

1.00/1.001 - Lecture 8

Arrays and ArrayLists

Arrays-1

- **Arrays are a simple data structure**
- **Arrays store a set of values of the same type**
 - Built-in types (int, double, etc.) or
 - Objects (Students, Dates, etc.)
- **Arrays are part of the Java language**
 - Arrays are objects, not primitives like int or double.
 - They are declared in the same way as other objects
`int[] intArray= new int[20];`
 - The array object has an int data member, length, that gives the number of elements in the array:
`int aSize= intArray.length; // aSize= 20`
- **Each value is accessed through an index**
`intArray[0]= 4; intArray[1]= 77;`

Arrays, p.2

- Array index always starts at 0, not 1
 - An array with N slots has indices 0 through N-1
 - `intArray` has elements `intArray[0]` through `intArray[19]`
- Array lengths cannot be changed once they are declared
- Arrays can be initialized when declared

```
int[] intArray= {5, 77, 4, 9, 28, 0, -9};
// Note that 'new' is implicit (not needed) in this case
```
- Arrays of numerical values are zero when constructed

Copying Arrays

- To copy an array, use `arraycopy()` method of `System` class, as in the following (assuming `intArray` exists)

```
int[] newArray= new int[intArray.length]
// arraycopy(fromArray, fromIndex, toArray, toIndex, count)
System.arraycopy(intArray, 0, newArray, 0, intArray.length);
// Now intArray and newArray have separate copies of data
```

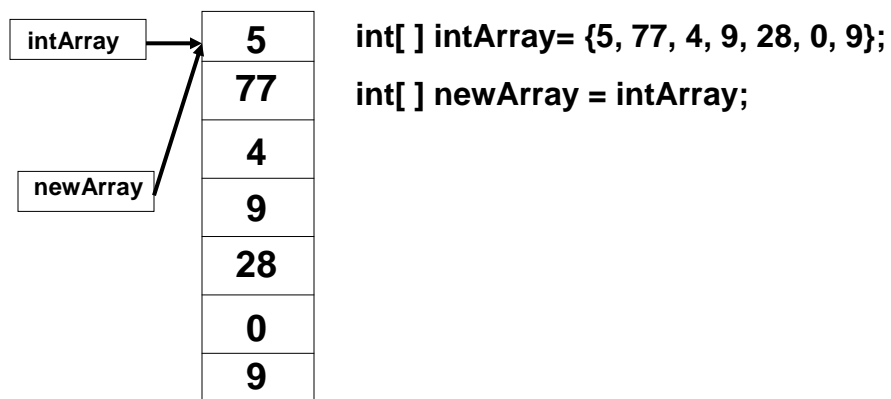
Note: Arrays don't have to be same length as long as segment copied fits into destination array

Copying an array reference

– If we had just defined newArray without copying:

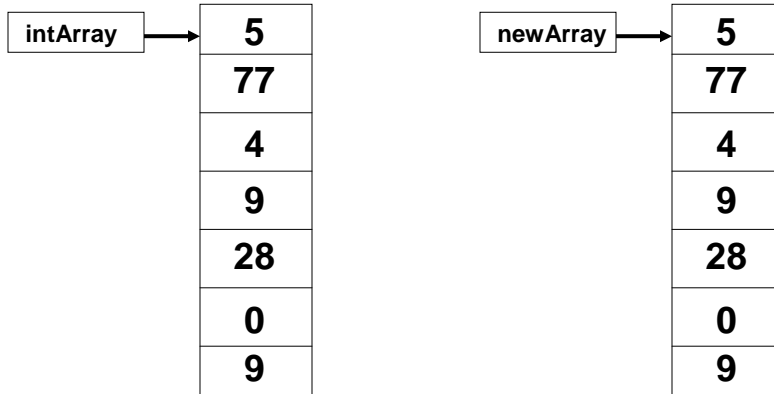
```
int[] newArray= intArray;  
newArray[2]= -44; // This sets  
                // intArray[2]= -44 also  
// intArray and newArray would  
// be two names for the  
// same array. Remember in Java  
// references refer to objects
```

Copying reference to array



Copying entire array

```
int[ ] intArray= {5, 77, 4, 9, 28, 0, 9};  
int[ ] newArray = new int[intArray.length];  
System.arraycopy(intArray,0,newArray,0,intArray.length)
```



Looping Over Array

- Java 1.5 introduced new way of looping over a collection.
- If arrDouble is a reference to array of doubles, the “old” way is:

```
double sum = 0.0;  
for(int i=0; i<arrDouble.length; i++)  
    sum = sum + arrDouble[i];
```

Looping Over Array – Java 1.5

In Java 1.5, we can loop over elements of a collection as follows:

```
double sum = 0.0;
for(double d : arrDouble)
    sum = sum + d;
```

Multidimensional Arrays

– You can create 2, 3, ..n dimensional arrays in Java.

```
int[ ][ ] twoDArray = new int[5][10];
```

A 2D array is actually an array of 1 D arrays, so that

```
twoDArray[3][5]
```

Refers to the sixth element of the fourth array of integers.

Test Your Knowledge

1. Which of the following expressions does not declare and construct an array?

- a. `int[] arr = new int[4];`
- b. `int[] arr;`
`arr = new int [4];`
- c. `int[] arr = {1,2,3,4};`
- d. `int[] arr;`

2. Given this code fragment:

```
int j = ?;  
int[] data = new int[10];  
System.out.println(data[ j ]);
```

Which of the following is a legal value of j?

- a. -1
- b. 0
- c. 1.5
- d. 10

Test Your Knowledge

1. Which of the following expressions does not declare and construct an array?

- a. `int[] arr = new int[4];`
- b. `int[] arr;`
`arr = new int [4];`
- c. `int[] arr = {1,2,3,4};`
- d. `int[] arr;`

2. Given this code fragment:

```
int j = ?;  
int[] data = new int[10];  
System.out.println(data[ j ]);
```

Which of the following is a legal value of j?

- a. -1
- b. 0
- c. 1.5
- d. 10

Test Your Knowledge

3. Given this code fragment:

```
int[] arrayA = new int[4];  
int[] arrayB;  
arrayB = arrayA;  
arrayB[2]=4;  
arrayA[0]=arrayB[2];
```

What are the values of the elements in array A?

- a. unknown
- b. 0,0,0,0
- c. 4,0,4,0
- d. 4,0,0,0

4. How many objects are present after the following code fragment has executed?

```
float[] arrayA = new float[10];  
float[] arrayB;  
arrayB = arrayA;
```

- a. 1
- b. 2
- c. 10
- d. 20

Test Your Knowledge

3. Given this code fragment:

```
int[] arrayA = new int[4];  
int[] arrayB;  
arrayB = arrayA;  
arrayB[2]=4;  
arrayA[0]=arrayB[2];
```

What are the values of the elements in array A?

- a. unknown
- b. 0,0,0,0
- c. 4,0,4,0
- d. 4,0,0,0

4. How many objects are present after the following code fragment has executed?

```
float[] arrayA = new float[10];  
float[] arrayB;  
arrayB = arrayA;
```

- a. 1
- b. 2
- c. 10
- d. 20

Test Your Knowledge

5. For which of these applications an array is NOT suitable?
- Holding the scores on 4 quarters of a Basketball game
 - Holding the name, account balance and account number of an individual
 - Holding temperature readings taken every hour through a day
 - Holding monthly expenses through a year

6. Given the following code fragment:
`int[] data = {1, 3, 5, 7, 11};
for(_____)
System.out.println(data[index]
);`

Fill in the blanks so that the program prints out every element in the array in order

- `int index = 4; index>0; index--`
- `int index=0; index<4; index++`
- `int index=0; index<data.length(); index++`
- `int index=0; index<data.length; index++`

Test Your Knowledge

5. For which of these applications an array is NOT suitable?
- Holding the scores on 4 quarters of a Basketball game
 - Holding the name, account balance and account number of an individual
 - Holding temperature readings taken every hour through a day
 - Holding monthly expenses through a year

6. Given the following code fragment:
`int[] data = {1, 3, 5, 7, 11};
for(_____)
System.out.println(data[index]
);`

Fill in the blanks so that the program prints out every element in the array in order

- `int index = 4; index>0; index--`
- `int index=0; index<4; index++`
- `int index=0; index<data.length(); index++`
- `int index=0; index<data.length; index++`

Test Your Knowledge

7. What is the output of the following program?

```
class Test{
    public static void main ( String[] args ){
        int value = 10;
        int[] arr = {10, 11, 12, 13};
        System.out.println("value before: "+value);
        alterValue( value );
        System.out.println("value after: "+value);
        System.out.println("arr[0]before: "+arr[0]);
        alterArray( arr );
        System.out.println("arr[0] after: "+arr[0]);
    }
    public static void alterValue (int x ){
        x = 0; }
    public static void alterArray (int[]a){
        a[0] = 0; }
}
```

- a. value before:10
value after:0
arr[0] before:10
arr[0] after: 0
- b. value before:10
value after:10
arr[0] before:10
arr[0] after: 10
- c. value before:10
value after:10
arr[0] before:10
arr[0] after: 0
- d. value before:10
value after:0
arr[0] before:10
arr[0] after: 10

Example-Computing an Average

- **Given a list of numerical data items, store them in an array and compute the average value.**
- **We'll use a for loop to iterate over the array entries.**
- **We'll put the computation of the average in a separate method.**

Computing Test Averages

```
public class AverageTest {
    public static void main(String args[ ]) {
        double[ ] testScores = new double[4];
        testScores[0] = 80.0;
        testScores[1] = 50.5;
        testScores[2] = 90.0;
        testScores[3] = 75.0;
        System.out.println("Average is: " + average(testScores)); }

    public static double average(double[ ] aDouble) {
        double aver = 0.0; // initialize value
        for(double d : aDouble)
            aver += d; // accumulate sum of values
        return(aver/aDouble.length); //return average
    }
}
```

Example

- **A. Create a TestArray class to store and manage temperatures for a week**
- **B. Start writing main():**
 - Declare and construct an array of doubles, called `dailyTemp` holding daily temperature data
 - Use an initializer list with braces { }

Mon	Tue	Wed	Thu	Fri	Sat	Sun
70	61	64	71	66	68	62

- Using a for loop, print every element of the `dailyTemp` array in reverse order (starting from Sunday and going backwards to Monday)

```
public class TestArray {
    public static void main(String args[])
    {
        int[] dailyTemp = {70, 61, 64, 71, 66, 68, 62};

        for (int i=dailyTemp.length-1; i>=0; i--)
            System.out.println(dailyTemp[i]);
    }
}
```

ArrayList Class

- The ArrayList class is a fancy version of an array
 - ArrayList can grow automatically as needed
 - Has capacity that is increased when needed
 - Has size() method which returns actual number of elements in the ArrayList
 - ArrayList can hold elements of different types
 - As long as each is an Object (reference),
 - Technically an ArrayList can't hold a basic type (not an int, double, etc.)!
 - BUT, in Java 1.5, conversion of primitive to an object happens automatically. This is called "auto-boxing".
 - Wrapper classes are objects (e.g., Boolean or Double) that hold basic types (e.g. boolean or double)

ArrayLists

- **ArrayList class is not in the core language**
 - They are in package `java.util`, which you must import:
`import java.util.*; // At top of program`
- **ArrayLists are slightly slower than arrays**
 - Matters only in large scale numerical methods
- **ArrayList class has many methods you can use that provide functionality beyond what arrays provide**
- **In Java 1.5 you can declare an ArrayList as containing objects of a particular type. Example:**

```
ArrayList<Point> pList = new ArrayList<Point>( );
```

Some Methods of ArrayList

<code>boolean add (Object o)</code>	Adds object to end, increases size by one
<code>void add(int i, Object o)</code>	Inserts o at index i moving subsequent elements to right
<code>Object get(int i)</code>	Returns object at index i
<code>int indexOf(Object o)</code>	Finds first occurrence of object; uses equals method
<code>void indexOf(Object o)</code>	Searches for first occurrence of o in ArrayList; returns -1 if not found
<code>boolean isEmpty()</code>	Returns true if ArrayList has no objects, false otherwise
<code>void remove (int i)</code>	Deletes obj at index i moving subsequent elements leftward
<code>void set(int i, Object o)</code>	Sets element at index i to be the specified object
<code>int size()</code>	Returns size of ArrayList

Some Methods of ArrayList<OType>

boolean add (OType o)	Adds object to end, increases size by one
void add(int i, OType o)	Inserts o at index i moving subsequent elements to right
Object get(int i)	Returns object at index i
int indexOf(OType o)	Finds first occurrence of object; uses equals method
void indexOf(OType o)	Searches for first occurrence of o in ArrayList; returns -1 if not found
boolean isEmpty()	Returns true if ArrayList has no objects, false otherwise
void remove (int i)	Deletes obj at index i moving subsequent elements leftward
void set(int i, OType o)	Sets element at index i to be the specified object
int size()	Returns size of ArrayList

ArrayList Constructors

- **Three options:**
 - ArrayList()
 - Constructs an empty ArrayList so that its internal data array has size 10 and its standard capacity increment is zero. Capacity will grow as needed, but how much is not specified.
 - ArrayList(int initialCapacity)
 - Constructs an empty ArrayList with the specified initial capacity and with its capacity increment equal to zero. Capacity will double whenever increased.
 - ArrayList(Collection c)
 - We'll talk about this in future lectures

ArrayList Example

- We will show a program that generates a random number of lines with random start and end points.
- We'll use an ArrayList to hold the lines, and use ArrayList's methods to control loop through the contents of the ArrayList

Line class

```
import java.awt.*; // to import the Point class
public class Line {
    int x1,y1; // coordinates of start of line
    int x2,y2; // coordinates of end of line
    // constructor method for Line
    public Line(Point p1, Point p2) {
        x1 = p1.x;
        y1 = p1.y;
        x2 = p2.x;
        y2 = p2.y; }
    public void showLine() // Method to draw line segment
    {
        System.out.println("Coordinates are: (" + x1+", "+ y1+"), (" +
x2+", "+ y2+" )"); }
} // end of class Line
```

ArrayList Example

```
import java.awt.*; // to use Point class
import java.util.*; // to use ArrayList class
public class ArrayListTest {
    static final double MAXCOORD = 10000.0;
    static final int MAXLINES = 30;
    public static void main(String args[]) {
        int numLines = (int) (Math.random()*MAXLINES);
        ArrayList<Line> vec = new ArrayList<Line> ( );
        for (int i=0; i< numLines;i++) {
            Point p1 = new Point( (int) (Math.random( ) * MAXCOORD),
                (int) (Math.random( ) * MAXCOORD));
            Point p2 = new Point( (int) (Math.random( ) * MAXCOORD),
                (int) (Math.random( ) * MAXCOORD));
            Line curLine = new Line (p1, p2);
            vec.add(curLine);
        }
        System.out.println("ArrayList size: "+vec.size());
        for (Line lin : vec)
            lin.showLine();
    }
}
```

Test Your Knowledge

1. Which of the following statements is NOT true about ArrayLists?
 - a. ArrayLists are slightly faster than arrays.
 - b. ArrayLists can store elements of different types.
 - c. ArrayLists can increase in size to store more elements.
 - d. ArrayLists have methods to manage their content.

Test Your Knowledge

1. Which of the following statements is NOT true about ArrayLists?
- a. ArrayLists are slightly faster than arrays.
 - b. ArrayLists can store elements of different types.
 - c. ArrayLists can increase in size to store more elements.
 - d. ArrayLists have methods to manage their content.

Test Your Knowledge

2. Given the following code fragment:

```
ArrayList<String> myArrayList = new  
    ArrayList<String>( );  
myArrayList.add("One");  
myArrayList.add("Two");  
myArrayList.addE("Three");  
myArrayList.add("Four");
```

Which one of the following expressions will modify myArrayList so it looks like:

- One; Two; Four
- a. myArrayList.remove(myArrayList.get(3));
 - b. myArrayList.remove(myArrayList.indexOf("Three"));
 - c. myArrayList.remove(3);
 - d. myArrayList.remove(myArrayList.get(2));

Test Your Knowledge

2. Given the following code fragment:

```
ArrayList<String> myArrayList = new  
    ArrayList<String>( );  
myArrayList.add("One");  
myArrayList.add("Two");  
myArrayList.addE("Three");  
myArrayList.add("Four");
```

Which one of the following expressions will modify myArrayList so it looks like:

One; Two; Four

- a. myArrayList.remove (myArrayList.get(3));
- b. myArrayList.remove (myArrayList.indexOf("Three"));
- c. myArrayList.remove (3);
- d. myArrayList.remove (myArrayList.get(2));

Test Your Knowledge

3. Given the code fragment of question 2 (before the changes in answer to question 2 are applied), which one of the following expressions will modify myArrayList so it looks like:

One; Two; Three; Five

- a. myArrayList[3] = "Five"
- b. myArrayList[4] = "Five"
- c. myArrayList.set (myArrayList.indexOf("Four"), "Five");
- d. myArrayList.set (myArrayList.indexOf("Five"), "Four");

Test Your Knowledge

3. Given the code fragment of question 2 (before the changes in answer to question 2 are applied), which one of the following expressions will modify myArrayList so it looks like:

One; Two; Three; Five

- a. myArrayList[3] = "Five"
- b. myArrayList[4] = "Five"
- c. myArrayList.set (myArrayList.indexOf("Four"), "Five");
- d. myArrayList.set (myArrayList.indexOf("Five"), "Four");

Automatic conversion of primitives to objects

- In Java 1.5, they introduced “boxing” and “unboxing”. When necessary, compiler converts a primitive (e.g int or double) to corresponding object type, (e.g. Integer or Double).

This makes it OK to add primitive types to a collection such as an ArrayList, as in

```
ArrayList<Integer> myALi st = new ArrayList<Integer>( );  
myALi st.add(1); // 1 is an int; it's boxed  
myALi st.add(3); // same as myALi st.add(new Integer(3));  
myALi st.add(7);  
int iValue = myALi st.get(1); // retrieves Integer and “unboxes” to int
```

Test Your Knowledge

Given the following code fragment:

```
ArrayList<Integer> myArrayList = new  
    ArrayList<Integer>( );  
myArrayList.add(1);  
myArrayList.add(3);  
myArrayList.add(7);
```

Which one of the following expressions will modify myArrayList so it looks like:

1 3 5 7

- a. myArrayList.add (5);
- b. myArrayList.add (2, 5);
- c. myArrayList.add (4, 5);
- d. myArrayList.add (3, 5);

Test Your Knowledge

Given the following code fragment:

```
ArrayList<Integer> myArrayList = new  
    ArrayList<Integer>( );  
myArrayList.add(1);  
myArrayList.add(3);  
myArrayList.add(7);
```

Which one of the following expressions will modify myArrayList so it looks like:

1 3 5 7

- a. myArrayList.add (5);
- b. myArrayList.add (2, 5);
- c. myArrayList.add (4, 5);
- d. myArrayList.add (3, 5);