Programming Fundamentals I
Java Fundamentals

Nguyen Ngoc Minh

Saigon Institute of Technology
Readings

• Readings for these lecture notes
  • Textbook:
    • Tony Gaddis & Godfrey, *Starting Out with Java From Control Structures through Data Structures*
  • Reference:
    • Cay S. Horstmann and Gary Cornell, *Core Java™*

• These lecture notes contain material © Sun, Tony Gaddis, and Godfrey Muganda, 2007
Chapter Objectives

Upon completion of this chapter, you should be able to:

• Use arrays of many data types including primitive, class.
• Declare and use multidimensional array.
• Perform operations on elements of array.
Outline

• Array
• Two-Dimensional Arrays
• ArrayList
Array

• Primitive variables hold only one value at a time.
• Arrays refer to a group of objects by a common name.
• An array is a list of data elements that are indexed.
• Elements in an array must have the same data type. The data type can be any type: primitive or class.
There’re ways to declare an array:

- Put the bracket [ ] right after data type:

```java
int[] a;
char[] b;
String[] c;
Rectangle[] d;
```

- Put the bracket [ ] right after variable name:

```java
int a[];
char b[];
String c[];
Rectangle d[];
```
Array
Declaring Array

- Multiple arrays can be declared on the same line.
  \[
  \text{int[]} \ \text{numbers, codes, scores;}
  \]
- With the alternate notation each variable must have brackets.
  \[
  \text{int numbers[], codes[], scores;}
  \]
- The scores variable in this instance is simply an int variable.
Array
Creating Array

- An array is an object. Declaring an array just create a reference.
- To create an concrete array in memory, use `new` keyword. Example:

```java
int[] numbers;
numbers = new int[6];
```

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>index 0</td>
<td>index 1</td>
<td>index 2</td>
<td>index 3</td>
<td>index 4</td>
<td>index 5</td>
</tr>
</tbody>
</table>

Array element values are initialized to 0.
Array indexes always start at 0.
Array
Creating Array

- We can combine declaring and creating an array into one statement.
  ```
  int[] numbers = new int[6];
  ```
- When created, elements in array are initialized to default values (like data fields of object).
- Arrays may be of any type, not just int.
  ```
  boolean[] b = new boolean[6];
  ```

<table>
<thead>
<tr>
<th>false</th>
<th>false</th>
<th>false</th>
<th>false</th>
<th>false</th>
<th>false</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>index 0</td>
<td>index 1</td>
<td>index 2</td>
<td>index 3</td>
<td>index 4</td>
<td>index 5</td>
<td></td>
</tr>
</tbody>
</table>
**Array**

Creating Array (Example)

```java
String[] s;
s = new String[6];
```

Array element values are initialized to 0.
Array indexes always start at 0.
Array
Creating Array

- The array size must be a non-negative number.

```
int[] numbers;
numbers = new int[6];
```

- Once created, an array size is fixed and cannot be changed.
- You can use the same reference variable to refer to an entirely new array.
Array
Accessing elements of array

• An array is accessed by:
  • the reference name
  • An index of the element in the array we want to access

• Example:

```java
int[] numbers = new int[6];
numbers[0] = 20;
numbers[1] = numbers[0] * 20;
```

<table>
<thead>
<tr>
<th>20</th>
<th>400</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>
Array
Example (ArrayDemo1.java)

```java
public static void main(String[] args) {
    int[] hours = new int[3];
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter the hours worked by 3 employees");
    System.out.print("Employee 1: ");
    hours[0] = keyboard.nextInt();
    System.out.print("Employee 2: ");
    hours[1] = keyboard.nextInt();
    System.out.print("Employee 3: ");
    hours[2] = keyboard.nextInt();
    System.out.println("The hours you entered are: ");
    System.out.println(hours[0]);
    System.out.println(hours[1]);
    System.out.println(hours[2]);
}
```
public static void main(String[] args) {
    final int MAX = 3;
    int[] hours = new int[MAX];
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter the hours worked by " + MAX + " employees");

    for(int i=0;i<MAX;i++){
        System.out.print("Employee " + (i +1) + ": ");
        hours[i] = keyboard.nextInt();
    }

    System.out.println("The hours you entered are: ");
    for(int i=0;i<MAX;i++)
        System.out.println(hours[i]);
}
Array
Array Bounds

- Elements in an array is accessed through an index.
- The minimum index is 0, the maximum index is (array size -1)
  
  ```java
  int values = new int[10];
  ```

  - This array would have indexes 0 through 9.
  - In for loops, it is typical to use i, j, and k as counting variables.
Array
Array Bounds (Example)

public class InvalidSubscript {

    public static void main(String[] args) {
        int[] values = new int[3];
        for(int i=0 ; i < 4 ; i ++)
            values[i] = 10;
    }
}

- This code access an element whose index is out of bound.
Array
Array Bounds (Example)

```java
public class InvalidSubscript {
    public static void main(String[] args) {
        int[] numbers = new int[100];
        for (int i = 1; i <= 100; i++)
            numbers[i] = 99;
    }
}
```

- This code would throw an `ArrayIndexOutOfBoundsException`. 
Array Initialization

- When relatively few items need to be initialized, an initialization list can be used to initialize the array.

```java
int[] days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

- The numbers in the list are stored in the array in order:
  - `days[0]` is assigned 31,
  - `days[1]` is assigned 28,
  - `days[2]` is assigned 31, …
Array
Processing Array Content

- Processing data in an array is the same as any other variable.

  \[ \text{grossPay} = \text{hours}[3] \times \text{payRate}; \]

- Pre and post operators will affect to elements of an array:

  \[ \text{int[]} \text{ score} = \{7, 8, 9, 10, 11\}; \]
  \[ \text{++score}[2]; \text{// Pre-increment operation} \]
  \[ \text{score}[4]++; \text{// Post-increment operation} \]
Array
Processing Array Content

- Array elements can be used in relational operations:

  ```
  if(cost[20] < cost[0])
  {
    //statements
  }
  ```

- They can be used as loop conditions:

  ```
  while(value[count] != 0)
  {
    //statements
  }
  ```
Arrays are objects and provide a public field named length that is a constant that can be tested.

```java
double[] temperatures = new double[25];
```

- The length of this array is 25.
- The length of an array can be obtained via its length constant.

```java
int size = temperatures.length;
```

- The variable size will contain 25.
Array

Array Size

- The length constant can be used in a loop to provide automatic bounding.

Index subscripts start at 0 and end at one less than the array length.

```java
for(int i = 0; i < temperatures.length; i++) {
    System.out.println("Temperature "+ i ": "+ temperatures[i]);
}
```
Array
Reassigning Array References

• An array reference can be assigned to another array of
  the same type.

// Create an array referenced by the numbers
variable.

    int[] numbers = new int[10];

// Reassign numbers to a new array.

    numbers = new int[5];

• If the first (ten element) array no longer has a reference
to it, it will be garbage collected.
Array
Reassigning Array References

The `numbers` variable holds the address of an `int` array.

```java
int[] numbers = new int[10];
```
Array

Reassigning Array References

The `numbers` variable holds the address of an `int` array.

```
numbers = new int[5];
```

This array gets marked for garbage collection.
Array
Copying Array

- This is not the way to copy an array.
  ```
  int[] array1 = { 2, 4, 6, 8, 10 };  
  int[] array2 = array1; // This does not copy array1.
  ```
Array

Copying Array

- You cannot copy an array by merely assigning one reference variable to another.
- You need to copy the individual elements of one array to another.

```java
int[] firstArray = {5, 10, 15, 20, 25};
int[] secondArray = new int[5];
for (int i = 0; i < firstArray.length; i++)
    secondArray[i] = firstArray[i];
```

- This code copies each element of firstArray to the corresponding element of secondArray.
Array
Passing Array Elements to a Method

- When a single element of an array is passed to a method it is handled like any other variable.

```java
public class PassElements {

    public static void main(String[] args) {
        int[] numbers = {5,10,15,20,25,30,35,40};
        for(int i=0 ; i < numbers.length ; i++)
            showValue(numbers[i]);
    }

    private static void showValue(int n) {
        System.out.print( n + " ");
    }

}
```
Arrays are objects.
Their references can be passed to methods like any other object reference variable.

```java
public static void showArray(int[] array) {
    for (int i = 0; i < array.length; i++)
        System.out.print(array[i] + " ");
}
```

```java
showArray(numbers);
```

```
5 10 15 20 25 30 35 40
```
public class PassArrayAsArgument {

    public static void main(String[] args) {
        int[] numbers = {1,2,3,4,5,6,7,8};
        squareArrayElement(numbers);
        for(int i=0 ; i < numbers.length ; i++)
            showValue(numbers[i]);
    }

    private static void showValue(int n) {
        System.out.print( n + "   ");
    }

    private static void squareArrayElement(int[] numbers) {
        for(int i=0 ; i < numbers.length ; i++)
            numbers[i] *= numbers[i];
    }
}

Output: 1 4 9 16 25 36 49 64
Array
Comparing Arrays

• When two variables are primitive types, == operator compare the value.
• When applied to class type, == operator only compare address. It compares if two references points to the same address.
• We must compare each element of two arrays to check if the content of two arrays is the same.
public class CompareArrayByOperator {

    public static void main(String[] args) {
        int[] array1 = {1,2,3,4,5,6,7,8};
        int[] array2 = {1,2,3,4,5,6,7,8};
        if(array1 == array2)
            System.out.println("They are equal");
        else
            System.out.println("They are different");
    }
}

• Output: They are different
public class CompareArrayByCheckingElements {

    public static void main(String[] args) {
        int[] array1 = {1,2,3,4,5,6,7,8};
        int[] array2 = {1,2,3,4,5,6,7,8};
        boolean arraysEqual = true;
        if (array1.length != array2.length)
            arraysEqual = false;
        else
        {
            for(int i = 0 ; i < array1.length ; i++)
                if (array1[i] != array2[i]){
                    arraysEqual = false;
                    break;
                }
        }
        if (arraysEqual)
            System.out.println("The arrays are equal.");
        else
            System.out.println("The arrays are not equal.");
    }
}
Array
Useful Array Operations

- Finding the Highest Value in an array.

```java
public class temp {
    public static void main(String[] args) {
        int[] numbers = new int[50];
        int highest = numbers[0];
        for (int i = 0; i < numbers.length; i++) {
            if (numbers[i] > highest)
                highest = numbers[i];
        }
    }
}
```
Array
Useful Array Operations

• Exercise: Finding the Lowest Value in an array.
Array

Useful Array Operations

• Summing Array Elements:

```java
public class temp {

    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8};
        int total = 0;
        for (int i = 0; i < numbers.length; i++)
            total += numbers[i];
    }
}
```
Array
Useful Array Operations

- Exercise: Finding the Average of Array Elements:
Array
Useful Array Operations (Exercise)

• GPA.java
  • Write a GPA class. The data field is an array whose elements are GPAs of students.
  • This class has public methods to return the total, average, highest, lowest GPA.

• GPA_Demo.java
  • Declare a variable whose data type is GPA.
  • Get input data from the user and demo methods of GPA class
The Array class has a static method named sort that will sort a numeric array in *ascending order*.

```java
Arrays.sort(numbers);
```

To use the class

```java
import java.util.Arrays;
```
Array
Sorting an Array (Increasing order)

```java
public class temp {

    public static void main(String[] args) {
        int[] numbers = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
        Arrays.sort(numbers);
        for (int i = 0; i < numbers.length; i++) {
            System.out.print(numbers[i] + " ");
        }
    }
}
```
Array

Sorting an Array (Decreasing order)

```java
import java.util.Arrays;
import java.util.Collections;

public class SortArrayDecreasingOrder {
    public static void main(String[] args) {
        Integer [] numbers = {9,8,7,6,5,4,3,2,1,0};
        Arrays.sort(numbers,Collections.reverseOrder());
        for (int i = 0; i < numbers.length; i++)
            System.out.print(numbers[i] + " ");
    }
}
```

- The class type of elements in array must implement `Comparable` interface so that it can be sorted in **descending** order
Array Enhanced for loop

for(datatype elementVariable : array)
    statement;

int[] numbers = {3, 6, 9};
for(int val : numbers)
{
    System.out.println("The next value is " + val);
}
Array
Partially Filled Arrays

- Typically, if it is unknown how much data an array will be holding:
  - size the array to the largest expected number of elements.
  - use a counting variable to keep track of how much valid data is in the array.

```java
int[] array = new int[100];
int count = 0;

... System.out.print("Enter a number or -1 to quit: ");
input = keyboard.readLine();
number = Integer.parseInt(input);
while (number != -1 && count <= 99) {
    count +=
    array[count - 1] = number;
}
...```
Array
Returning an Array Reference

- A method can return a reference to an array.
- The return type of the method must be declared as an array of the right type.

```java
public static double[] getArray()
{
    double[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };
    return array;
}

public static void main(String[] args) {
    double[] array = getArray();
}
```
• Arrays are not limited to primitive data.

• An array of String objects can be created:

```java
String[] names = { "Bill", "Susan", "Steven", "Jean" };
```

The *names* variable holds the address to the array. A String array is an array of references to String objects.
Array
String Arrays

- If an initialization list is not provided, the new keyword must be used to create the array:

```java
String[] names = new String[4];
```

The `names` variable holds the address to the array.

```
    Address
        ↓
names[0]     null
names[1]     null
names[2]     null
names[3]     null
```
Array
String Arrays

- When an array is created in this manner, each element of the array must be initialized.

The `names` variable holds the address to the array.

```
names[0] = "Bill";
names[1] = "Susan";
names[2] = "Steven";
names[3] = "Jean";
```
Array

Calling String Methods On Array Elements

- String objects have several methods.
  - toUpperCase,
  - compareTo
  - equals
  - charAt
- Each element of a String array is a String object.
- Methods can be used by using the array name and index as before.

```java
System.out.println(names[0].toUpperCase());
char letter = names[3].charAt(0);
```
Array

The length Field & The length Method

- Arrays have a **final field** named length.
- String objects have a **method** named length.
- To display the length of each string held in a String array:

  ```java
  for (int i = 0; i < names.length; i++)
      System.out.println(names[i].length());
  ```

- An array’s length is a **field**
  - You **do not** write a set of parentheses after its name.
- A String’s length is a **method**
  - You **do** write the parentheses after the name of the String class’s length method.
Array

Arrays of Objects

- Since Strings are objects, we know that arrays can contain objects.

```java
BankAccount[] accounts = new BankAccount[5];
```

The `accounts` variable holds the address of an `BankAccount` array.

The array is an array of references to `BankAccount` objects.
Arrays of Objects

- Each element needs to be initialized.
  
  ```java
  for (int i = 0; i < accounts.length; i++)
      accounts[i] = new BankAccount();
  ```

- Example: [ObjectArray.java](ObjectArray.java)

  The `accounts` variable holds the address of an BankAccount array.
Array

The Sequential Search Algorithm

• A search algorithm is a method of locating a specific item in a larger collection of data.

• The *sequential search algorithm* uses a loop to:
  • sequentially step through an array,
  • compare each element with the search value, and
  • stop when
    • the value is found or
    • the end of the array is encountered.

• Example: [SearchArray.java](#)
Array
Parallel Arrays

- By using the same subscript, you can build relationships between data stored in two or more arrays.

```java
String[] names = new String[5];
String[] addresses = new String[5];
```

- The names array stores the names of five persons.
- The addresses array stores the addresses of the same five persons.
- The data for one person is stored at the same index in each array.
Array
Parallel Arrays

Relationship between names and addresses array elements.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Person #1</td>
<td>Person #2</td>
<td>Person #3</td>
<td>Person #4</td>
<td>Person #5</td>
</tr>
</tbody>
</table>

- Parallel arrays are useful when storing data of unlike types.
Outline

• Array
• Two-Dimensional Arrays
• ArrayList
Two-dimentional Array

- A two-dimensional array is an array of arrays.
- It can be thought of as having rows and columns.

<table>
<thead>
<tr>
<th>row 0</th>
<th>column 0</th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two-dimensional Array

- Declaring a two-dimensional array requires two sets of brackets and two size declarators:
  - The first one is for the number of rows
  - The second one is for the number of columns.

```java
double[][] scores = new double[3][4];
```

- Notice that each size declarator is enclosed in its own set of brackets.
Two-dimentional Array
Accessing Two-Dimensional Array Elements

• An array is accessed by:
  • the reference name
  • An index of row
  • An index of column

• Example:

```java
public static void main(String[] args) {
    int[][] numbers = new int[2][3];

    numbers[1][2] = 5;
}
```
Two-dimentional Array
Accessing Two-Dimensional Array Elements

The `scores` variable holds the address of a 2D array of doubles.

<table>
<thead>
<tr>
<th>Address</th>
<th>column 0</th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
<td>scores[0][0]</td>
<td>scores[0][1]</td>
<td>scores[0][2]</td>
<td>scores[0][3]</td>
</tr>
<tr>
<td>row 1</td>
<td>scores[1][0]</td>
<td>scores[1][1]</td>
<td>scores[1][2]</td>
<td>scores[1][3]</td>
</tr>
</tbody>
</table>
Two-dimensionnal Array
Accessing Two-Dimensional Array Elements

The `scores` variable holds the address of a 2D array of doubles.

Accessing one of the elements in a two-dimensional array requires the use of both subscripts.

```
scores[2][1] = 95;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>column 0</th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>row 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>row 2</td>
<td>0</td>
<td>95</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Two-dimensional Array
Accessing Two-Dimensional Array Elements

- Programs that process two-dimensional arrays can do so with nested loops.
- To fill the scores array:

```java
for (int row = 0; row < 3; row++)
{
    for (int col = 0; col < 4; col++)
    {
        System.out.print("Enter a score: ");
        input = keyboard.readLine();
        scores[row][col] = Double.parseDouble(input);
    }
}
```
To print out the scores array:

```java
for (int row = 0; row < 3; row++)
{
    for (int col = 0; col < 4; col++)
    {
        System.out.println(scores[row][col]);
    }
}
```
Two-dimensional Array

Initializing

- Initializing a two-dimensional array requires enclosing each row’s initialization list in its own set of braces.

  ```java
  int[][] numbers = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
  ```

- Java automatically creates the array and fills its elements with the initialization values.
  
  - row 0  {1, 2, 3}
  - row 1  {4, 5, 6}
  - row 2  {7, 8, 9}

- Declares an array with three rows and three columns.
Two-dimensional Array
Initializing

The `numbers` variable holds the address of a 2D array of int values.

```java
int[][] numbers = {{1, 2, 3},
                  {4, 5, 6},
                  {7, 8, 9}};
```

**produces:**

<table>
<thead>
<tr>
<th></th>
<th>column 0</th>
<th>column 1</th>
<th>column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>row 1</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>row 2</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
Two-dimensional Array
The length Field

- Two-dimensional arrays are arrays of one-dimensional arrays.
- The length field of the array gives the number of rows in the array.
- Each row has a length constant tells how many columns is in that row.
- Each row can have a different number of columns.
Two-dimentional Array

The length Field

• To access the length fields of the array:

```java
int[][] numbers = {{ 1, 2, 3, 4 },
                  { 5, 6, 7 },
                  { 9, 10, 11, 12 } };

for (int row = 0; row < numbers.length; row++)
{
    for (int col = 0; col < numbers[row].length; col++)
        System.out.println(numbers[row][col]);
}
```

• Example: [Lengths.java](Lengths.java)
Two-dimentional Array
Summing The Elements

```java
int[][] numbers = { { 1, 2, 3, 4 },
                    { 5, 6, 7, 8 },
                    { 9, 10, 11, 12 } };

int total;
total = 0;
for (int row = 0; row < numbers.length; row++)
{
    for (int col = 0; col < numbers[row].length; col++)
        total += numbers[row][col];
}

System.out.println("The total is "+ total);
```
Two-dimentional Array

Summing The Rows

```java
int[][] numbers = {
    { 1, 2, 3, 4},
    { 5, 6, 7, 8},
    { 9, 10, 11, 12}};

int total;

for (int row = 0; row < numbers.length; row++)
{
    total = 0;
    for (int col = 0; col < numbers[row].length; col++)
        total += numbers[row][col];
    System.out.println("Total of row "+row+" is "+total);
}
```
Two-dimensional Array

Summing The Columns

```java
int[][] numbers = {{1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12}};

int total;

for (int col = 0; col < numbers[0].length; col++)
{
    total = 0;
    for (int row = 0; row < numbers.length; row++)
        total += numbers[row][col];
    System.out.println("Total of column "+ col + " is " + total);
}
```
Two-dimensional Array
Passing and Returning Array References

- There is no difference between passing a single or two-dimensional array as an argument to a method.
- The method must accept a two-dimensional array as a parameter.
- Example: Pass2DArray.java
Ragged Arrays

• When the rows of a two-dimensional array are of different lengths, the array is known as a ragged array.

• You can create a ragged array by creating a two-dimensional array with a specific number of rows, but no columns.

```java
int [][] ragged = new int [4][];
```

• Then create the individual rows.

```java
ragged[0] = new int [3];
ragged[1] = new int [4];
ragged[2] = new int [5];
ragged[3] = new int [6];
```
More Than Two Dimensions

- Java does not limit the number of dimensions that an array may be.
- More than three dimensions is hard to visualize, but can be useful in some programming problems.
Selection Sort

- In a selection sort:
  - The smallest value in the array is located and moved to element 0.
  - Then the next smallest value is located and moved to element 1.
  - This process continues until all of the elements have been placed in their proper order.
- Example: SelectionSortDemo.java
Binary Search

A binary search:

- requires an array sorted in ascending order.
- starts with the element in the middle of the array.
- If that element is the desired value, the search is over.
- Otherwise, the value in the middle element is either greater or less than the desired value
- If it is greater than the desired value, search in the first half of the array.
- Otherwise, search the last half of the array.
- Repeat as needed while adjusting start and end points of the search.

Example: [BinarySearchDemo.java](BinarySearchDemo.java)
Outline

• Array
• Two-Dimensional Arrays
• ArrayList
ArrayList Class

• Similar to Array, allows object storage
• Unlike Array, an ArrayList object:
  • Automatically expands when a new item is added
  • Automatically shrinks when items are removed
• Requires:
  • `import java.util.ArrayList;`
ArrayList Class
Creating and Using ArrayList

- Create ArrayList object with no-args constructor
  - ArrayList nameList = new ArrayList();
- To populate the ArrayList, use the add() method
  - nameList.add("James");
  - nameList.add("Catherine");
- To get the current size, call the size() method
  - nameList.size(); returns 2
- To access items in an ArrayList, use the get() method
  - nameList.get(1); where 1 is the index of the item
- Example: ArrayListDemo1.java
ArrayList Class
Using an ArrayList

- **ArrayList class toString() method**
  - Returns string representing all items in the ArrayList
  - `System.out.println(nameList);` yields
    - `[ James, Catherine ]`

- **The ArrayList class remove() method**
  - Removes designated item from the ArrayList
  - `nameList.remove(1);` removes second item

- **Example:** [ArrayListDemo3.java](#)
ArrayList Class
Using an ArrayList

- ArrayList class add() method with one argument adds new items to the end of the ArrayList

- To insert items at a location of choice, use the add() method with two arguments
  - `nameList.add(1, "Mary");` inserts the new item at index 1
  - `Yielding [ James, Mary, Catherine ]`

- To replace an existing item, use the set() method
  - `nameList.set(1, "Becky");` replaces "Mary" with "Becky"

- Example: `ArrayListDemo4.java`
ArrayList Class
Using an ArrayList

• Capacity and Capacity Increment
  • Default initial size of an ArrayList is 10 items
  • To designate initial size, use a parameterized constructor
    • ArrayList list = new ArrayList(100);
  • To designate initial size and size of increment:
    • ArrayList list = new ArrayList(100, 50);
ArrayList Class
Casting with ArrayList get() Method

- An ArrayList object is not typed
- To retrieve items from an ArrayList, you must cast the item to the appropriate type
  - `ArrayList nameList = new ArrayList();`
  - `nameList.add("Mary");` inserts an item
  - `String str = (String)nameList.get(0);`
- Try `get()` without the cast to see the effect
- Example: [ArrayListDemo6.java](#)
ArrayList Class
Using ArrayList as a Generic Data Type

- We can create a type-safe ArrayList object by using generics
- For example an ArrayList object for Strings:
  - `ArrayList<String> nameList = new ArrayList<String>();`
- The `get()` no longer requires casts to work.
- Example: `GenericArrayListDemo1.java`
- Example: `GenericArrayListDemo2.java`