.println(string);
    }
}

```

```
// ArrayList also supports the "iterator" style...
Iterator it = strings.iterator();
while (it.hasNext()) {
    String string = (String) it.next(); // get and cast pointer
    System.out.println(string);
}

// Call toString()
System.out.println("to string:" + strings.toString());

// Iterate through and remove elements
it = strings.iterator(); // get a new iterator (at the beginning again)
while (it.hasNext()) {
    it.next(); // get pointer to elem
    it.remove(); // remove the above elem
}

System.out.println("size:" + strings.size());
}
/* Output...
size:10
0
1
2
3
4
5
6
7
8
9
to string:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
size:0
*/
```