

More on Arrays & Collections

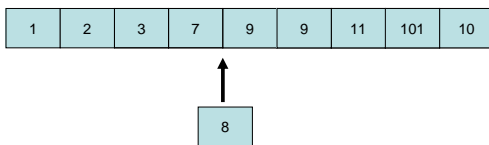
October 26, 2005

What Else?

- First let's try to insert and delete from an array.
- Not as easy as what we had last time due to need to move elements around.

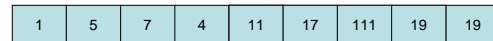
Insertion

- Insertion will need to open up a spot to insert the item – this means moving things.



Deletion

- Deletion means we will need to close up the list once we have deleted an item. There are a couple of ways to approach that...



Insertion-Deletion

```
/**
 *
 * @author Jerry Heurling
 * <BR>
 * Date: October 26, 2005
 * <BR>
 * This is the example of insertion and deletion into an
 * array that was presented in class. It inserts and
 * deletes items into an array of strings.
 */
public class InsertDelete {
    private String list[];
    private int numberInList;
```

Constructor and toString()

```
/**
 * The default constructor. Allocates the array
 * and sets the initial size of the list to zero.
 */
public InsertDelete() {
    list = new String [100];
    numberInList = 0;
}
/**
 * Convert the list to a string.
 */
public String toString() {
    String output = "";
    int currentIndex;

    for (currentIndex = 0; currentIndex < numberInList;
         currentIndex++) {
        output = output + list[currentIndex] + "\n";
    }
    return output;
}
```

Insert()

```
/**
 * Insert the given value at the spot indicated.
 * All the other items will be moved up a spot.
 * @param newValue the string to add to the list.
 * @param spot the spot at which to add the item.
 */
public void Insert ( String newValue, Int spot ) {
    Int currentIndex;

    if ( spot > numberInList ) {
        System.out.println("Can't insert -- "+
            spot + " is beyond the spots in use " +
            numberInList);
    } else if ( spot < 0 ) {
        System.out.println("Can't insert to a
            negative spot. Spot = " + spot);
    } else if ( numberInList == list.length ) {
        System.out.println("Can't insert into list --
            the list is full");
    } else {
```

insert() - continued

```
        for ( currentIndex = numberInList;
            currentIndex > spot;
            currentIndex = currentIndex-1 ) {
            list[currentIndex]=list[currentIndex-1];
        }
        list[spot] = newValue;
        numberInList++;
    }
}
```

delete()

```
/**
 * Delete the item at the spot indicated and move
 * all the following items in the list up one spot.
 * @param spot The spot to delete from the list.
 */
public void delete ( Int spot ) {
    Int currentIndex;

    if ( spot >= numberInList ) {
        System.out.println("Can't delete -- spot
            given is beyond end of list");
    } else if ( spot < 0 ) {
        System.out.println("Can't delete -- spot
            given is prior to start of list (negative)");
    } else {
```

delete() -- continued

```
        for ( currentIndex = spot;
            currentIndex < numberInList-1;
            currentIndex = currentIndex + 1 ) {
            list[currentIndex]=list[currentIndex+1];
        }
        numberInList--;
    }
}
```

location()

```
/**
 * Find the location (index) of a string in the list.
 * The method returns a -1 if the string is not found.
 * Strings are assumed to be case sensitive.
 * @param value the string to search for in the list.
 * @return the location of the value or -1 if the
 * value is not found
 */
public Int location(String value) {
    Int currentIndex;

    for ( currentIndex = 0; currentIndex < numberInList;
        currentIndex++) {
        if ( value.equals(list[currentIndex])) {
            return currentIndex; // found the value!
        }
    }
    return -1; // didn't find the value...
}
```

delete()

```
/**
 * Delete an item from the list by comparing
 * names. This routine only deletes the first
 * occurrence if the name appears more than once.
 * @param value the value to delete from the list.
 */
public void delete (String value) {
    Int slot;

    slot = location(value);
    if ( slot >= 0 ) {
        delete(slot);
    }
}
```

main()

```
/**
 * The main program inserts a group of states and then
 * deletes them in various ways. It is a simple test
 * of the methods in the class.
 * @param args -- not used.
 */
public static void main(String[] args) {
    InsertDelete test = new InsertDelete();
    test.insert("Ohio", 0);
    test.insert("Michigan", 0);
    test.insert("Maine", 0);
    System.out.println(test);
    test.insert("Missouri", 2);
    test.insert("Illinois", 0);
    test.insert("Indiana", 1);
    test.insert("Iowa", 2);
    System.out.println("This is the list after the
    insertions");
    System.out.println(test);
}
```

main () – continued

```
test.delete(0);
System.out.println("After deleting the item at
index 0");
System.out.println(test);
test.delete(5);
System.out.println("After deleting the item at
index 5");
System.out.println(test);
test.delete(3);
System.out.println("After deleting the item at
index 3");
System.out.println(test);
test.delete("Maine");
test.delete("Pennsylvania");
System.out.println("After deleting Maine and
Pennsylvania");
System.out.println(test);
}
System.exit(0);
}
```

Using an ArrayList

- The array list collection inside of Java already does some of these functions for you. The resulting class has a lot of short methods due to this.

```
import java.util.*;
/**
 *
 * @author Jerry Heuring
 * <BR>
 * Date: October 26, 2006
 * <BR>
 * An example doing insertions and deletions from a
 * collection. It turns out that most of the work is
 * already done in a so that we minimize the amount we do
 */
public class InsertDeleteCollection {
    private ArrayList list;
}
```

Constructor and toString()

```
/**
 * Default constructor – build the ArrayList
 */
public InsertDeleteCollection() {
    list = new ArrayList();
}
/**
 * Convert the list to a string.
 */
public String toString() {
    String output="";
    int currentIndex;

    for (currentIndex = 0; currentIndex < list.size();
         currentIndex++) {
        output = output +
            (String)list.get(currentIndex)+ "\n";
    }
    return output;
}
```

insert(), delete()

```
/**
 * Insert the item in the slot indicated
 * @param value the value to add to the list
 * @param slot the position to add it at.
 */
public void insert (String value, int slot) {
    list.add(slot, value);
}
/**
 * Delete the item in the slot indicated.
 * @param slot the slot to remove from the list.
 */
public void delete (int slot) {
    list.remove(slot);
}
```

location() and delete()

```
/**
 * Returns the location in the list of the value
 * @param value The value that we are going to look
 * for.
 * @return the index of the value in the list.
 */
public int location(String value) {
    return list.indexOf(value);
}

/**
 * Delete the string from the list.
 * @param value The string to remove from the list.
 */
public void delete(String value) {
    list.remove(value);
}
```

Main

- The main program is the same other than test is declared and instantiated as an InsertDeleteCollection rather than InsertDelete.

What Should We Take Away From This?

- You should realize what an array is and be able to use it in your programs.
- This includes:
 - declaration
 - allocation
 - enumeration (stepping through the array)
 - adding
 - insertion
 - deletion
 - locating an item

For Collections

- Should realize a collection does most of this and expands to contain the items (not fixed in size)
- Should realize the very basic routines:
 - add()
 - remove()
 - size()
 - get()
 - put()