ArrayLists
COMP1400 Week 7

Arrays
Arrays are intrinsically fixed-length storage.
Adding or deleting elements from arrays is cumbersome:
• create a new larger/smaller array
• copy the unaffected elements
• add/remove the affected element

Abstraction
Rather than rewriting methods that perform these actions every time we use an array, we should abstract them into a new object.

What we want is a new class that implements a variable-length list.
The public interface should provide methods to implement the actions listed above.

Working with arrays
Common actions we want to do with arrays:
• adding and deleting
• copying
• looking for a particular element
• counting the elements
**ArrayList**

The ArrayList class in the Java Class Library provides this capability.

http://docs.oracle.com/javase/1.5.0/docs/api/java/util/ArrayList.html

**Encapsulation**

The ArrayList class abstracts and encapsulates all the methods for implementing a variable-length list using fixed-length arrays.

```java
ArrayList<String>
private String[] myList;
public void add(String s);
public String get(int pos);
```

**Generic classes**

Each ArrayList holds a specific type (or class) of object.

This class is given as a type parameter in angle brackets <> after the name.

The type of the methods get, set and add depend on the given type.

Parameterised types are called generic classes.

**Type parameters**

When we create an ArrayList we need to provide the type of objects it contains:

- ArrayList<String> listOfStrings;
- ArrayList<Integer> listOfInts;
- ArrayList<Book> listOfBooks;

These are called the type parameters.
List of Strings

```java
ArrayList<String> classRoll = new ArrayList<String>();
classRoll.add("Sim Mautner");
String who = classRoll.get(0);
// who is now “Sim Mautner”
```

List of ints

This does not work:

```java
ArrayList<int> scores = new ArrayList<int>();
scores.add(1);
int s = scores.get(0);
```

The type parameter must be a class not a primitive type.

List of ints

Instead we use:

```java
ArrayList<Integer> scores = new ArrayList<Integer>();
scores.add(1);
int s = scores.get(0);
```

The Integer class is an object-wrapper for the primitive type int.

Wrapper Classes

Java implements wrapper classes to convert primitive types into objects automatically.

```java
Integer intObject = 1;
Double doubleObject = 2.0;
Boolean boolObject = true;
Character charObject = ‘x’;
```
Importing

Since ArrayLists are part of the JCL, we must first import them before use:

```java
import java.util.ArrayList;
```

Construction

To construct an ArrayList we will usually use the simple no-parameter constructor:

```java
// create an empty list
ArrayList<String> classRoll = new ArrayList<String>();
```

There are two other constructors but we won't be using them much.

Adding items

A newly constructed list is empty.

We can add items using the `add` method.

```java
classRoll.add("Malcolm");
classRoll.add("Claude");
```

The type of objects we add must match the type parameter.

```java
classRoll.add(47); // ERROR
```

Adding duplicates

We can add something to a list more than once.

```java
classRoll.add("Malcolm");
classRoll.add("Malcolm");
```
Size

We can use the `size` method to access the number of objects on the list:

```java
int nStudents = classRoll.size();
// nStudents == 2
classRoll.add("Sim");
nStudents = classRoll.size();
// nStudents == 3
```

Get

We can access elements on the list by their index using the `get` method.

```java
String me = classRoll.get(0);
// me == "Malcolm"
String him = classRoll.get(1);
// him == "Claude"
```

Indexing

Note that the indices always start from 0 and end at size-1.

```java
classRoll.size();
// returns 3
classRoll.get(0);
// returns "Malcolm"
classRoll.get(2);
// returns "Sim"
classRoll.get(3); // ERROR
```

Remove

We can use the `remove` method to remove the element at a given index.

The other elements get renumbered.

```java
classRoll.remove(1);
classRoll.size();
// returns 2
classRoll.get(1); // "Sim"
```
Copying

There is a special constructor for creating a copy of a list:

```java
ArrayList<String> copy = new ArrayList<String>(classRoll);
copy.add("Troy");
```

Other methods

There are other methods you can read about in the API documentation:

- `clear` - remove all elements
- `set` - set value of a given element
- `insert` - insert an element at an index
- `contains` - test if an element is on the list

For-each loop

A `for-each` loop is used to iterate over all the elements of a collection:

```java
for (String who: classRoll) {
    // action performed on each element
    System.out.println(who);
}
```

Arrays vs ArrayLists

```java
String[] arrayOfStrings;
int[] arrayOfInts;

ArrayList<String> listofStrings;
ArrayList<Integer> listofInts;
```
// constructing arrays
arrayOfStrings = new String[5]
arrayOfInts = new int[10]

// constructing lists (empty)
listOfStrings = new ArrayList<String>();
listOfInts = new ArrayList<Integer>();

// measuring size
int arraySize = arrayOfStrings.length;
int listSize = listOfStrings.size();

// getting elements
String s1 = arrayOfStrings[2];
int i1 = arrayOfInts[1];
String s2 =
    listOfStrings.get(2);
int i2 =
    listOfInts.get(1);

// setting elements
arrayOfInts[1] = 40;
listOfStrings.set(2, "Malcolm");
listOfInts.set(1, 40);
### Arrays vs ArrayLists

#### // iterating
```java
for (int i = 0; i < arrayOfStrings.length; i++)
{
    // process arrayOfStrings[i]
}
```

```java
foreach (String s: listOfStrings)
{
    // process s
}
```

### Other collections

Sets (unordered, no duplicates):

- [http://docs.oracle.com/javase/1.5.0/docs/api/java/util/HashSet.html](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/HashSet.html)

Maps (look-up tables with key/value pairs):

- [http://docs.oracle.com/javase/1.5.0/docs/api/java/util/HashMap.html](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/HashMap.html)