Java Programming 2 – Lecture #10 – Jeremy.Singer@glasgow.ac.uk

Java Collections

The Collections framework[1] is a Java standard library. Each Collections class supports the processing of a series of related elements in a common data structure. In this course, we are going to consider the `java.util.ArrayList` class, but there are other kinds of Collections including `Set`, `Stack` and `HashMap`. You will explore some of these data structures in the ADS2 course next semester. The advantage of the Collections framework is that it provides a standardized, reusable implementation for these common data structures.

The Collections framework forms an object-oriented class hierarchy. The base class[2] is `java.util.Collection`[3] which defines methods that all Collections must implement including `add(), remove(), contains(), size()` and `toArray()`.

The ArrayList Data Structure

The major limitation of Java arrays is that they have a fixed length. The `java.util.ArrayList`[4] class is a more flexible (although correspondingly less efficient) Collections class that implements variable length arrays. The backing array grows and shrinks dynamically as elements are added to the `ArrayList`.

`ArrayList` operations are invoked by method calls, rather than by built-in Java syntax (unlike arrays). Below is an example source code snippet:

```java
ArrayList<Integer> nums = new ArrayList<Integer>();
nums.add(1);
nums.add(1);
int i = 2;
int fib = 1;
while (fib < LIMIT && nums.size() < SIZE_LIMIT) {
  fib = nums.get(i-1) + nums.get(i-2);
  nums.add(fib);
}
```

Elements can be removed from an `ArrayList` using the `remove()` call. If an element is removed from the middle of the list, elements to the right are shuffled down.

---

[1] http://docs.oracle.com/javase/7/docs/technotes/guides/collections/overview.html
[2] Actually an interface, but we haven't learnt about these yet.
[3] http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html
[4] See http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

Note that `ArrayList` structures can be converted to arrays using the `toArray()` method, and vice versa using the `java.util.Arrays`[5] helper methods.

Iterating over Collections

Collections are *Iterable*, which means that we can use a `for-each` loop to iterate over Collection elements.

```
ArrayList<String> words = new ArrayList<String>();
nums.add("antidisestablishmentarianism");
nums.add("monosyllabic");
int totalChars = 0;
for (String word : words) {
    totalChars += word.length()
}
System.out.printf("total number of characters: %d\n",
                  totalChars);
```

Generic Types

Collection classes are type-parameterized. The type specified in angle brackets after the Collection class name specifies the type of the elements stored in that Collection. This makes the Collections classes generic, in that they can be used with any type of element. The most general element is java.lang.Object. Note that subclasses of the specified element type are also valid element types. The element type is specified for the declaration and the construction of the Collection – e.g.

```
ArrayList<Object> objects = new ArrayList<Object>();
```

Wrapper Classes

Only object references (i.e. pointers to objects in the heap) can be stored as elements in a Collection. Therefore primitive values cannot be stored directly as Collection elements. To get around this, we use wrapper classes for each primitive. java.lang.Integer is the wrapper class for the int primitive type.

```
int iPrim = 42;
Integer iWrap = new Integer(iPrim);
int x = iWrap.intValue();
```

Question

Strings and Wrapper classes are immutable. What does this mean? Why is it useful?

[5] http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html