Rigidity of arrays

- Length of array is fixed at creation time.
- Can't be expanded.
- Can't be shrunk.
- Arrays are part of Java language — uses special syntax.
- E.g., `myArray[i]` for accessing the ith element.

ArrayList

**ArrayList:**
- Can grow and shrink as needed;
- provides methods for inserting and removing elements.

**Declaration**

```java
ArrayList<String> cheers = new ArrayList<String>();
```

- This is an array list of strings; counterpart to `String[]`.
- Angle brackets indicate that `String` is a type parameter.
- Can replace `String` with e.g. `HotelRoom` to get different array list type.
- In general: use `ArrayList<E>` to collect objects of type `E`, but `E` cannot be a primitive type.

**ArrayList: Methods**

- A newly constructed `ArrayList` has size 0.
- `ArrayList` has various methods, which allow us to:
  - keep on adding new elements;
  - remove elements.
- The size changes after each addition / removal.
**ArrayList: Adding**

**Adding Elements**

```java
ArrayList<String> cheers = new ArrayList<String>();
cheers.add("hip");
cheers.add("hip");
cheers.add("hooray");
int n = cheers.size(); // n gets value 3
```

- `add()` appends each element to the end of the list.

**Printing an ArrayList**

```java
System.out.println(cheers);
```

**Output**

```
[hip, hip, hooray]
```

**ArrayList: More methods**

**Index of first occurrence**

```java
int ind = cheers.indexOf("hip"); // ind gets value 0
```

**Adding element at an index**

```java
cheers.add(1, "hop"); // 2nd "hip" gets shunted along
```

**Elements of cheers:** "hip", "hop", "hip", "hooray"

**ArrayList and Loops**

**Looping over ArrayList:**

**Standard for loop**

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

**Enhanced for again**

```java
for (String s : cheers) {
    System.out.println(s);
}
```
ArrayList and Loops

Enhanced for again

```java
for (String s : cheers) {
    System.out.print(s + " has index: ");
    System.out.println(cheers.indexOf(s));
}
```

Output

hop has index: 0
hip has index: 1
hooray has index: 2

Wrapper Classes

Wrapper Classes:

- The type variable E in a generic type like `ArrayList<E>` must resolve to a reference type.
- So `ArrayList<int>` will not compile.
- All the primitive types can be turned into objects by using wrapper classes:

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
</tbody>
</table>

NB Wrapper class names are always capitalized, always complete words.

Auto-boxing

- Conversion between primitive types and corresponding wrapper classes is automatic.
- Process of conversion is called auto-boxing

Auto-box example

```
Double batteryCharge = 2.75;
double x = batteryCharge;
```

Auto-box example

```
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);
double x = data.get(0);
```

Auto-boxing, contrast

- Automatic vs explicit conversion between primitive types and corresponding wrapper classes.

Auto-box example

```
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);
double x = data.get(0);
```

Contrast...

```
data.add(Double.valueOf(29.95));
Double tmp = data.get(0);
double x = tmp.doubleValue();
```

Using Double class’ methods:

```java
double doubleValue() Return double value of object.
Double valueOf(double d) Return Double value of specified double.
```
Import

Importing:
▶ To get full access to Java API, we need to import classes.
▶ Not necessary if class is in same folder, or part of java.lang (e.g., Math library).
▶ To use ArrayList, add the appropriate import statement at top of your file:

Import example

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

Import example — Wrong!

```java
import java.util.ArrayList<String>; // Don’t use parameter
```

More ArrayList Goodies

▶ Sorting an arraylist, vs sorting an array. Multiple ways, e.g,

Array

```java
import java.util.Arrays;
...
int[] myNums = new int[] {1, 3, 7, 5, 10};
Arrays.sort(myNums);
System.out.println(Arrays.toString(myNums)); // [1, 3, 5, 7, 10]
```

ArrayList

```java
import java.util.ArrayList;
import java.util.Collections;
...
ArrayList<Integer> myNums<Integer> = new ArrayList<Integer>();
myNums.add(1); myNums.add(5); myNums.add(3);
Collections.sort(myNums);
System.out.println(myNums); // [1, 3, 5]
```

More ArrayList Goodies

▶ Convert between Array and Arraylist

Array to ArrayList

```java
import java.util.Collections;
import java.util.ArrayList;
...
Integer[] myNums = new Integer[] {1, 3, 7, 5, 10};
ArrayList<Integer> myNums2 = new ArrayList<Integer>();
Collections.addAll(myNums2, myNums);
System.out.println(myNums2); // [1, 3, 7, 5, 10];
```

ArrayList to Array

```java
...
Integer[] myNums3 = myNums2.toArray(new Integer[myNums2.size()]);
System.out.println(Arrays.toString(myNums)); // [1, 3, 7, 5, 10]
```

Java API

Look at sample Javadoc web page.
http://docs.oracle.com/javase/8/docs/api/
Java API: Top Node

Maps

Associative Arrays

Associative array:
- Associates a collection of unique keys with values.
- Ordinary arrays: keys can only be integers.
- Associative arrays allow keys of many types, most notably strings.
- Examples:
  1. Given a person’s name, look up a telephone number.
  2. Given an internet domain, look up its IP address.
  3. Given a geo-location, look up its GPS coordinates.
  4. Given a word, look up its frequency in a text.
- Relationship between key and value: mapping.

Java: associative arrays are implemented by type HashMap.
Map People to their Matric Nos.

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>s0189034</td>
</tr>
<tr>
<td>Michael</td>
<td>s0289125</td>
</tr>
<tr>
<td>Helen</td>
<td>s0378435</td>
</tr>
<tr>
<td>Mary</td>
<td>s0412375</td>
</tr>
<tr>
<td>John</td>
<td>s0456782</td>
</tr>
</tbody>
</table>

Map Words to Length

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;this&quot;</td>
<td>4</td>
</tr>
<tr>
<td>&quot;is&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;the&quot;</td>
<td>3</td>
</tr>
<tr>
<td>&quot;time&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;and&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Map People to their Matric Nos: Wrong!

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>s0189034</td>
</tr>
<tr>
<td>Michael</td>
<td>s0289125</td>
</tr>
<tr>
<td>Peter</td>
<td>s0378435</td>
</tr>
<tr>
<td>Mary</td>
<td>s0412375</td>
</tr>
<tr>
<td>John</td>
<td>s0456782</td>
</tr>
</tbody>
</table>

NB Keys must be unique.

Map People to their Telephone Nos: Wrong!

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>504455</td>
</tr>
<tr>
<td>Michael</td>
<td>502331</td>
</tr>
<tr>
<td>Helen</td>
<td>509800</td>
</tr>
<tr>
<td>Mary</td>
<td>506666</td>
</tr>
<tr>
<td>John</td>
<td>501235</td>
</tr>
</tbody>
</table>

▶ A given key can only be mapped to one value.
▶ However, type of value can be array, or some other object.
HashMap

Import HashMap

import java.util.HashMap;

Declare HashMap

HashMap<String, Integer> map
= new HashMap<String, Integer>();

- HashMap takes two type parameters.
- Here, String is type of key, Integer is type of value.

HashMap: Add and retrieve mappings

- put(Key, Value): put Value as the value of Key in wordLengths.
  ```java
  HashMap<String, Integer> wordLengths = new HashMap<String, Integer>();
  for (String word : words) {
      wordLengths.put(word, word.length());
  }
  ```
  
  - get(Key): get the value of Key in wordLengths.
    ```java
    int wl = wordLengths.get("record"); // value is 6
    ```

Mapping Words to their Lengths

Goal: Given a string of words, derive an associative array that maps each word to its length.

1. Split the string on whitespace, to yield words.
2. For each word \( w \), add it as a key, and associate it with value \( w.length() \).
3. When we add the same key again, we overwrite the previous association — wasteful but harmless in this case.

split() method of String

String sent = "this is the time and this is the record of the time";
String[] words = sent.split(" "); // split on whitespace

HashMap: Add and retrieve mappings

wordLengths.keySet(): the set of keys in wordLengths.
[of, record, time, is, the, this, and]

wordLengths.values(): the collection of values in wordLengths.
[2, 6, 4, 2, 3, 4, 3]
HashMap: Add and retrieve mappings

wordLengths.keySet(): the set of keys in wordLengths.

[of, record, time, is, the, this, and]

Q: How do we list all key-value pairs in a map?
A: Loop over the set of keys.

```java
for (String key : wordLengths.keySet()) {
    System.out.printf("%s => %s\n", key, wordLengths.get(key));
}
```

Output:

```
of => 2
record => 6
time => 4
is => 2
the => 3
this => 4
and => 3
```

HashMap: Printing

```java
System.out.println(wordLengths);
```

Output:

```
{of=2, record=6, time=4, is=2, the=3, this=4, and=3}
```

Format is `{Key1=Value1, Key2=Value2, ... }

HashMap: More methods

Check if a key is present?

```java
boolean b = wordLengths.containsKey(“Hello”);
```

Remove a key value pair

```java
wordLengths.remove(“and”);
```

Check number of mappings present

```java
int s = wordLengths.size();
```

ArrayList & HashMap

▶ Use ArrayList when you want your arrays to be able to grow, or you want to easily insert and remove items in the middle of an array.

▶ Use HashMap when you want to use keys other than a predetermined list of integers.

▶ For more on ArrayList and HashMap, look at the Java API:

http://docs.oracle.com/javase/8/docs/api/
Java Tutorial
pp219-226, i.e. Chapter 7 Generics, stopping at Generic Methods.
pp423-505, i.e. Chapter 12 Collections, stopping at Algorithms.
In both cases, the book sections contain more material than we talked about. Remember the exam is open book! I don't expect you to remember all the different kinds of collections, but I do expect you to be able to look them up and use them.