Lecture 03
Java Fundamentals (III)

Dr. Ahmed ElShafee

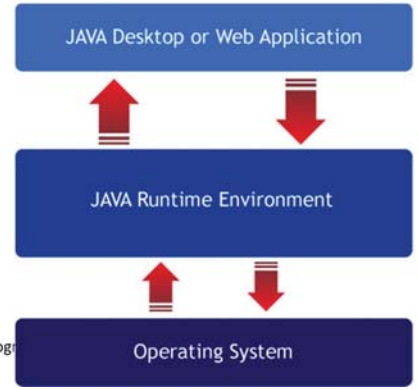
Agenda

1. Another prospective to java programs
2. Main method args
3. Arrays
4. Methods
5. Files

another prospective to java programs

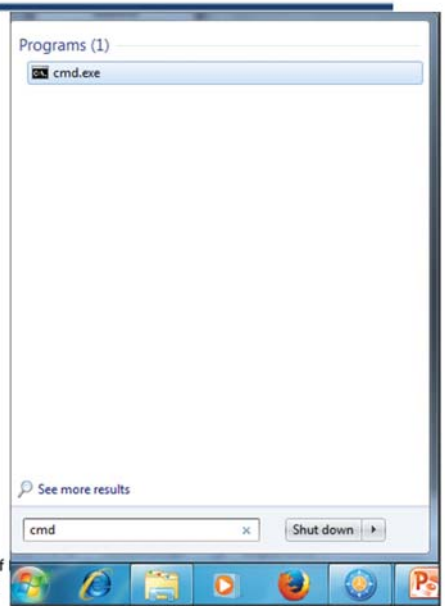
Java components

1. **The Java Virtual Machine or (Java Runtime Environment (JRE))**
2. **The Java Software Development Kit**

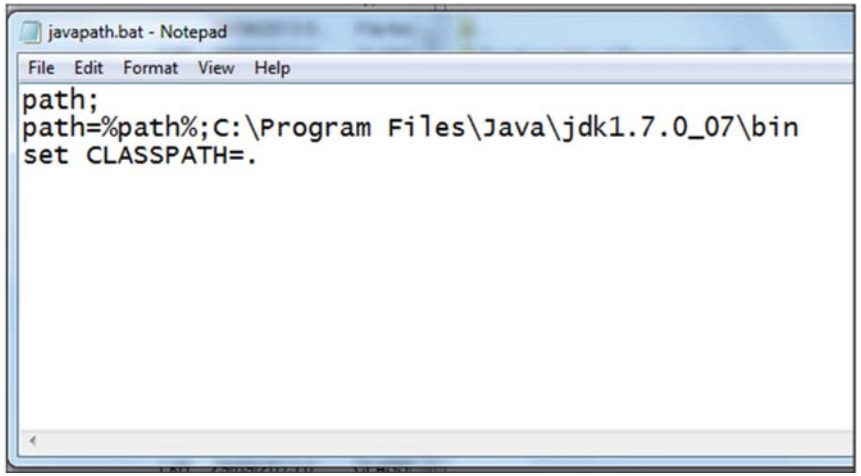


User prospective to a java program (1)

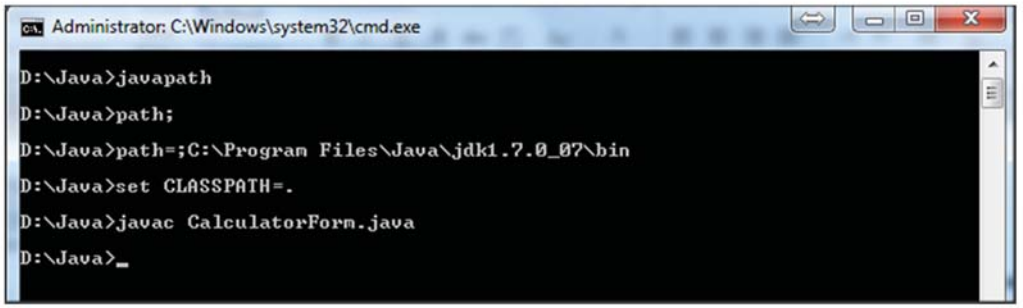
- Create source code with the extension .java (text Editor)
- Open command prompt



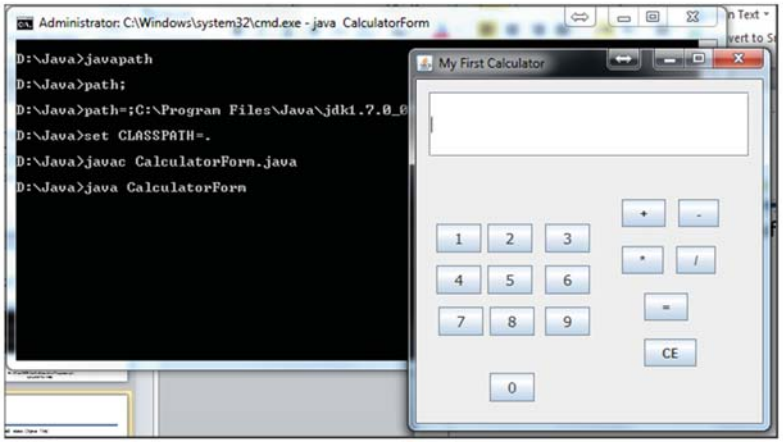
- Make sure that “java/bin” folder is in system path
- Make sure system variable “CLASSPATH=.”



- Use Javac to create (compile) a file ending in .class (>Javac.exe)



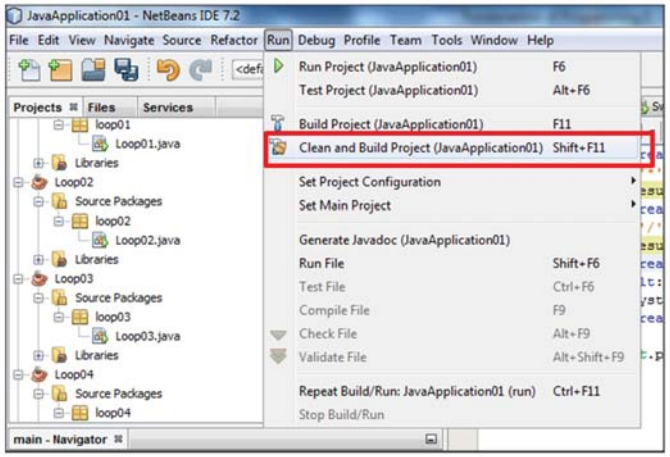
- Run the compiled class (>java file)



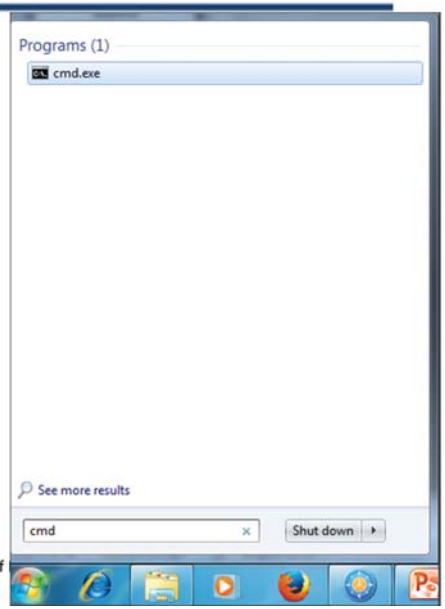
JavaApplication01

- Check lab manual
- Now edit your java file (after removing package statement, use javac to create *.class file, use java.exe to run file under command prompt)

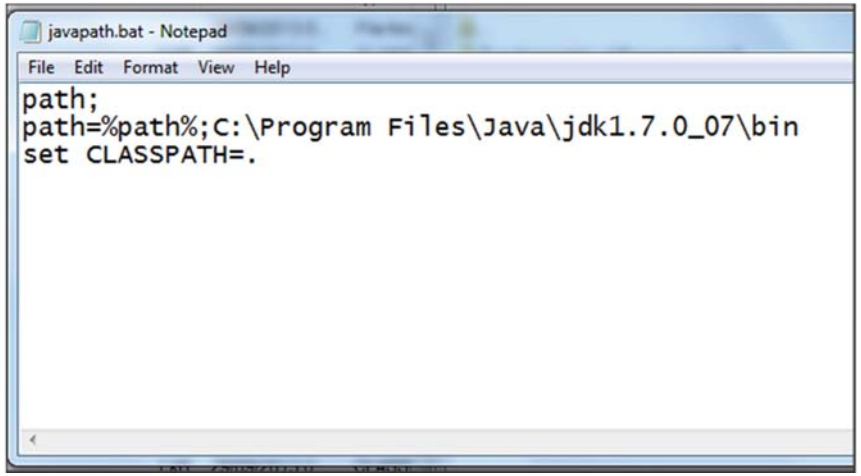
- **User prospective to a java program (2)**
- Create jar archive from your last project



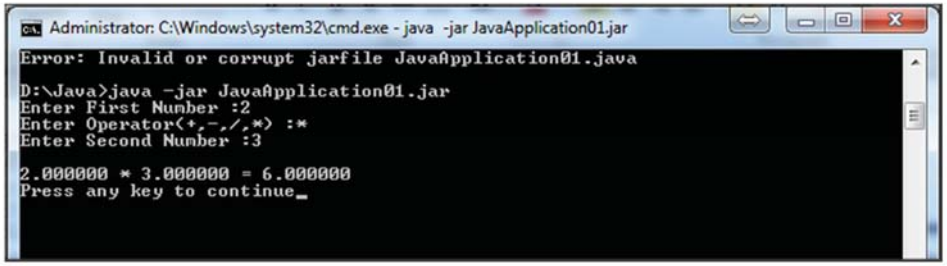
- Open command prompt



- Make sure that "java/bin" folder is in system path
- Make sure system variable "CLASSPATH=."

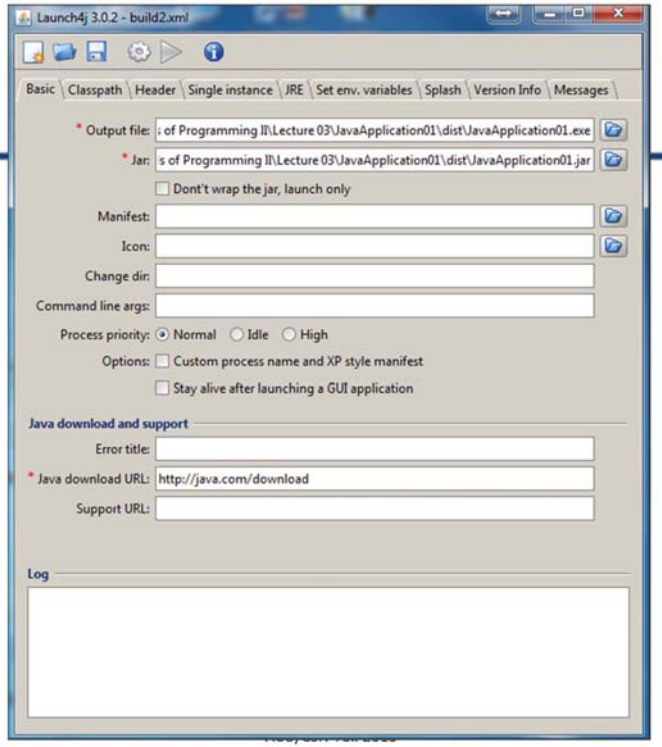


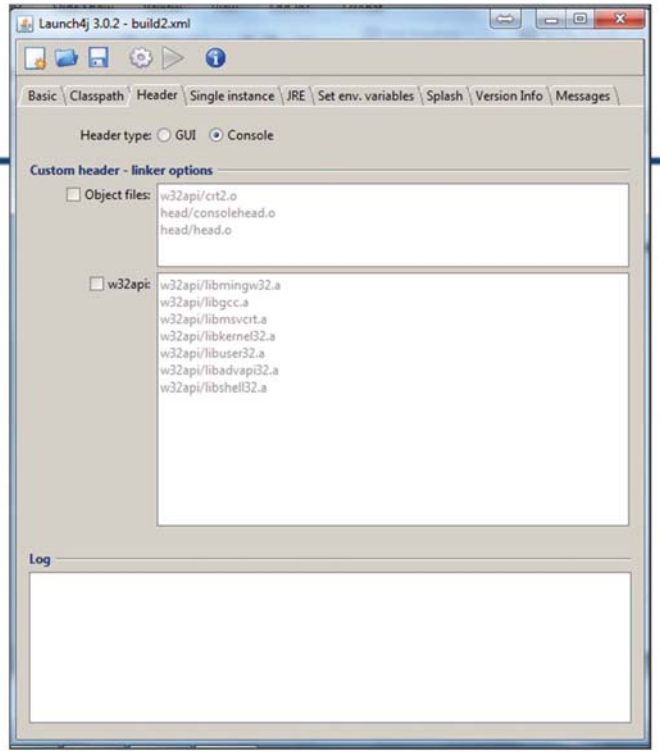
- Run the compiled class (>java -jar file.jar)

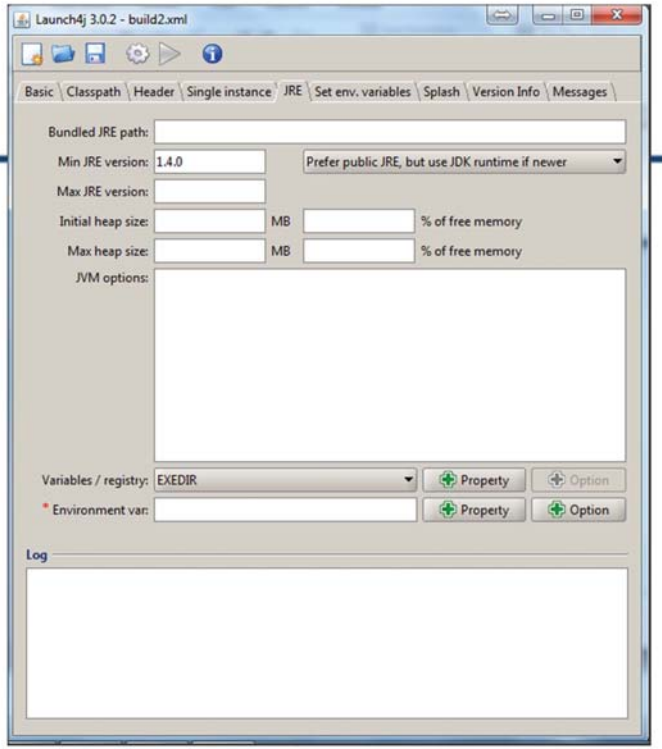


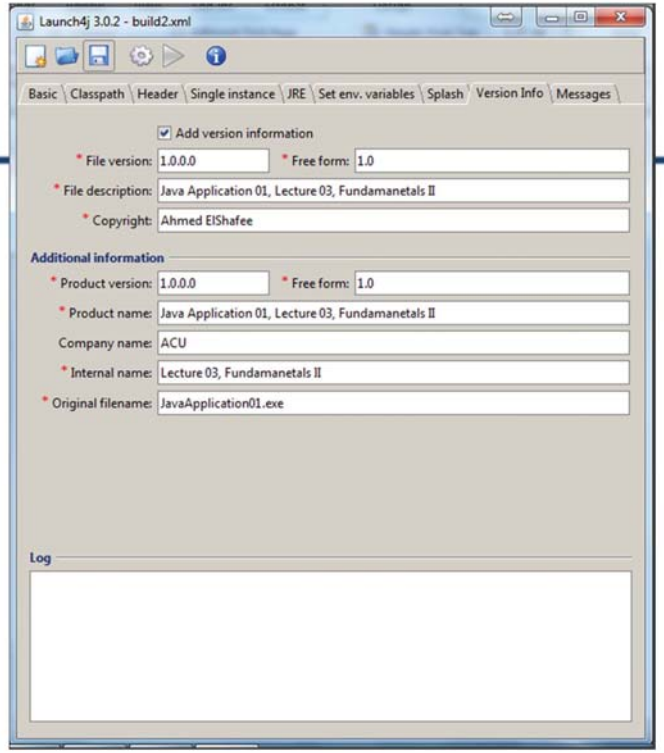
- **User prospective to a java program (3)**
- Use lunch4j to create executable file from jar file

-





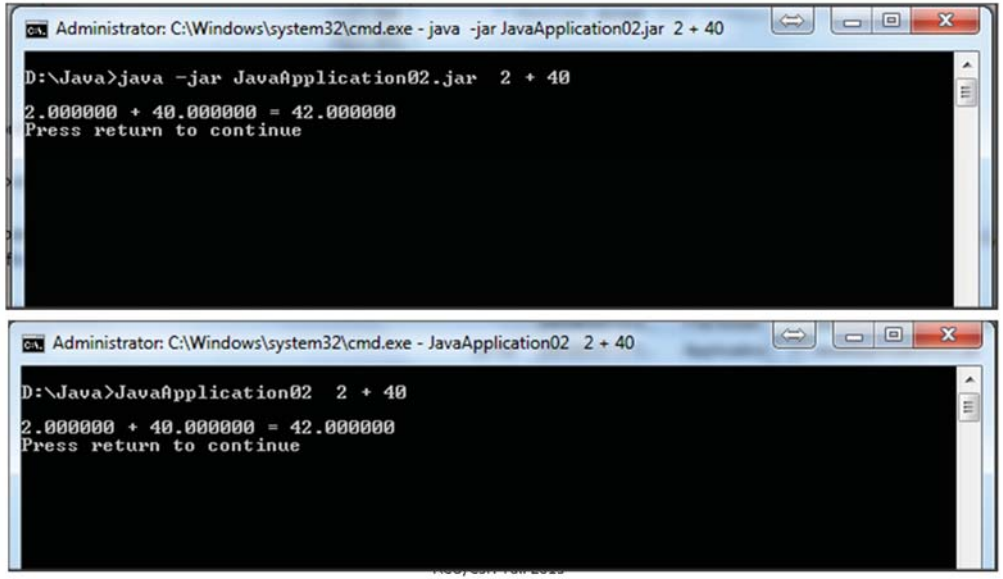




Main method args

JavaApplicatin02

- Check lab manual



Java - Arrays

- Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Declaring Array Variables:

1. Creating array type

```
dataType[] arrayRefVar; // preferred way.
dataType arrayRefVar[]; // works but not preferred way. (C/C++)
```

```
double[] myList; // preferred way.
double myList[]; // works but not preferred way.
```

2. Creating array reference

```
arrayRefVar = new dataType[arraySize];
```

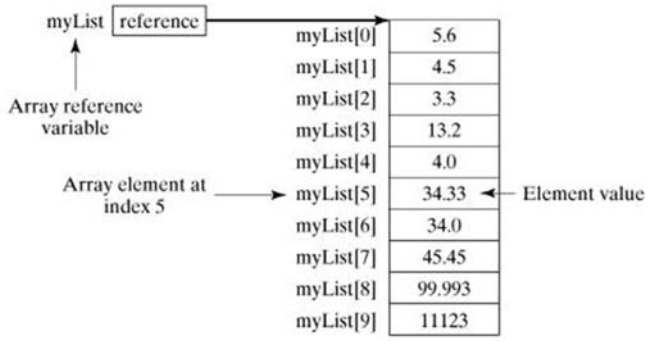
```
myList = new double[10];
```


- Single step

```
dataType[] arrayRefVar = new dataType[arraySize];

dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

```
double[] myList = new double[10];
```



JavaApplication03

Processing Arrays:

- When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

```
public class JavaApplication03 {
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // Print all the array elements
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }
    // Summing all elements
    double total = 0;
    for (int i = 0; i < myList.length; i++) {
      total += myList[i];
```



```
    }
    System.out.println("Total is " + total);
    // Finding the largest element
    double max = myList[0];
    for (int i = 1; i < myList.length; i++) {
      if (myList[i] > max) max = myList[i];
    }
    System.out.println("Max is " + max);
  }
}
```

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

The foreach Loops:

```
public class TestArray {
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // Print all the array elements
    for (double element: myList) {
      System.out.println(element);
    }
  }
}
```

```
1.9
2.9
3.4
3.5
```


Passing Arrays to Methods:

- Just as you can pass primitive type values to methods, you can also pass arrays to methods

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

- You can invoke it by passing an array.

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
Int[] list={1,2,3,4,5};
printArray(list);
```


Returning an Array from a Method:

- A method may also return an array.

```
public static int[] reverse(int[] list)
   {
     int[] result = new int[list.length];
     int j=list.length-1;
     for(int i=0;i<(list.length);i++)
     {
       result[j] = list[i];
       j--;
     }
     return result;
}
```


JavaApplication04

```
public class JavaApplication04 {
public static void main(String[] args) {
      int[] list={3, 1, 5, 6, 4, 2};
      printArray(list);
      int [] revlist=reverse(list);
      System.out.println("");
      printArray(revlist);
      int[] sortedlist=sortascending(list);
      System.out.println("");
      printArray(sortedlist);


  }
```

```
 public static void printArray(int[]
array)
    {
    for (int i = 0; i < array.length; i++)
     {
       System.out.print(array[i] + " ");
     }
    }
    public static int[] reverse(int[] list)
    {
      int[] result = new int[list.length];
      int j=list.length-1;
      for(int i=0;i<(list.length);i++)
      {
        result[j] = list[i];
        j--;
      }
      return result;
}
```

```
public static int[] sortascending (int[] list)
  {
    int[] result=list;
    for(int i=0;i<result.length;i++)
    {
      for(int j=i+1;j<result.length;j++)
      {
        if(result[i]>result[j])
        {
          int temp = result[i];
          result[i] = result[j];
          result[j] = temp;
        }       }     }
    return result;
  }
```

```
3 1 5 6 4 2
2 4 6 5 1 3
1 2 3 4 5 6
```


1	**public static int binarySearch(Object[] a, Object key)** Searches the specified array of Object (Byte, Int , double etc) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, (-(insertion point + 1).
2	**public static boolean equals(long[] a, long[] a2)** Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. This returns true if the two arrays are equal. Same method could be used by all other premitive data types (Byte, short, Int etc.)
3	**public static void fill(int[] a, int val)** Assigns the specified int value to each element of the specified array of ints. Same method could be used by all other premitive data types (Byte, short, Int etc.)
4	**public static void sort(Object[] a)** Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. Same method could be used by all other premitive data types (Byte, short, Int etc.)

Java - Methods

- A Java method is a collection of statements that are grouped together to perform an operation.
- When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console.

Creating a Method:

```
modifier returnValueType methodName(list of parameters) {
  // Method body;
}
```


- A method definition consists of a method header and a method body. Here are all the parts of a method:
  - **Modifiers:** The modifier, which is optional, tells the compiler how to call the method. This defines the access type of the method.
  - **Return Type:** A method may return a value. The returnValueType is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the returnValueType is the keyword **void**.

- **Method Name:** This is the actual name of the method. The method name and the parameter list together constitute the method signature.
- **Parameters:** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- **Method Body:** The method body contains a collection of statements that define what the method does.


```
/** Return the max between two numbers */
public static int max(int num1, int num2) {
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```


Calling method

- If the method returns a value, a call to the method is usually treated as a value. For example:

```
int larger = max(30, 40);
```

- If the method returns void, a call to the method must be a statement.

```
System.out.println("Welcome to Java!");
```



```
public class TestMax {
  /** Main method */
  public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum
between " + i + " and " + j + " is " + k);
}
```

```
/** Return the max between two
numbers */
public static int max(int num1, int num2)
{
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

```
The maximum between 5 and 2 is 5
```

javaapplication05

```
public class javaapplication05 {
  public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
     float degree=1;
     while(degree>0)
     {
     System.out.print("Enter degree to
calculate grade; 0 to exit: ");
     degree=sc.nextFloat();
     printGrade(degree);
     } }
  public static void printGrade(double score) {
    if (score >= 90.0) {
      System.out.println('A');
    }
    else if (score >= 80.0) {
      System.out.println('B');
    }
    else if (score >= 70.0) {
      System.out.println('C');
    }

    else if (score >= 60.0) {
      System.out.println('D');
    }
    else {
      System.out.println('F');
    }
  }
}
```


JavaApplication06

Passing Parameters by Values:

- parameter order association

```
public class JavaApplication06{
  public static void main(String[] args) {
    int num1 = 1;
    int num2 = 2;
    System.out.println("Before swap method, num1 is " + num1 + " and num2 is " +
num2);
    // Invoke the swap method
    swap(num1, num2);
    System.out.println("After swap method, num1 is " + num1 + " and num2 is " +
num2);
  }
```



```
/** Method to swap two variables */
  public static void swap(int n1, int n2) {
    System.out.println("\tInside the swap method");
    System.out.println("\t\tBefore swapping n1 is " + n1 + " n2 is " + n2);
    // Swap n1 with n2
    int temp = n1;
    n1 = n2;
    n2 = temp;
    System.out.println("\t\tAfter swapping n1 is " + n1 + " n2 is " + n2);
  }
}
```

```
Before swap method, num1 is 1 and num2 is 2
      Inside the swap method
            Before swapping n1 is 1 n2 is 2
            After swapping n1 is 2 n2 is 1
After swap method, num1 is 1 and num2 is 2
```


Overloading Methods:

- The max method that was used earlier works only with the int data type.
- But what if you need to find which of two floating-point numbers has the maximum value?
- The solution is to create another method with the same name but different parameters,

```
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```

JavaApplication07

```
public class JavaApplication07 {
   /**
    * @param args the command line
arguments
    */
   public static void main(String[] args) {
      int int1,int2;
      float float1,float2;
      Scanner sc=new Scanner(System.in);
      System.out.print("Enter first integer
:");
      int1=sc.nextInt();
      System.out.print("Enter second
integer :");
      int2=sc.nextInt();
```

```
System.out.printf("Max of %d , and %d =
%d\n",int1,int1,max(int1,int2));
      System.out.print("Enter first float :");
      float1=sc.nextFloat();
      System.out.print("Enter second Float
:");
      float2=sc.nextFloat();
      System.out.printf("Max of %f , and
%f=
%f\n",float1,float2,max(float1,float2));
   }
```



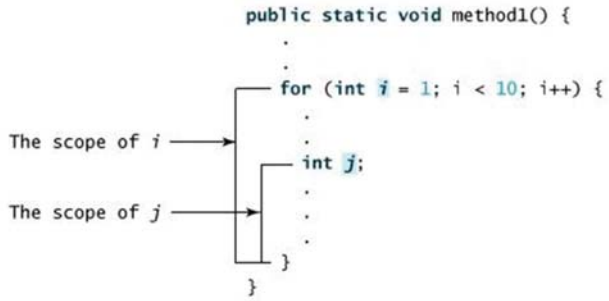
```
public static int max(int num1, int num2) {
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;
  return result;
}
 public static float max(float num1, float
num2) {
  float result;
  if (num1 > num2)
    result = num1;
```

```
else
    result = num2;
  return result;
}
```

```
Enter first integer :5
Enter second integer :10
Max of 5 , and 5 = 10
Enter first float :2.3
Enter second Float :6.2
Max of 2.300000 , and 6.200000=
6.200000
```


The Scope of Variables

- A variable defined inside a method is referred to as a local variable.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.



- Passing array to method
- Method declaration

```
public static void printMax( double...numbers) {
}
```

- Method call

```
printMax(new double[]{1, 2, 3});
```

javaapplication09

- Check lab manual

Files

Reading text files

- Firstly you need to import the following

```
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;
```

- Create object from class called FileReader

```
 FileReader fr=new FileReader(path);
File f = new File(path);
FileReader ft = new FileReader(f);
```

- Read dara

```
ch=fr.read();
char[] c = new char[(int) f.length()];
fr.read(c);
fr.read(c1,0,c1.length);
 BufferedReader br=new BufferedReader(fr);
aLine = br.readLine()
```


textFiles01

- Check lab manual

textFiles02

- Check lab manual

textFiles03

- Check lab manual

textFiles04

- Check lab manual

Writing text files

- Import library

```
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
```

- Create object from WriteFile class (append: true, new: false)

```
FileWriterwf = new FileWriter( fileName ,true);
File f = new File(filename);
FileWriter fw = new FileWriter( f ,true);
```

- Write data

```
fw.write(st.charAt(i));
PrintWriter pw = new PrintWriter(fw);
pw.printf("%s\r\n", st);
```


textFiles05

check lab manual

textFiles06

- Check lab manual

- **Read & write data to text files**

Java IO: Data

The DataInputStream class enables you to read Java primitives from InputStream's instead of only bytes.

You wrap an InputStream in a DataInputStream and then you can read primitives from it.

```
DataInputStream input = new DataInputStream(
                            new FileInputStream("binary.data"));

int    aByte   = input.read();
int    anInt   = input.readInt();
float  aFloat  = input.readFloat();
double aDouble = input.readDouble();
//etc.

input.close();
```

- The DataOutputStream class enables you to write Java primitives to OutputStream's instead of only bytes.
- You wrap an OutputStream in a DataOutputStream and then you can write primitives

```
DataOutputStream output = new DataOutputStream(
                           new FileOutputStream("binary.data"));

output.write(45);            //byte data
output.writeInt(4545);       //int data
output.writeDouble(109.123); //double data

output.close();
```


textFiles07

- Check lab manual

textFiles08

- Check lab manual