

1 Announcements

- (Continue to) Read Chapter 3 by next Tuesday.
- Quiz next Tuesday (on Chapters 1 and 2)
- Labs start **this** week.
- Remember the first written homework assignment is due this Thursday and this week's lab is due next Wednesday.

2 Arrays

We shall go over a little about arrays in class but most of it is review and for those not familiar it is discussed in the book. There are a few points to remember:

- Declare an array: `int[] arr;`
- Create an array: `arr = new int[10];`
- Length of array: `int l = arr.length;`
- Do NOT go out of bounds: `arr[-1]; arr[100]; arr[10];` all throw `ArrayIndexOutOfBoundsException`.
- The array variable *refers* to the array object. It is *not* the actual array. So, `int[] arr2 = arr;` is not a copy but the exact same array.
- **Copying** an array can be done by making a routine to do it or using the clone method as follows:
`int[] arr3 = (int[]) arr.clone();`
Notice the type-casting done at the beginning. The clone method returns a type `Object` so it must be cast to the proper type, in this case an array.
- Passing arrays is the same as passing other `Objects`. The contents can be manipulated unlike with normal primitive data types.

3 First real ADT: IntArrayBag Class

We are now about to study our first real Abstract Data Type, the `IntArrayBag` Class. As its name suggests, this class represents a bag of integers. Such a data type could be useful in many ways, keeping track of a student's score. Keeping track of a classes average (though double would be better here). Keeping a list of prices. Many different things, and, when we abstract it more, it will become even more useful. The following are the methods that shall be provided for manipulating the structure (the source code shall be listed shortly afterwards - for note-taking purposes):

- Constructors: (two of them)
`public IntArrayBag(initialCapacity)` - uses given initial capacity of the bag
`public IntArrayBag()` - uses capacity of size 10
- add: used to add an element (at the end of the bag)
`public void add(int element)`

- remove: remove a particular item from the bag. If the “target” value given is present it is removed. Note this is the actual content of the bag and *not* its location.
`public boolean remove(int target)` - why does this one return boolean?
- size: size of the bag (number of elements)
`public int size()`
- countOccurrences: Count the number of times a given “target” appears in the bag
`public int countOccurrences(int target)`
- addAll: Adds the contents of the given bag to the *current* bag. Does not affect contents of given bag.
`public void addAll(IntArrayBag addend)`
- union: Similar to addAll but creates a *new* bag from two given bags. It does not affect either bag *and* is *static*. Why?
`public static IntArrayBag union(IntArrayBag b1, IntArrayBag b2)`
- clone: Makes a true COPY of the bag.
`public Object clone()`
- capacities: Methods to deal with how much space the bag is actually using:
`public int getCapacity()` - returns capacity
`public void ensureCapacity(int minimumCapacity)` - make sure it has enough space for minimum
`public void trimToSize()` - trims capacity down to true number of elements in bag. Most space efficient.

The book goes into more detail on each method, what the pre-/post-conditions are for these methods and what exceptions are thrown, including an `OutOfMemoryError`. Although we shall not deal with this issue right now, memory allocation is certainly one of great importance. Every time you create a new object, using `new`, the java interpreter *must* find that memory somewhere. It uses what is called the program’s **heap**, which contains all memory available to the program. When this memory is exhausted, rare but in larger programs or careless programs it happens... you **run out of memory**. At this point, one a **new** operation is called, an exception is thrown and the program either terminates or the program catches the exception and deals with it gracefully.

3.1 Details

This package implements the bag using an array of integers (hence the name). For those that may recall it, it will use a partially-filled array. That is, the array will be, say, size 100, but there may only be 10 elements in there. This involves keeping track of **two** variables. The array and the number of elements in the array. Actually, you also need to know the size of the array, but fortunately, this is stored by the array itself. In C/C++, you would also need to keep track of this value.

The operations are quite straightforward except when the array gets full. Now, adding an element means resizing the array, which involves allocating a new array of larger size and copying the information from one to the other, and finally setting this copy to the original array variable. We will look at the details in the code itself. The one thing also interesting to point out is the `clone()` method for the bag class. Here is that batch of code again:

```
public Object clone() {
    IntArrayBag answer;

    try {
        answer = (IntArrayBag) super.clone();
    } catch (CloneNotSupportedException e) {
        throw new RuntimeException
```

```

        ("This class does not implement Cloneable");
    }

    answer.data = (int[]) data.clone(); // Why do we need this step???
    return answer;
}

```

3.2 Running Time Analysis

Given time we shall discuss the performance of each method and analyze some variations of the implementations to see what can be done to improve performance. Otherwise, we shall continue here in the next lecture.

Here is the code, with commenting stripped off for the sake of space. It can also be found in the book, pages 126-129.

```

// File: IntArrayBag.java from the package edu.colorado.collections
// Additional javadoc documentation is available from the IntArrayBag link in:
// http://www.cs.colorado.edu/~main/docs
// package edu.colorado.collections;

public class IntArrayBag implements Cloneable {
    private int[] data;
    private int manyItems;

    public IntArrayBag() {
        final int INITIAL_CAPACITY = 10;
        manyItems = 0;
        data = new int[INITIAL_CAPACITY];
    }

    public IntArrayBag(int initialCapacity) {
        if (initialCapacity < 0)
            throw new IllegalArgumentException
                ("The initialCapacity is negative: " + initialCapacity);
        data = new int[initialCapacity];
        manyItems = 0;
    }

    public void add(int element) {
        if (manyItems == data.length) {
            ensureCapacity(manyItems*2 + 1);
        }

        data[manyItems] = element;
        manyItems++;
    }

    public void addAll(IntArrayBag addend) {
        ensureCapacity(manyItems + addend.manyItems);

        System.arraycopy(addend.data, 0, data, manyItems, addend.manyItems);
        manyItems += addend.manyItems;
    }

    public Object clone() {
        IntArrayBag answer;

        try {
            answer = (IntArrayBag) super.clone();
        }
        catch (CloneNotSupportedException e) {
            throw new RuntimeException
                ("This class does not implement Cloneable");
        }
    }
}

```

```

        answer.data = (int [ ]) data.clone( );
        return answer;
    }

    public int countOccurrences(int target) {
        int answer = 0;
        int index;
        for (index = 0; index < manyItems; index++)
            if (target == data[index])
                answer++;
        return answer;
    }

    public void ensureCapacity(int minimumCapacity) {
        int biggerArray[ ];

        if (data.length < minimumCapacity)
        {
            biggerArray = new int[minimumCapacity];
            System.arraycopy(data, 0, biggerArray, 0, manyItems);
            data = biggerArray;
        }
    }

    public int getCapacity( ) { return data.length; }

    public boolean remove(int target) {
        int index; // The location of target in the data array.
        for (index = 0; (index < manyItems) && (target != data[index]); index++);

        if (index == manyItems)
            return false;
        else {
            manyItems--;
            data[index] = data[manyItems];
            return true;
        }
    }

    public int size( ) { return manyItems; }

    public void trimToSize( ) {
        int trimmedArray[ ];

        if (data.length != manyItems) {
            trimmedArray = new int[manyItems];
            System.arraycopy(data, 0, trimmedArray, 0, manyItems);
            data = trimmedArray;
        }
    }

    public static IntArrayBag union(IntArrayBag b1, IntArrayBag b2) {
        IntArrayBag answer = new IntArrayBag(b1.getCapacity( ) + b2.getCapacity( ));
        System.arraycopy(b1.data, 0, answer.data, 0, b1.manyItems);
        System.arraycopy(b2.data, 0, answer.data, b1.manyItems, b2.manyItems);
        answer.manyItems = b1.manyItems + b2.manyItems;
        return answer;
    }
}

```