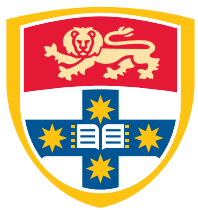
INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney



Lecture 22: The Java Collections Framework

A library of containers

Java Collections

Java comes with many *collections* built in. The *collection classes* include:

- `ArrayList`, `LinkedList`
- `TreeSet`, `HashSet`
- `HashMap`, `TreeMap`

We will look at these basic collections, get a sense of what they have in common and how they differ.

ArrayList

An array is useful for storing many values of the same type

Consider the following situation:

```
1  String[] words = new String[2];
2  int lines = 0;
3  Scanner keyboard = new Scanner(System.in);
4  while (true) {
5      String word = keyboard.next();
6      if (word.equalsIgnoreCase("STOP"))
7          break;
8      words[ lines ] = word;
9      lines++;
10 }
```

What needs to be done to fix this situation?

ArrayList

When we don't know how much data to expect, we should do this every time...

Consider the following:

```
1  String[] words = new String[10];
2  Scanner keyboard = new Scanner(System.in);
3  while (true) {
4      int index = Integer.parseInt(keyboard.next());
5      String word = keyboard.next();
6      if (word.equalsIgnoreCase("STOP"))
7          break;
8      words[ index ] = word;
9  }
```

Firstly, what is missing with the above code?

How many words have been entered? is there a way to accurately count?

ArrayList

We can solve these problems by describing an array with *operators*

ArrayList describes an array to help us avoid repeating code for common activities

```
ArrayList<E> list = new ArrayList<E>();
```

```
ArrayList();
boolean add(E e);
void add(int index, E e);
E get(int index)
boolean isEmpty();
int size();
E remove(int index);
boolean remove(Object o);
void clear();
```

ArrayList

We create an `ArrayList` with the type as follows:

```
1    ArrayList<String> myStrings = new ArrayList<String>();
```

which declares and initialises an `ArrayList` to store `String` objects.

This is just one of several ways of having a *list* of objects; another is the `LinkedList`.

For now you don't need to worry about which one to use: `ArrayList` is fine for most uses.

ArrayList<String>

`ArrayList` is a class in Java that wraps an array in a class, with more methods defined on it.

`ArrayList` can store any kind of `Object` — that is, `String`, `Integer`, even `ArrayList`, but not a primitve type like `int` or `boolean`[1]

Because `ArrayList` can store different kinds of objects, we have to specify what kind of object we're storing when we create it.
This enables the compiler to check whether we're using it the right way.

We say we want to have an `ArrayList` of `Strings` like this:

```
1        ArrayList<Person> L = new ArrayList<Person>();
```

[1] Remember: primitive types have lower case names, and classes have upper case names by convention.

Methods in `ArrayList`

Once we've *constructed* an `ArrayList` to store the type of objects we need, we can use it by calling methods on it like this:

```
1  import java.util.ArrayList;
2  public class AList {
3      public static void main(String[] args) {
4          ArrayList<String> L = new ArrayList<String>();
5          L.add("hello");
6          L.add("goodbye");
7          System.out.println("L has " + L.size() + " elements.");
8          if (L.contains("hello")) {
9              System.out.println("... and is friendly");
10         }
11     }
12 }
```

Methods in `ArrayList` (cont.)

```
~> javac AList.java
~> java AList
L has 2 elements.
... and is friendly
~>
```

Methods in `ArrayList`

and we can also use a really nice feature of Java: the *enhanced for loop*:

```
11        for (String s : L) {
12            System.out.println(s);
13        }
```

What this means is

ENHANCEDFOR (L)

1 **for each** (string $s \in L$) **do**

2 · print s

All you need for this to work is that `L` be *array* or one of the *Collection* types of Java. One of these is `ArrayList`.

Cards with `ArrayList`

Suppose you are making a card game. You are responsible for making the Card player

A card player has a name

A card player holds zero or more cards, each card is represented as text "Spades 4" or "Hearts Queen"

During the game, players can:

- pick up one more card
- put down one card

Write an implementation using an ArrayList that will work with the Test code on the next slide

Cards with ArrayList

```
1  public class CardPlayerTest {
2        public static void main(String[] args) {
3            CardPlayer player1 = new CardPlayer("Dora");
4
5            player1.pickupCard("Spades King");
6            player1.pickupCard("Spades 10");
7            player1.pickupCard("Diamonds 3");
8            player1.pickupCard("Diamonds Jack");
9            player1.pickupCard("Clubs King");
10
11           // Dora's first move
12           player1.placeCard("Clubs King");
13           player1.placeCard("Spades King");
14
15           player1.showCards();
16       }
17 }
```

```
$ java CardPlayerTest
Cards for Dora:
Spades 10
Diamonds 3
Diamonds Jack
```

Sets

No duplicates

TreeSet and HashSet

We may need to describe elements without a given order. Just add items and retrieve them

In a set, elements are not ordered

There are two different implementations of a `Set`: `TreeSet` and `HashSet`.

TreeSet keeps objects in order, if they can be compared sensibly (e.g., like `Strings`, or `Integers`, but not our `IntPair` class)

HashSet doesn't keep things in order, but uses a very fast storage method called *hashing* that is beyond the scope of this course.

If you don't care about the order of objects, they you can use a `HashSet`, but if you do, then you should use a `TreeSet`.

SetDemo

Here's a quick demonstration:

```
1  import java.util.HashSet;
2  import java.util.TreeSet;
3  public class SetDemo {
4     public static void main(String[] args) {
5        HashSet<String> HS = new HashSet<String>();
6        TreeSet<String> TS = new TreeSet<String>();
7        String[] strings = new String[]
8           { "one","fish","two","fish","red","fish","blue","fish" };
9        for (String s : strings) {
10          HS.add(s);
11          TS.add(s);
12       }
13       System.out.println("TreeSet: " + TS);
14       System.out.println("HashSet: " + HS);
15    }
16 }
```

SetDemo (cont.)

Which yields

```
~> javac SetDemo.java
~> java SetDemo
TreeSet: [blue, fish, one, red, two]
HashSet: [two, red, blue, fish, one]
~>
```

Note how the order in the two sets is different; the `TreeSet` has stored them in the "right" order but the `HashSet` seems to be just any-old-how.

CardPlayer revisited

We now consider using a Set to describe the players cards

A new rule says that the player cannot pick up the same card that they already have

Write an alternative implementation of the CardPlayer class using the HashSet collection

Maps

Storing Key - Value pair

The `Map` class

A *Map* is also called an *associative array* or a *dictionary* in other programming languages.

A map keeps associations between *key* and *value* objects.

Mathematically speaking, a map is a *function* from one set, the *key set*, to another set, the *value set*

Every key in a map has a unique value: you can't map to more than one thing!

On the other hand, any value may be associated with several keys.

Implementations of Map

In Java there are two kinds of `Maps`, the `HashMap` and the `TreeMap`.

A `HashMap` is one way of implementing the Map, in which access is very fast (it doesn't matter really how big your collection is).

> In fact it is essentially *constant time*: the amount of time taken to check whether an item is in a hash-based container is independent of how many items are in the container.

The `TreeMap` implementation of the Map interface keeps (key,value) pairs in order of their *keys*.
It takes an amount of time that is approximately the logarithm of the number of items in the Map to find a given item.

The Great Big Collection Collection

First we'll have a simple `Number` class. It doesn't do much but it will serve to show that you can use these Collections on your own classes.

```
1  public class Number implements Comparable<Number> {
2      private int num;
3      public Number(int x) {
4          num = x;
5      }
6      public String toString() {
7          return "(" + num + ")";
8      }
9      @Override
10     public int compareTo(Number o) {
11         return num - o.num;
12     }
13 }
```

The Great Big Collection Collection (cont.)

Next we will look at a collection of the Java Collections:

```
1  import java.util.ArrayList;
2  import java.util.HashMap;
3  import java.util.HashSet;
4  import java.util.List;
5  import java.util.Map;
6  import java.util.Set;
7  import java.util.TreeMap;
8  import java.util.TreeSet;
9
10 public class UsingCollections {
```

The Great Big Collection Collection (cont.)

We will need some String objects from a great work of literature:

```
10 public class UsingCollections {
11
12     private static String[] rawStrings = new String[] { "one",
13             "fish", "two", "fish", "red", "fish", "blue", "fish" };
```

The Great Big Collection Collection (cont.)

We will create some collection variables:

```
16        List<String> stringArrayList = new ArrayList<String>();
17        Set<String> stringHashSet = new HashSet<String>();
18        Set<String> stringTreeSet = new TreeSet<String>();
19        Set<Number> numberTreeSet = new TreeSet<Number>();
20        Map<String, Number> stringToNumHashMap
21              = new HashMap<String, Number>();
22        Map<String, Number> stringToNumTreeMap
23              = new TreeMap<String, Number>();
```

The Great Big Collection Collection (cont.)

Now fill them up:

```
26          int n = 3;
27          for (int i = 0; i < rawStrings.length; i++) {
28                                  String s = rawStrings.get(i);
29              stringArrayList.add(s);
30              stringHashSet.add(s);
31              stringTreeSet.add(s);
32              Number num = new Number(n);
33                                  n--;
34              numberTreeSet.add(num);
35              stringToNumHashMap.put(s, num);
36              stringToNumTreeMap.put(s, num);
37          }
```

The Great Big Collection Collection (cont.)

And print them out:

```
36          System.out.println("stringArray: " + stringArrayList);
37          System.out.println("stringSet: " + stringHashSet);
38          System.out.println("sortedStringSet: " + stringTreeSet);
39          System.out.println("sortedPairSet: " + numberTreeSet);
40          System.out.println("mapOfPairs: " + stringToNumHashMap);
41          System.out.println("orderedMap: " + stringToNumTreeMap);
```

```
stringArray: [one, fish, two, fish, red, fish, blue, fish]
stringSet: [two, red, blue, fish, one]
sortedStringSet: [blue, fish, one, red, two]
sortedPairSet: [(-5), (-4), (-3), (-2), (-1), (0), (1), (2)]
mapOfPairs: {two=(0), red=(-2), blue=(-4), fish=(-5), one=(2)}
orderedMap: {blue=(-4), fish=(-5), one=(2), red=(-2), two=(0)}
```

Summary

- The Java Collections Framework has a range of containers of different types;
- All the collections can store any kind of reference type;
- All the collections can use the enhanced for loop (or “for each” loop);
- Each major type of collection is an interface;
- There are different implementations of those interfaces;
- The implementations have different properties and performance.