Review: ArrayList class

- To use the methods of ArrayList, we need to create an instance of an ArrayList.
- ArrayList has methods that allow us to add, remove, change elements in the list.
- ArrayList supports dynamic arrays that can grow as needed.
Review: ArrayList class

- To use the methods of ArrayList, we need to create an instance of an ArrayList.
- ArrayList has methods that allow us to add, remove, change elements in the list.
- ArrayList supports dynamic arrays that can grow as needed.

**Iterator**

Can use an Iterator class's hasNext() and next() methods to process ArrayList
Standard Java arrays

- An array is a logical, homogenous collection of data elements, grouped together under a common variable name and referenced individually by position number.
Standard Java arrays

- An **array** is a logical, homogenous collection of data elements, grouped together under a common variable name and referenced individually by position number.
- An array in Java is a **reference type** since it is an object.
Standard Java arrays

- An **array** is a logical, homogenous collection of data elements, grouped together under a common variable name and referenced individually by position number.
- An array in Java is a **reference type** since it is an object.
- To refer to a particular location or element in the array, we specify the name of the array and the position number of the particular element in the array.
Declaring Arrays

- Arrays occupy a fixed amount of space in memory.
- The programmer specifies the type of each element and the number of elements required by each array so that the compiler may reserve the appropriate amount of space in memory for the array.
Arrays

This array is called `c` and would be declared (without the values in the array) like this:

```java
int c[] = new int [12];
```
or like this:

```java
int[] c = new int [12];
```
or like this:

```java
int c[];
c = new int [12];
```
or like this:

```java
int[] c;
c = new int [12];
```

It’s preferred to use the `int[]` notation since brackets identify the variable as an array; easier code to read.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c[0]</td>
<td>-45</td>
</tr>
<tr>
<td>c[1]</td>
<td>6</td>
</tr>
<tr>
<td>c[2]</td>
<td>0</td>
</tr>
<tr>
<td>c[3]</td>
<td>72</td>
</tr>
<tr>
<td>c[4]</td>
<td>1543</td>
</tr>
<tr>
<td>c[5]</td>
<td>-89</td>
</tr>
<tr>
<td>c[6]</td>
<td>0</td>
</tr>
<tr>
<td>c[7]</td>
<td>62</td>
</tr>
<tr>
<td>c[8]</td>
<td>-3</td>
</tr>
<tr>
<td>c[9]</td>
<td>1</td>
</tr>
<tr>
<td>c[10]</td>
<td>6453</td>
</tr>
</tbody>
</table>
Arrays

- This array contains 12 elements.
- Arrays are 0-based in Java, meaning the first element is always element 0.
- An element can be referred to by giving the name of the array followed by the position of the element in square brackets [].
- First element is element 0 and is coded as c[0]. The second element is c[1], the seventh c[6], etc. The i\text{th} element of array c is referenced as c[i-1].
- Array names are just variable names.
To fill this array with the values seen here, the following code would work:

```c
    c[0] = -45;
c[1] = 6;
c[2] = 0;
c[3] = 72;
c[4] = 1543;
c[5] = -89;
c[6] = 0;
c[7] = 62;
c[8] = -3;
c[9] = 1;
c[10] = 6453;
```
Arrays

- The position within square brackets is called the subscript.
- Subscripts must be an `int`, an `int` expression or a value of a type that can be promoted to `int`, such as `byte`, `short` or `char`.
- You can use expressions to refer to subscripts, such as:

```
int a = 5, b = 6;
c[a+b] += 2;
```
- This adds 2 to the value of the array element `c[11]`.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c[0]</td>
<td>-45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[1]</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[2]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[3]</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[4]</td>
<td>1543</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[5]</td>
<td>-89</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[6]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[7]</td>
<td>62</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[8]</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[9]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c[10]</td>
<td>6453</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays

- To print the sum of the first two elements:

```
System.out.println(c[0] + c[1]);
```

- To divide the value of the eighth element of c by 2 and assign value to frog:

```
int frog = c[7] / 2;
```

- To test the value of the last element in an if structure:

```
if ( c[11] < 100 )
    System.out.println("True!");
```

<table>
<thead>
<tr>
<th>c[0]</th>
<th>-45</th>
</tr>
</thead>
<tbody>
<tr>
<td>c[1]</td>
<td>6</td>
</tr>
<tr>
<td>c[2]</td>
<td>0</td>
</tr>
<tr>
<td>c[3]</td>
<td>72</td>
</tr>
<tr>
<td>c[4]</td>
<td>1543</td>
</tr>
<tr>
<td>c[5]</td>
<td>-89</td>
</tr>
<tr>
<td>c[6]</td>
<td>0</td>
</tr>
<tr>
<td>c[7]</td>
<td>62</td>
</tr>
<tr>
<td>c[8]</td>
<td>-3</td>
</tr>
<tr>
<td>c[9]</td>
<td>1</td>
</tr>
<tr>
<td>c[10]</td>
<td>6453</td>
</tr>
</tbody>
</table>
1. To declare an array of 12 int elements:
   
   ```java
   int[] c = new int[12];
   ```

2. To declare two float arrays of differing sizes:
   
   ```java
   float[] fred = new float[668],
   barney = new float[15];
   ```
   
   or
   
   ```java
   float[] fred, barney;
   fred = new float[668];
   barney = new float[15];
   ```

3. To declare some String arrays:
   
   ```java
   final int SIZE = 1000;
   String x[] = new String[SIZE],
   y[] = new String[SIZE*10];
   ```
Declaring Arrays

Arrays can be initialized using a comma-separated initializer list.

```c
int n[] = {32, 27, 64, 19, 95, 14, 90, 70, 37};
```
Declaring Arrays

- Arrays can be initialized using a comma-separated initializer list.
- `int n[] = {32, 27, 64, 19, 95, 14, 90, 70, 37}.`
- The size of the array is inferred from the number of elements in the initializer list.
Processing Arrays

- To iterate through an array, we use a loop.
- To get the number of elements (size) in the array:
  
  ```java
  arrayName.length;
  ```

```java
int i = 0;
int n [] = new int [100];
while(i < n.length)
{
    n = i*10;
}
```

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
</tbody>
</table>