Java

More Details

Array

Arrays

- A group of variables containing values that all have the same type
- Arrays are fixed-length entities
- In Java, arrays are objects, so they are considered reference types
- But the elements of an array can be either primitive types or reference types (including arrays)

Arrays

- We access the element of an array using the following syntax
  - name[index]
  - Here, “index” must be a nonnegative integer
    - “index” can be *int*, *byte*, *short* or *char* but not *long*
- In Java, every array knows its own length.
- The length information is maintained in a public final int member variable called length.

Declaring and Creating Arrays

- int c[ ] = new int [12];
  - Here, ‘c’ is a reference to an integer array
  - ‘c’ is now pointing to an array object holding 12 integers
  - Like other objects arrays are created using “new” and are created in the heap
  - “int c[ ]” represents both the data type and the variable name. Placing number here is a syntax error.
    - int c[12]; // syntax error/compiler error

Declaring and Creating Arrays

- int[ ] c = new int [12];
  - Here, the data type is more evident i.e. “int[ ]”
  - But does the same work as
    - int c[ ] = new int [12];
- Is there any difference between the above two approaches?
- Yes. See the next slide.

Declaring and Creating Arrays

- int c[ ], x;
  - Here, ‘c’ is a reference to an integer array
  - ‘x’ is just a normal integer variable
- int[ ] c, x;
  - Here, ‘c’ is a reference to an integer array (same as above)
  - But, now ‘x’ is also a reference to an integer array

Using an Array Initializer

- We can also use an array initializer to create an array
  - int n[ ] = {10, 20, 30, 40, 50};
  - The length of the above array is 5
  - n[0] is initialized to 10, n[1] is initialized to 20, and so on
  - The compiler automatically performs a “new” operation taking the count information from the list and initializes the elements properly.

Arrays of Primitive Types

- When created by “new”, all the elements are initialized with default values
  - byte, short, char, int, long, float and double are initialized to ***zero***
  - boolean is initialized to false
- This happens for both member arrays and local arrays

Arrays of Reference Types

- String str[ ] = new String[3];
  - Only 3 String references are created
  - Those references are initialized to “null” by default
  - Need to explicitly create and assign actual String objects in the above three positions.
    - str[0] = new String(“Hello”);
    - str[1] = “World”;
    - str[2] = “I” + “ Like” + “ Java”;

Passing Arrays to Methods

- void modifyArray(double d[ ]){…}
  double temperature[ ] = new double[24];
  modifyArray(temperature);
- Changes made to the elements of ‘d’ inside “modifyArray” is visible and reflected in the original “temperature” array.
- But inside “modifyArray” if we create a new array using “new” and assign it to ‘d’ then ‘d’ will point to the newly created array and changing its elements will have no effect on “temperature” (see next slide).

Passing Arrays to Methods

So, we can say that while passing array to methods, changing the elements is visible, but changing the array reference itself is not visible.

```
void modifyArray( double d[ ] )
    {
  d[0] = 1.1; // visible to the
    caller
}


void modifyArray( double d[ ] )
    {
  d = new double [ 10 ];
  d[0] = 1.1; // not visible to
    the caller
}
```

Multidimensional Arrays

- Can be termed as array of arrays.
- int b[ ][ ] = new int[3][4];
  - Length of first dimension = 3
    - b.length equals 3
  - Length of second dimension = 4
    - b[0].length equals 4
- int[ ][ ] b = new int[3][4];
  - Here, the data type is more evident i.e. “int[ ][ ]”

Multidimensional Arrays

- int b[ ][ ] = { { 1, 2, 3 }, { 4, 5, 6 } };
  - b.length equals 2
  - b[0].length and b[1].length equals 3
- All these examples represent rectangular two dimensional arrays where every row has same number of columns.
- Java also supports jagged array where rows can have different number of columns (see next slide).

Multidimensional Arrays

- **Example – 1**

```
int b[ ][ ];
b = new int[ 2 ][ ];
b[ 0 ] = new int[ 2 ];
b[ 1 ] = new int[ 3 ];
b[0][2] = 7; //will throw an exception
```

- **Example – 2**

```
int b[ ][ ] = { { 1, 2 }, { 3, 4, 5 } };
b[0][2] = 8; //will throw an exception
```

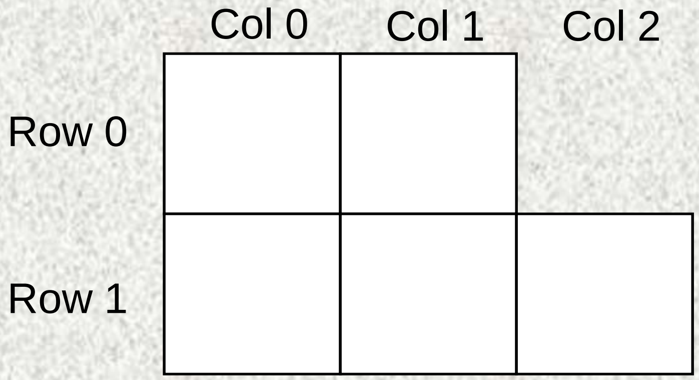
In both cases

b.length equals 2

b[0].length equals 2

b[1].length equals 3

Array ‘b’



b[0][2] does not exist

Command Line Arguments

Using Command-Line Arguments

java MyClass arg1 arg2 … argN

- words after the class name are treated as command-line arguments by “java”
- “java” creates a separate String object containing each command-line argument, places them in a String array and supplies that array to “main”
- That’s why we have to have a String array parameter (String args[ ]) in “main”.
- We do not need a “argc” type parameter (for parameter counting) as we can easily use “args.length” to determine the number of parameters supplied.

Using Command-Line Arguments

```
public class MyClass
{
  public static void main ( String args[ ] )
  {
    System.out.println( args.length );

    for( int i = 0; i < args.length; i++)
    {
      System.out.println( args[i] );
    }
  }
}
```

- **Sample Execution – 1**
  - java MyClass Hello World
  - Output:
    - 2
    - Hello
    - World
- **Sample Execution – 2**
  - java MyClass Hello 2 you
  - Output:
    - 3
    - Hello
    - 2
    - you

Others

Unsigned right shift operator

- The >> operator automatically fills the high-order bit with its previous contents each time a shift occurs.
- This preserves the sign of the value.
- But if you want to shift something that doesn’t represent a numeric value, you may not want the sign extension.
- So Java’s unsigned, shift right operator >>> shifts zeros into the high-order bit.
- *int a= -1; a = a >>> 24;*

```
11111111  11111111   11111111    11111111  [-1]
00000000  00000000   00000000    11111111  [255]
```

For-Each version of the for loop

```
int nums [] = {1,2,3,4,5};
for(int x : nums)
{
    System.out.print(x + " ");
     x=x*10; // no effect on nums
}
System.out.println("");
for(int x : nums)
{
    System.out.println(x + " ");
}
```

Output:

```
1 2 3 4 5
1 2 3 4 5
```

```
int nums[][] =new int[3][5];
for(int i=0;i<3;i++)
     for(int j=0;j<5;j++)
               nums[i][j]=(i+1)*(j+1);

for(int x[]:nums)
{
     for(int y:x)
     {
               System.out.print(y + "  ");
     }
     System.out.println("");
}
```

Nested and Inner Classes

- It is possible to define a class within another classes, such classes are known as nested classes.
- The scope of nested class is bounded by the scope of its enclosing class. That means if class B is defined within class A, then B doesn’t exists without A.
- The nested class has access to the members (including private !) of the class in which it is nested.
- The enclosing class doesn’t have access to the members of the nested class.

Nested and Inner Classes

- Two types of nested classes.
  - Static
  - Non-Static
- A static nested class is one which has the static modifier applied. Because it is static, it must access the members of its enclosing class through an object.
- That is, it cannot refer to members of its enclosing class directly. Because of this restriction, static nested classes are seldom used.

Nested and Inner Classes

- The most important type of nested class is the inner class.
- An inner class is a non-static nested class.
- It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.
- Thus, an inner class is fully within the scope of its enclosing class.

Nested and Inner Classes

```
class Outer
{
    int outer_x = 100;
    void test()
    {       Inner inner = new Inner();
            inner.display();
    }
    // this is an inner class
    class Inner
    {       void display() {
            System.out.println(outer_x);
            }
    }
}
```

```
class InnerClassDemo
{
    public static void main(String args[])
    {
            Outer outer = new Outer();
            outer.test();
    }
}
```

Nested and Inner Classes

```
class Outer
{
    int outer_x = 100;
    void test()
    {       Inner inner = new Inner();
            inner.display();
    }
    // this is an inner class
    class Inner
    {
            int y = 10; // y is local to Inner
            void display() {
            System.out.println(outer_x);
            }
    }
```

```
    void showy()
    {
            System.out.println(y); // error,
            y not known here!
    }
}

class InnerClassDemo
{
    public static void main(String args[])
    {
            Outer outer = new Outer();
            outer.test();
    }
}
```

Nested and Inner Classes

```
class Outer
{    int outer_x = 100;
     void test() {
       for(int i=0; i<5; i++) {
          class Inner
          {
               void display(){
               System.out.println(outer_x);
               }
          }
          Inner inner = new Inner();
          inner.display();
       }
    }
}
```

```
class InnerClassDemo
{
     public static void main(String args[])
     {
               Outer outer = new Outer();
               outer.test();
     }
}
```

Output:

```
100
100
100
100
100
```

Scanner

- It is one of the utility class located in the java.util package.
- Using Scanner class, we can take inputs from the keyboard.
- Provides methods for scanning
  - Int
  - float
  - Double
  - line etc.

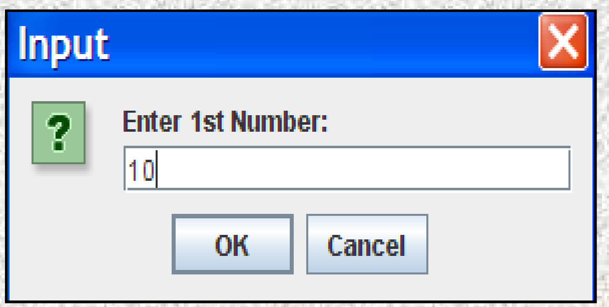
Scanner

```
class ScannerTest
{
  public static void main(String args[])
  {
   Scanner scn=new Scanner(System.in);
   while(scn.hasNextLine())
    {
      System.out.println(scn.nextLine());
   }
  }
}
```

```
class ScannerTest
{
  public static void main(String args[])
  {
   Scanner scn=new Scanner(System.in);
   while(scn.hasNextInt())
    {
      System.out.println(scn.nextInt());
   }
  }
}
```

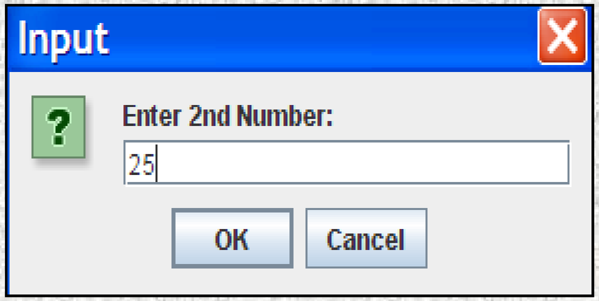
JOptionPane

```
import javax.swing.*;
class TestInput
{
	public static void main(String args[])
	{
String s1=JOptionPane.showInputDialog(null,"Enter
	1st number:");
String s2=JOptionPane.showInputDialog(null,"Enter
	2nd Number:");
int num1=Integer.parseInt(s1);
int num2=Integer.parseInt(s2);
JOptionPane.showMessageDialog(null,"Sum is : " +
	(num1+num2));
	}
}
```



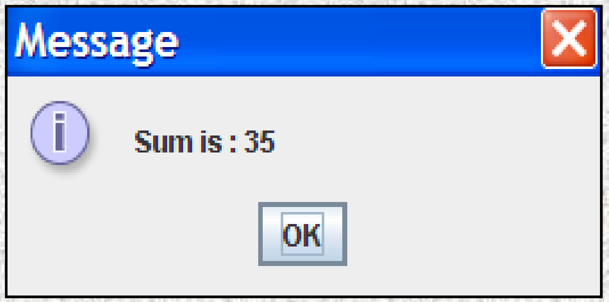
Enter 1st Number:

10



Enter 2nd Number:

25



Sum is : 35