Computational Expression

Arrays, Do While Loop, For Loop

Janyl Jumadinova

18-20 November, 2019
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.
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
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 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 \texttt{c[0]}. The second element is \texttt{c[1]}, the seventh \texttt{c[6]}, etc. The \texttt{i}th element of array \texttt{c} is referenced as \texttt{c[i-1]}.
- Array names are just variable names.

| \texttt{c[0]} | -45 |
| \texttt{c[1]} |  6  |
| \texttt{c[2]} |  0  |
| \texttt{c[3]} |  72 |
| \texttt{c[4]} |1543 |
| \texttt{c[5]} | -89 |
| \texttt{c[6]} |  0  |
| \texttt{c[7]} |  62 |
| \texttt{c[8]} | -3  |
| \texttt{c[9]} |  1  |
| \texttt{c[10]}| 6453|
| \texttt{c[11]}|  78 |
Arrays

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:

  ```java
  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>
</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

- To print the sum of the first two elements:
  
  ```java
  System.out.println (c[0] + c[1]);
  ```

- To divide the value of the eighth element of `c` by 2 and assign value to `frog`:
  
  ```java
  int frog = c[7] / 2;
  ```

- To test the value of the last element in an `if` structure:
  
  ```java
  if ( c[11] < 100 )
    System.out.println ("True!");
  ```
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.
- `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.
To iterate through an array, we use a loop.

To get the number of elements (size) in the array:
```
arrayName.length;
```

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

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<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>
Review: Repetition structures

while() loop

- while (condition) { action(s) }

Figure: Flowchart of while loop
Review: Repetition structures

while() loop

- while (condition) { action(s) }

Example:
```java
int num = 10;
while(num < 10) {
    System.out.print(num + " ");
    num++;
}
```

Output:
Review: Repetition structures

while() loop

- while (condition) { action(s) }
  - Example:

```java
int num = 10;
while(num < 10) {
    System.out.print(num + " ");
    num++;
}
```
Review: Repetition structures

while() loop

- while (condition) { action(s) }

- Example:
  ```java
  int num = 10;
  while(num < 10) {
    System.out.print(num + " ");
    num++;
  }
  ```

- Output:
Repetition structures

**do**{} **while**() loop

- do {action(s)} while(condition);

![Flowchart of do...while loop](image-url)
Repetition structures

do{ } while() loop
  do {action(s) } while(condition);
Repetition structures

do{ } while() loop

- do {action(s) } while(condition);

- Example:

  int num = 10;
  do {
    System.out.print(num + " ");
    num++;
  }
  while(num < 10);
Repetition structures

do{} while() loop
  do {action(s) } while(condition);
  - Example:
    int num = 10;
    do {
       System.out.print(num + " ");
       num++;
    }
    while(num < 10);
  
  Output: 10
For repetition structure

```
for ( init; testForFinal; modification ) { actionStatement(s); }
```

- The `init` section should create and initialize your loop control variable.
- The `testForFinal` section should be a Boolean expression to test for the final value.
- The `modification` section should be an arithmetic statement (usually increment `++` or decrement `--` action) to modify the loop control variable.
For loop

public class ForLoop {
    public static void main ( String args[] ) {
        for ( int counter = 1; counter <= 10; counter++)
            System.out.println ( counter );
    }
}