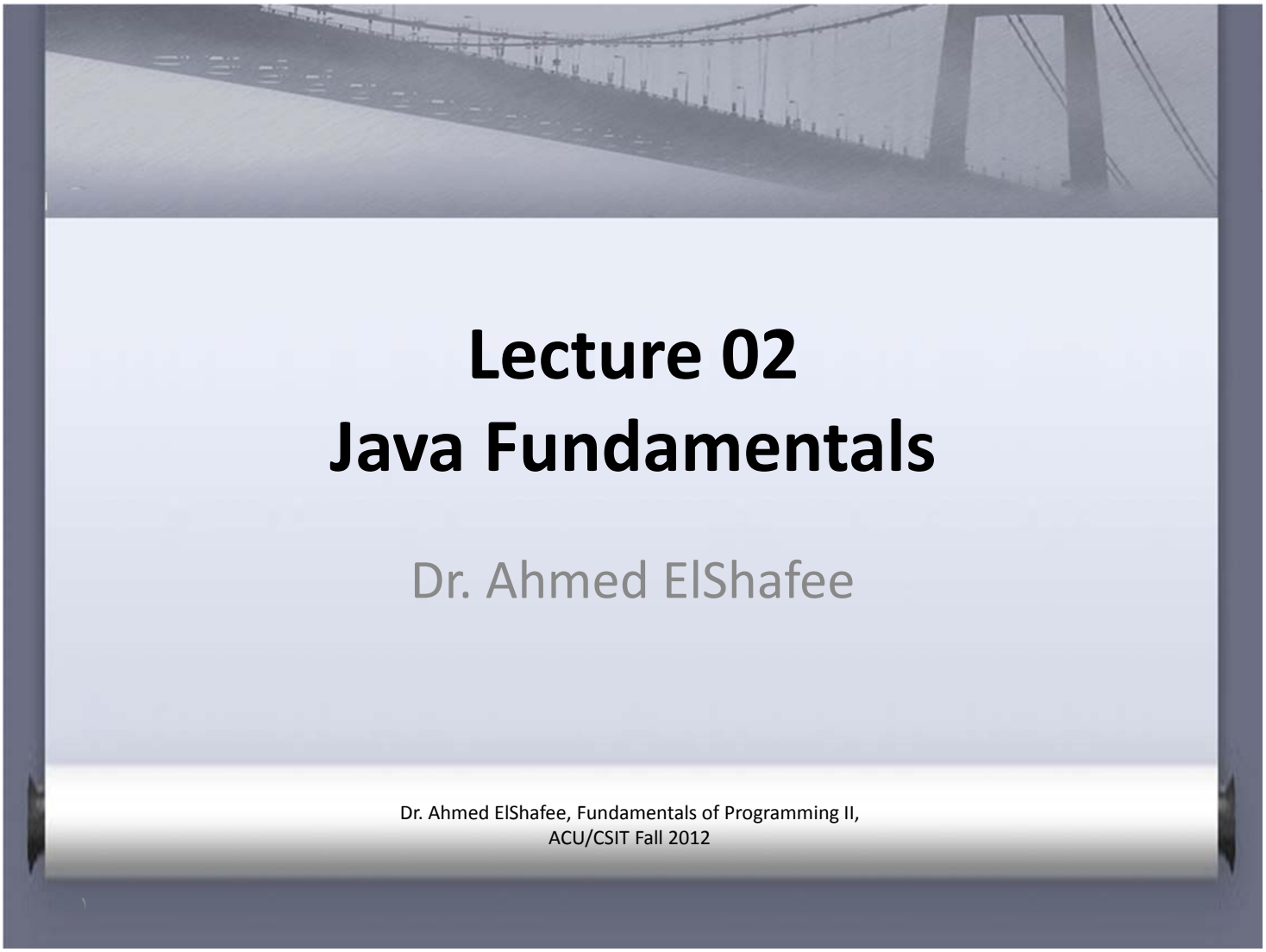


# Lecture 02
# Java Fundamentals

Dr. Ahmed ElShafee



## Agenda

1. **Java - Number Classes**
2. Java – String classes
3. Arrays
4. Methods
5. Classes and Objects

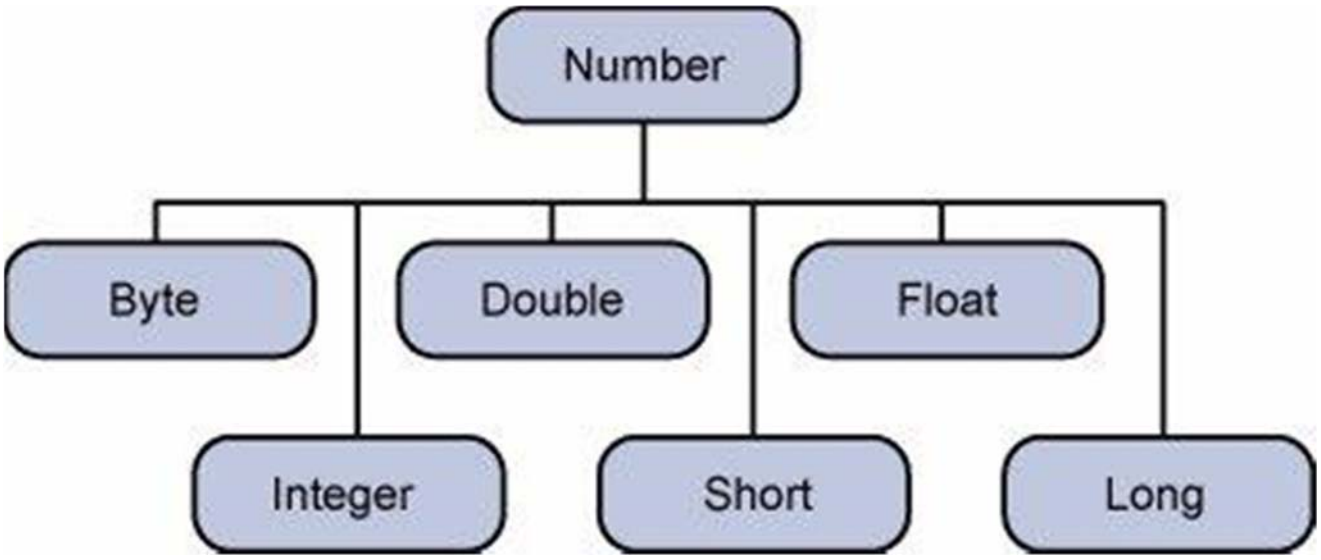
# Java - Number Class

- Normally, when we work with Numbers, we use primitive data types such as byte, int, long, double etc.

```
int i = 5000;
float gpa = 13.65;
byte mask = 0xaf;
```

- However in development we come across situations were we need to use objects instead of primitive data types. In-order to achieve this Java provides wrapper classes for each primitive data type.
- All the wrapper classes ( Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number.





- This wrapping is taken care of by the compiler The process is called boxing.
- So when a primitive is used when an object is required the compiler boxes the primitive type in its wrapper class.
- Similarly the compiler unboxes the object to a primitive as well.
- The **Number** is part of the java.lang package.



```
public class JavaApplication01 {
  public static void main(String args[]){
    Integer x = 5; // boxes int to an Integer object
    x =  x + 10;   // unboxes the Integer to a int
    System.out.println(x);
  }
}
```

```
15
```

- Here is the list of the instance methods that all the subclasses of the Number class implement:

| SN | Methods with Description |
|---|---|
| 1 | xxxValue(Integer)<br>Converts the value of *this* Number object to the xxx data type and returned it. (byteValue(), doubleValue(), shortValue(), floatValue(), intValue(),…) |
| 2 | compareTo(Integer)<br>Compares *this* Number object to the argument. |
| 3 | equals()<br>Determines whether *this* number object is equal to the argument. |
| 4 | valueOf()<br>Returns an Integer object holding the value of the specified primitive. |



| SN | Methods with Description |
|---|---|
| 5 | toString()<br>Returns a String object representing the value of specified int or Integer. |
| 6 | parseInt()<br>This method is used to get the primitive data type of a certain String. |
| 7 | toBinaryString(int i)<br>Returns a string representation of the integer argument as an unsigned integer in base 2. |
| 8 | toHexString(int i)<br>Returns a string representation of the integer argument as an unsigned integer in base 16. |

```
public class JavaApplication01 {
public static void main(String[] args) {
    Integer x; // boxes int to an Integer object
    int y,z;
    Scanner sc=new Scanner(System.in);
    System.out.printf("\nEnter X = ");
    x=sc.nextInt();
    System.out.printf("\nEnter Y = ");
    y=sc.nextInt();
   System.out.printf("\n%d << 1 = %d",x,Integer.rotateLeft(x, 1));
   System.out.printf("\n%d << 1 = %d",y,Integer.rotateLeft(y, 1));
   System.out.printf("\n%d >> 1 = %d",x,Integer.rotateRight(x, 1));
   System.out.printf("\n%d >> 1 = %d",y,Integer.rotateRight(y, 1));
```



```
y=Integer.reverse(x);
    System.out.printf("\n%d' = %d",x,y);
    z=Integer.reverse(y);
    System.out.printf("\n%d' = %d",y,z);
   }
}
```

```
Enter X = 4
Enter Y = 5
4 << 1 = 8
5 << 1 = 10
4 >> 1 = 2
5 >> 1 = -2147483646
4' = 536870912
536870912' = 4
```

# Java - String Class

- Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.
- The Java platform provides the String class to create and manipulate strings.

```
String greeting = "Hello world!";
```

- Whenever it encounters a string literal in your code, the compiler creates a String object with its valuein this case, "Hello world!'.



- The String class has eleven constructors that allow you to provide the initial value of the string using different sources, such as an array of characters:

```
public class StringDemo{
   public static void main(String args[]){
      char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};
      String helloString = new String(helloArray);
      System.out.println( helloString );
   }
}
```

```
hello
```

- **Note:** The String class is immutable, so that once it is created a String object cannot be changed. If there is a necessity to make alot of modifications to Strings of characters then you should use String Buffer & String Builder Classes.

**String Length:**

```
public class StringDemo{
   public static void main(String args[]){
      String palindrome = "Dot saw I was Tod";
      int len = palindrome.length();
      System.out.println( "String Length is : " + len );
   }
}
```

```
String Length is : 17
```



**Concatenating Strings:**

```
string1.concat(string2);
```

This returns a new string that is string1 with string2 added to it at the end. You can also use the concat() method with string literals, as in:

```
"My name is ".concat("Zara");
```

Strings are more commonly concatenated with the + operator, as in:

```
"Hello," + " world" + "!"
```

```
"Hello, world!"
```

```
public class StringDemo{
   public static void main(String args[]){
      String string1 = "saw I was ";
      System.out.println("Dot " + string1 + "Tod");
   }
}
```

```
Dot saw I was Tod
```

**Creating Format Strings:**

You have printf() and format() methods to print output with formatted numbers.

The String class has an equivalent class method, format(), that returns a String object rather than a PrintStream object.



- Using String's static format() method allows you to create a formatted string that you can reuse, as opposed to a one-time print statement.
- For example, instead of:

```
public class StringDemo{
   public static void main(String args[]){
      String string1 = "saw I was ";
      System.out.println("Dot " + string1 + "Tod");
   }
}
```

- You can use

```
String fs;
fs = String.format("The value of the float variable is " +
                   "%f, while the value of the integer " +
                   "variable is %d, and the string " +
                   "is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```



- **String Methods:**

| SN | Methods with Description |
|---|---|
| | char charAt(int index)<br>Returns the character at the specified index. |
| | int compareTo(Object o)<br>Compares this String to another Object. |
| | int compareTo(String anotherString)<br>Compares two strings lexicographically. |
| | int compareToIgnoreCase(String str)<br>Compares two strings lexicographically, ignoring case differences. |
| | String concat(String str)<br>Concatenates the specified string to the end of this string. |
| | boolean contentEquals(StringBuffer sb)<br>Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer. |

| SN | Methods with Description |
|---|---|
| | static String copyValueOf(char[] data)<br>Returns a String that represents the character sequence in the array specified. |
| | static String copyValueOf(char[] data, int offset, int count)<br>Returns a String that represents the character sequence in the array specified. |
| | boolean endsWith(String suffix)<br>Tests if this string ends with the specified suffix. |
| | boolean equals(Object anObject)<br>Compares this string to the specified object. |
| | boolean equalsIgnoreCase(String anotherString)<br>Compares this String to another String, ignoring case considerations. |
| | byte getBytes()<br>Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array. |

| SN | Methods with Description |
|---|---|
| | byte[] getBytes(String charsetName<br>Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array. |
| | void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)<br>Copies characters from this string into the destination character array. |
| | int indexOf(int ch)<br>Returns the index within this string of the first occurrence of the specified character. |
| | int indexOf(int ch, int fromIndex)<br>Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index. |
| | int indexOf(String str)<br>Returns the index within this string of the first occurrence of the specified substring. |

| SN | Methods with Description |
|---|---|
| | int indexOf(String str, int fromIndex)<br>Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. |
| | String intern()<br>Returns a canonical representation for the string object. |
| | int lastIndexOf(int ch)<br>Returns the index within this string of the last occurrence of the specified character. |
| | int lastIndexOf(int ch, int fromIndex)<br>Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index. |
| | int lastIndexOf(String str)<br>Returns the index within this string of the rightmost occurrence of the specified substring. |
| | int lastIndexOf(String str, int fromIndex)<br>Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index. |

| SN | Methods with Description |
|---|---|
| | int length()<br>Returns the length of this string. |
| | boolean regionMatches(int toffset, String other, int ooffset, int len)<br>Tests if two string regions are equal. |
| | String replace(char oldChar, char newChar)<br>Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar. |

| SN | Methods with Description |
|---|---|
| | int length()<br>Returns the length of this string. |
| | boolean regionMatches(int toffset, String other, int ooffset, int len)<br>Tests if two string regions are equal. |
| | String replace(char oldChar, char newChar)<br>Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar. |
| | String replaceAll(String regex, String replacement<br>Replaces each substring of this string that matches the given regular expression with the given replacement. |



| SN | Methods with Description |
|---|---|
| | String substring(int beginIndex)<br>Returns a new string that is a substring of this string. |
| | String substring(int beginIndex, int endIndex)<br>Returns a new string that is a substring of this string. |
| | char[] toCharArray()<br>Converts this string to a new character array. |
| | String toLowerCase()<br>Converts all of the characters in this String to lower case using the rules of the default locale. |
| | String toLowerCase(Locale locale)<br>Converts all of the characters in this String to lower case using the rules of the given Locale. |
| | String toString()<br>This object (which is already a string!) is itself returned. |

| SN | Methods with Description |
|---|---|
| | String toUpperCase()<br>Converts all of the characters in this String to upper case using the rules of the default locale. |
| | String toUpperCase(Locale locale)<br>Converts all of the characters in this String to upper case using the rules of the given Locale. |
| | String trim()<br>Returns a copy of the string, with leading and trailing whitespace omitted. |
| | static String valueOf(primitive data type x)<br>Returns the string representation of the passed data type argument. |



```
public class JavaApplication02 {
public static void main(String[] args) {
    char[] helloArray = { 'h', 'e', 'l', 'l', 'o'};
    String str = new String(helloArray);
    System.out.println( str );
    str=str+' '+"world.";
    System.out.println( str );
    str=String.format("%s \n hello again", str);
    System.out.println( str );
    System.out.printf( "length of \"%s\"=%d\n",str,str.length());
    str=str.concat(".");
    System.out.printf( "length of \"%s\"=%d\n",str,str.length());
  }
}
```

```
hello
hello world.
hello world.
 hello again
length of "hello
world.
 hello again"=26
length of "hello
world.
 hello again."=27
```

# Java - String Buffer & String Builder Classes

- The **StringBuffer** and **StringBuilder** classes are used when there is a necessity to make alot of modifications to Strings of characters.
- Unlike Strings objects of type StringBuffer and Stringbuilder can be modified over and over again with out leaving behind alot of new unused objects.
- The StringBuilder class was introduced as of Java 5 and the main difference between the StringBuffer and StringBuilder is that StringBuilders methods are not thread safe(not Synchronised).
- It is recommended to use **StringBuilder** whenever possible because it is faster than StringBuffer. However if thread safety is necessary the best option is StringBuffer objects.



```
public class Test{
   public static void main(String args[]){
      StringBuffer sBuffer = new StringBuffer(" test");
      sBuffer.append(" String Buffer");
      System.ou.println(sBuffer);
   }
}
```

```
test String Buffer
```

## StringBuffer Methods:

| SN | Methods with Description |
|---|---|
| 1 | public StringBuffer append(String s)<br>Updates the value of the object that invoked the method. The method takes boolean, char, int, long, Strings etc. |
| 2 | public StringBuffer reverse()<br>The method reverses the value of the StringBuffer object that invoked the method. |
| 3 | public delete(int start, int end)<br>Deletes the string starting from start index until end index. |
| 4 | public insert(int offset, int i)<br>This method inserts an string s at the position mentioned by offset. |
| 5 | replace(int start, int end, String str)<br>This method replaces the characters in a substring of this StringBuffer with characters in the specified String. |



Here is the list of other methods (Except set methods ) which are very similar to String class:

| | |
|---|---|
| | **char charAt(int index)**<br>**The specified character of the sequence currently represented by the string buffer, as indicated by the index argument, is returned.** |
| | **void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)**<br>Characters are copied from this string buffer into the destination character array dst. |
| | **int indexOf(String str)**<br>Returns the index within this string of the first occurrence of the specified substring. |

| SN | Methods with Description |
|---|---|
| 5 | **int indexOf(String str)**<br>Returns the index within this string of the first occurrence of the specified substring. |
| 6 | **int indexOf(String str, int fromIndex)**<br>Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. |
| 7 | **int lastIndexOf(String str)**<br>Returns the index within this string of the rightmost occurrence of the specified substring. |
| 8 | **int lastIndexOf(String str, int fromIndex)**<br>Returns the index within this string of the last occurrence of the specified substring. |
| 9 | **int length()**<br>Returns the length (character count) of this string buffer. |
| 10 | **void setCharAt(int index, char ch)**<br>The character at the specified index of this string buffer is set to ch. |

| SN | Methods with Description |
|---|---|
| 11 | **void setLength(int newLength)**<br>Sets the length of this String buffer. |
| 13 | **String substring(int start)**<br>Returns a new String that contains a subsequence of characters currently contained in this StringBuffer.The substring begins at the specified index and extends to the end of the StringBuffer. |
| 14 | **String substring(int start, int end)**<br>Returns a new String that contains a subsequence of characters currently contained in this StringBuffer. |
| 15 | **String toString()**<br>Converts to a string representing the data in this string buffer. |

# 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 refrence

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

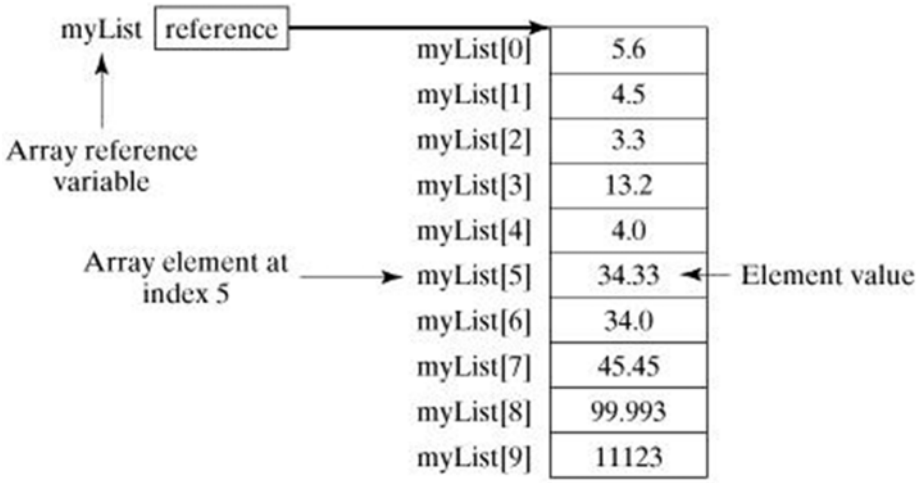
```
myList = new double[10];
```

- Single step

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

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

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






## 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=new int[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];

  for (int i = 0; i = result.length - 1;
                  i <  list.length; i++, j--) {
    result[j] = list[i];
  }
  result result;
}
```

```
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)**<br>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)**<br>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)**<br>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)**<br>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
```



```
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');
    }
  }
}
```

## 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;
}
```



```
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;
}
```

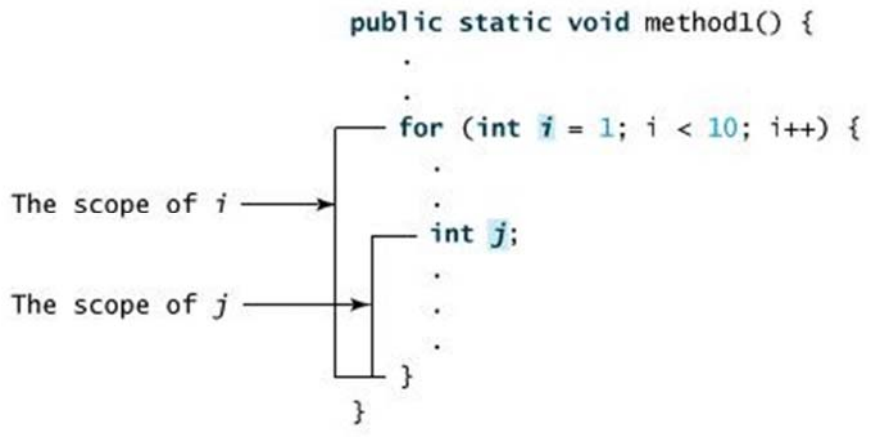
```
Enter first integer :5
Enter second integer :10
Max of 5 , and 5 = 10
Enter first float :2.3
Enter second Float :6.2
Max of 2.300000 , and 6.200000=
6.200000
```





## 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.



## Using Command-Line Arguments:

```
class JavaApplication08 {
   public static void main(String args[]){
      for(int i=0; i<args.length; i++){
         System.out.println("args[" + i + "]: " +
                                        args[i]);
      }
   }
}
```

```
java CommandLine this is a command line 200 -100
```

```
args[0]: this
args[1]: is
args[2]: a
args[3]: command
args[4]: line
args[5]: 200
args[6]: -100
```



## Variable Arguments(var-args):

- JDK 1.5 enables you to pass a variable number of arguments of the same type to a method.
- The parameter in the method is declared as follows:

```
typeName... parameterName
```

```
public class javaapplication09 {
   public static void main(String args[]) {
      // Call method with variable args
               printMax(34, 3, 3, 2, 56.5);
      printMax(new double[]{1, 2, 3});
   }

   public static void printMax( double... numbers) {
   if (numbers.length == 0) {
      System.out.println("No argument passed");
      return;
```

```
}
  double result = numbers[0];
  for (int i = 1; i <  numbers.length; i++)
    if (numbers[i] >  result)
    result = numbers[i];
    System.out.println("The max value is " + result);
  }
}
```

```
The max value is 56.5
The max value is 3.0
```



# 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);
```

- Pass the file reader object to buffer reader to speedup the operation

```
BufferedReader br=new BufferedReader(fr);
```

- Now read lines from text file

```
String aLine=br.readLine()
```

## The whole code

```
Int readFile (String path) throws IOException
   {
            int numberOfLines = 0;
           Try{
                FileReader fr=new FileReader(path);
                BufferedReader br=new BufferedReader(fr);
                String aLine;
           while((aLine=br.readLine())!=null)
                     {          numberOfLines++;
                                System.out.println(aLine);
                     }
                br.close();}
           Ctach(IOException e) {}
     return numberOfLines;
   }
```



## 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)

```
WriteFile wf = new WriteFile( fileName ,true);
```

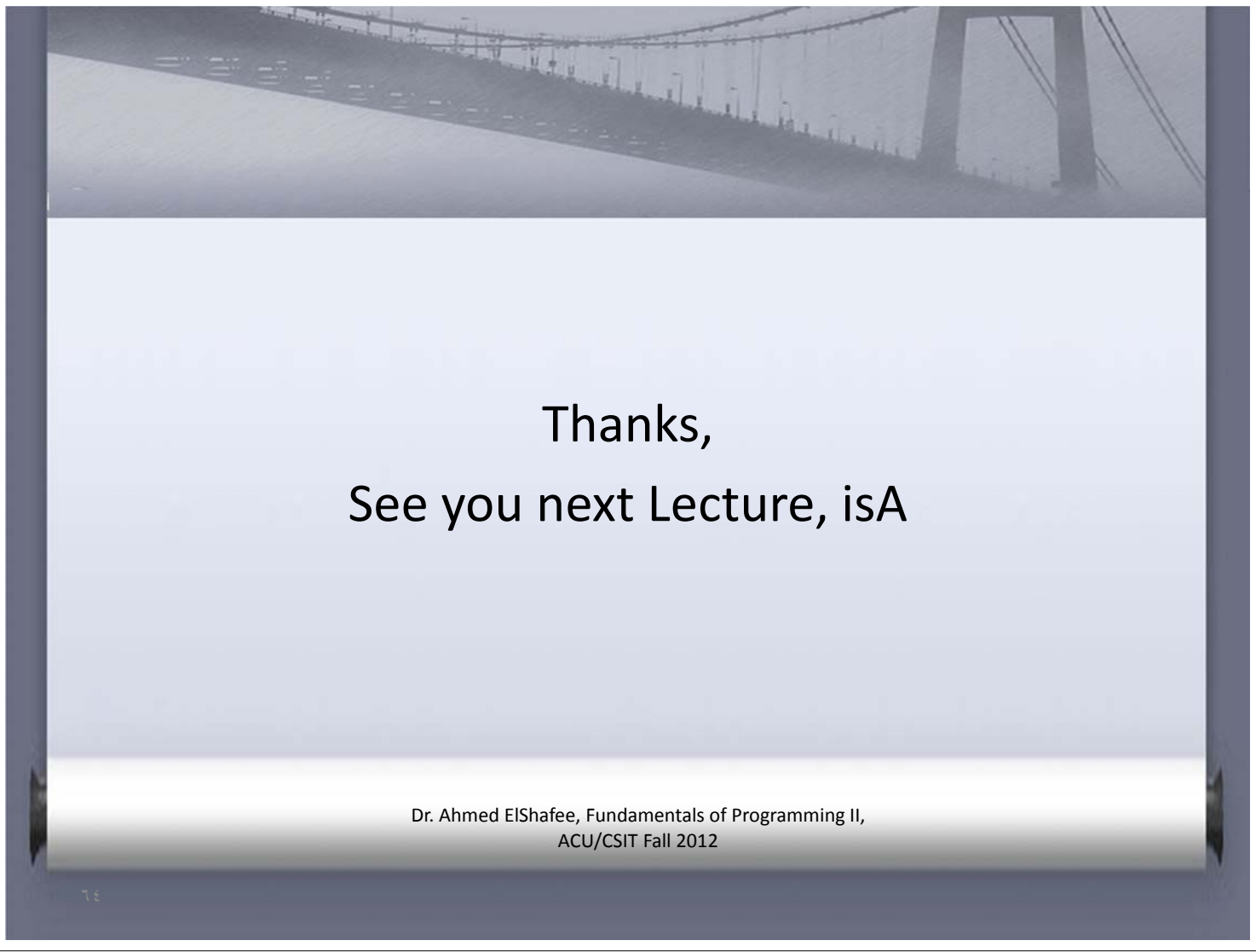
- Write line to a file

```
wf.writeToFile("This is another line of text");
```

## Here is the code

```
try
        {
            WriteFile wf = new WriteFile( fileName ,true);
            wf.writeToFile("This is another line of text");
            System.out.println( "Text File Written To" );
        }
        catch (IOException e)
        {
            System.out.println(e.getMessage());
        }
```





Thanks,

See you next Lecture, isA