Collections: ArrayList

COMP 110: Lecture 11
Arrays solve the problem of tracking multiple objects of the same type.

Elements of an array are accessed by index.

But there’s a problem… they cannot change size.
Collections

Java has a set of classes known as the “Java Collections Framework”

These are classes that store collections of objects.

**Ordered collections:**
- `ArrayList`, `LinkedList`, `Stack`, `ArrayDeque`

**Unique elements (set theory):**
- `TreeSet`, `HashSet`, `LinkedHashSet`

**Map from one element to another:**
- `TreeMap`, `HashMap`, `LinkedHashMap`
ArrayList

ArrayList is a growable array.

Like String:
• Uses a private array to keep a list of items of the same type.
• Get items by index (.charAt(i) for String, vs. .get(i) for ArrayList).
• Has a number of elements (.length() for String, .size() for ArrayList)

Unlike String:
• Element type of ArrayList is not limited to char.
• ArrayList is mutable (changeable) (via set(), add(), remove())
```java
class CoffeeBean {
    String _name;
    String _origin;
    double _ounces;
    double _pricePerOunce;

    CoffeeBean(String name, String origin, double oz, double price) {
        _name = name;
        _origin = origin;
        _ounces = oz;
        _pricePerOunce = price;
    }

    public double getCost() {
        return _ounces * _pricePerOunce;
    }

    public String toString() {
        return _name + " from " + _origin + " " + _ounces + " @ $" + _pricePerOunce + "/oz = $" + getCost();
    }
}

We'll use this class in our ArrayList next…
```
import java.util.ArrayList;

class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<CoffeeBean> beans = new ArrayList<CoffeeBean>();

        beans.add(new CoffeeBean("Las Nubes", "Nicaragua", 12.0, 1.08));
        beans.add(new CoffeeBean("Cafe Femenino", "Peru", 24.0, 1.17));
        beans.add(new CoffeeBean("Sidamo", "Ethiopia", 16.0, 0.84));
        beans.add(new CoffeeBean("Huehueutenango", "Guatemala", 12.0, 1.24));
        beans.add(new CoffeeBean("Santa Amalia", "Brazil", 16.0, 1.00));

        double pricePerOunceSum = 0.0;
        for (int i=0 ; i<beans.size() ; ++i) {
            CoffeeBean bean = beans.get(i);
            System.out.println(bean.toString());
            pricePerOunceSum += bean._pricePerOunce;
        }

        System.out.println(
            "Average price/ounce of beans: $"+
            pricePerOunceSum/beans.size());
    }
}
import java.util.ArrayList;

class ArrayListDemo {
    public static void main(String[] args) {
        import java.util.ArrayList so that we can use it.

        ArrayList<CoffeeBean> beans = new ArrayList();
        beans.add(new CoffeeBean("Las Nubes", "Nicaragua", 12.0, 1.08));
        beans.add(new CoffeeBean("Cafe Femenino", "Peru", 24.0, 1.17));
        beans.add(new CoffeeBean("Sidamo", "Ethiopia", 16.0, 0.84));
        beans.add(new CoffeeBean("Huehueutenango", "Guatemala", 12.0, 1.24));
        beans.add(new CoffeeBean("Santa Amalia", "Brazil", 16.0, 1.00));

        double pricePerOunceSum = 0.0;
        for (int i = 0; i < beans.size(); ++i) {
            CoffeeBean bean = beans.get(i);
            System.out.println(bean.toString());
            pricePerOunceSum += bean._pricePerOunce;
        }

        System.out.println("Average price/ounce of beans: ",
                          pricePerOunceSum / beans.size());
    }
}

import java.util.ArrayList;

class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<CoffeeBean> beans = new ArrayList<CoffeeBean>();

        beans.add(new CoffeeBean("Las Nubes", "Nicaragua", 12.0, 1.08));
        beans.add(new CoffeeBean("Cafe Femenino", "Peru", 24.0, 1.17));
        beans.add(new CoffeeBean("Sidamo", "Ethiopia", 16.0, 0.84));
        beans.add(new CoffeeBean("Huehueutenango", "Guatemala", 12.0, 1.24));
        beans.add(new CoffeeBean("Santa Amalia", "Brazil", 16.0, 1.00));

        double pricePerOunceSum = 0.0;
        for (int i = 0 ; i < beans.size() ; ++i) {
            CoffeeBean bean = beans.get(i);
            System.out.println(bean.toString());
            pricePerOunceSum += bean._pricePerOunce;
        }

        System.out.println("Average price/ounce of beans: $" +
                           pricePerOunceSum/beans.size());
    }
}
import java.util.ArrayList;

class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<CoffeeBean> beans = new ArrayList<CoffeeBean>();
        beans.add(new CoffeeBean("Las Nubes", "Nicaragua", 12.0, 1.08));
        beans.add(new CoffeeBean("Cafe Femenino", "Peru", 24.0, 1.17));
        beans.add(new CoffeeBean("Sidamo", "Ethiopia", 16.0, 0.84));
        beans.add(new CoffeeBean("Huehueutenango", "Guatemala", 12.0, 1.24));
        beans.add(new CoffeeBean("Santa Amalia", "Brazil", 16.0, 1.00));

        double pricePerOunceSum = 0.0;
        for (int i=0; i<beans.size(); ++i) {
            System.out.println(bean.toString());
            cartTotal += bean.getCost();
        }
        System.out.println("Average price/ounce of beans: $" + pricePerOunceSum/beans.size());
    }
}

beans starts out empty (beans.size() == 0)

4. Add five new CoffeeBeans(…) to it.

beans.size() will now be 5
import java.util.ArrayList;

class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<CoffeeBean> beans = new ArrayList<>();
        beans.add(new CoffeeBean("Las Nubes", "Nicaragua", 12.0, 1.08));
        beans.add(new CoffeeBean("Cafe Femenino", "Peru", 24.0, 1.17));
        beans.add(new CoffeeBean("Sidamo", "Ethiopia", 16.0, 0.84));
        beans.add(new CoffeeBean("Huehueutenango", "Guatemala", 12.0, 1.24));
        beans.add(new CoffeeBean("Santa Amalia", "Brazil", 16.0, 1.00));

        double pricePerOunceSum = 0.0;
        for (int i=0 ; i<beans.size() ; ++i) {
            CoffeeBean bean = beans.get(i);
            System.out.println(bean.toString());
            pricePerOunceSum += bean._pricePerOunce;
        }

        System.out.println("Average price/ounce of beans: 
$" +
pricePerOunceSum/beans.size());
    }
}
Try it out…

Download collections.zip

Modify ArrayListDemo.java to:

1. compute the cart total (sum of bean.getCost())
2. add another bean: “Chapel Hill Grit” from “USA”, 160 oz @ $0.10/oz.
int[] array = new int[5];
String string = "Hello";
ArrayList<String> arrayList = ...; // .add(...) 5x to it

System.out.println("array.length is "+array.length);
System.out.println("string.length() is "+string.length());
System.out.println("arrayList.size() is "+arrayList.size());

This is likely a historic artifact… the authors of Java cannot change it to one convention without breaking existing code.

Good thing (?) is that the compiler will tell you when you get it wrong…
int[] array = new int[5];
String string = "Hello";
ArrayList<String> arrayList = ...; // .add(...) 5x to it

System.out.println("array[2] is "+array[2]);
System.out.println("string.charAt(2) is "+string.charAt(2));
System.out.println("arrayList.get(2) is "+arrayList.get(2));

This is likely a historic artifact... the authors of Java cannot change it to one convention without breaking existing code.

Good thing (?) is that the compiler will tell you when you get it wrong...
### ArrayList<T> methods

Where T is the placeholder for the element type.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList&lt;T&gt;()</td>
<td>constructor to create an empty ArrayList</td>
</tr>
<tr>
<td>int size()</td>
<td>returns the number of elements in the ArrayList</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>returns true iff size() == 0</td>
</tr>
<tr>
<td>T get(int index)</td>
<td>returns the element at index</td>
</tr>
<tr>
<td>void clear()</td>
<td>removes all elements (size() == 0 after calling clear())</td>
</tr>
<tr>
<td>T remove(int index)</td>
<td>removes the element at index</td>
</tr>
<tr>
<td>boolean add(T obj)</td>
<td>adds obj to the end of the ArrayList increases size() by 1</td>
</tr>
<tr>
<td>T set(int index, T obj)</td>
<td>replaces element at index with obj returns previous element at index</td>
</tr>
</tbody>
</table>
Primitives in Collections

Most of the time you will find that you are using ArrayList<T> where the element type T is as class.

On occasion you may find that you need an ArrayList of “primitive” type, and T cannot be a primitive type…

Recall primitive types are: byte, short, int, long, float, double, char, and boolean

(One way to remember is that primitive types are not capitalized. Another way is that we do not allocated primitive types with “new”)

Primitive Wrappers

class Integer {
    private int _value;

    Integer(int value) {
        _value = value;
    }
}

class Double {
    private double _value;

    Double(double value) {
        _value = value;
    }
}

class Float { ... }
class Short { ... }
class Byte { ... }
class Long { ... }
class Character { ... }
class Boolean { ... }

... are pre-defined classes with a single field with a type (e.g., long) matching the class name (e.g., Long).

exceptions are Integer and Character, which are spelled out for some reason...
Primitives in Collections

```java
ArrayList<Integer> ints = new ArrayList<Integer>();
ArrayList<Double> dbls = new ArrayList<Double>();
ArrayList<Boolean> bools = new ArrayList<Boolean>();
    etc...
```

Primitive wrappers are treated specially by the compiler, so you can use collections without explicitly allocating objects. For example:

```java
    ints.add(5);
    dbls.add(2.5);
    bools.add(false);
```
Primitives in Collections

```java
import java.util.ArrayList;

class PrimitiveArrayList {
    public static void main(String[] args) {
        ArrayList<Double> dbls = new ArrayList<Double>();

        // automatic conversion from double to Double
        dbls.add(1.0);
        dbls.add(2.5);
        dbls.add(4.5);
        dbls.add(7.2);
        dbls.add(11.4);

        for (int i=0 ; i<dbls.size() ; ++i) {
            // automatic conversion from Double to double
            // dbls.get(i) returns a Double
            // assignment converts it to a double
            double value = dbls.get(i);
            System.out.println("dbls.get("+i+") = "+value);
        }
    }
}
```
“for each” loop

Tired of writing “for (int i=0; i<...size() ; ++i)”?

```java
for (int i=0 ; i<beans.size() ; ++i) {
    CoffeeBean bean = beans.get(i);
    System.out.println(bean.toString());
}
```

Can be written instead as:

```java
for (CoffeeBean bean : beans) {
    System.out.println(bean.toString());
}
```

**Advantage:** less typing, fewer lines, readable

**Disadvantage:** no index variable (i) when you need one, can only iterate one way
“for each” loops

Also work for arrays, and a whole variety of collections.
Classes as Containers

What if we wanted our coffee beans to have multiple origins?

```java
import java.util.ArrayList;

class CoffeeBean2 {
    String _name;
    double _ounces;
    double _pricePerOunce;
    ArrayList<String> _origins = new ArrayList<String>();

    CoffeeBean2(String name, double oz, double price) {
        _name = name;
        _ounces = oz;
        _pricePerOunce = price;
    }

    public void addOrigin(String origin) {
        _origins.add(origin);
    }
}
```

Full source in CoffeeBean2.java
Example w/ nested for-each loops!

double pricePerOunceSum = 0.0;
for (CoffeeBean2 bean : beans) {
    System.out.println(bean.toString());
    for (String origin : bean.getOrigins()) {
        System.out.println("    " + origin);
    }
    pricePerOunceSum += bean._pricePerOunce;
}
System.out.println(
    "Average price/ounce of beans: $" +
    pricePerOunceSum/beans.size());

Full source in ArrayListDemo2.java
java.util.Collections

At the top of the file:
import java.util.Collections;

then you can:

Collections.sort(arrayListToSort)
Collections.reverse(arrayListToReverse)
Collections.shuffle(arrayListToShuffle)

Note: in order to sort, the element type must have a natural ordering, or you will have to give it one. We discuss how to make your classes sortable later. For you can try it out on: ArrayList<String>, ArrayList<Integer>, and the rest of the primitive wrappers which do have a natural ordering.
Collections.sort() Example

```java
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> origins = new ArrayList<String>();
for (CoffeeBean2 bean : beans) {
    names.add(bean.getName());
    origins.addAll(bean.getOrigins());
}

System.out.println("Names: "+names);
Collections.sort(names);
System.out.println("Names: "+names);

System.out.println("Origins: "+origins);
Collections.sort(origins);
System.out.println("Origins: "+origins);
```

Full source in ArrayListDemo3.java

Try this: (1) print the names in reverse sorted order.
(2) shuffle the origins.
Tomorrow: Midterm Review

We’ll go through whatever you want to go through.

In absence of questions, I’ll review a bit of everything.