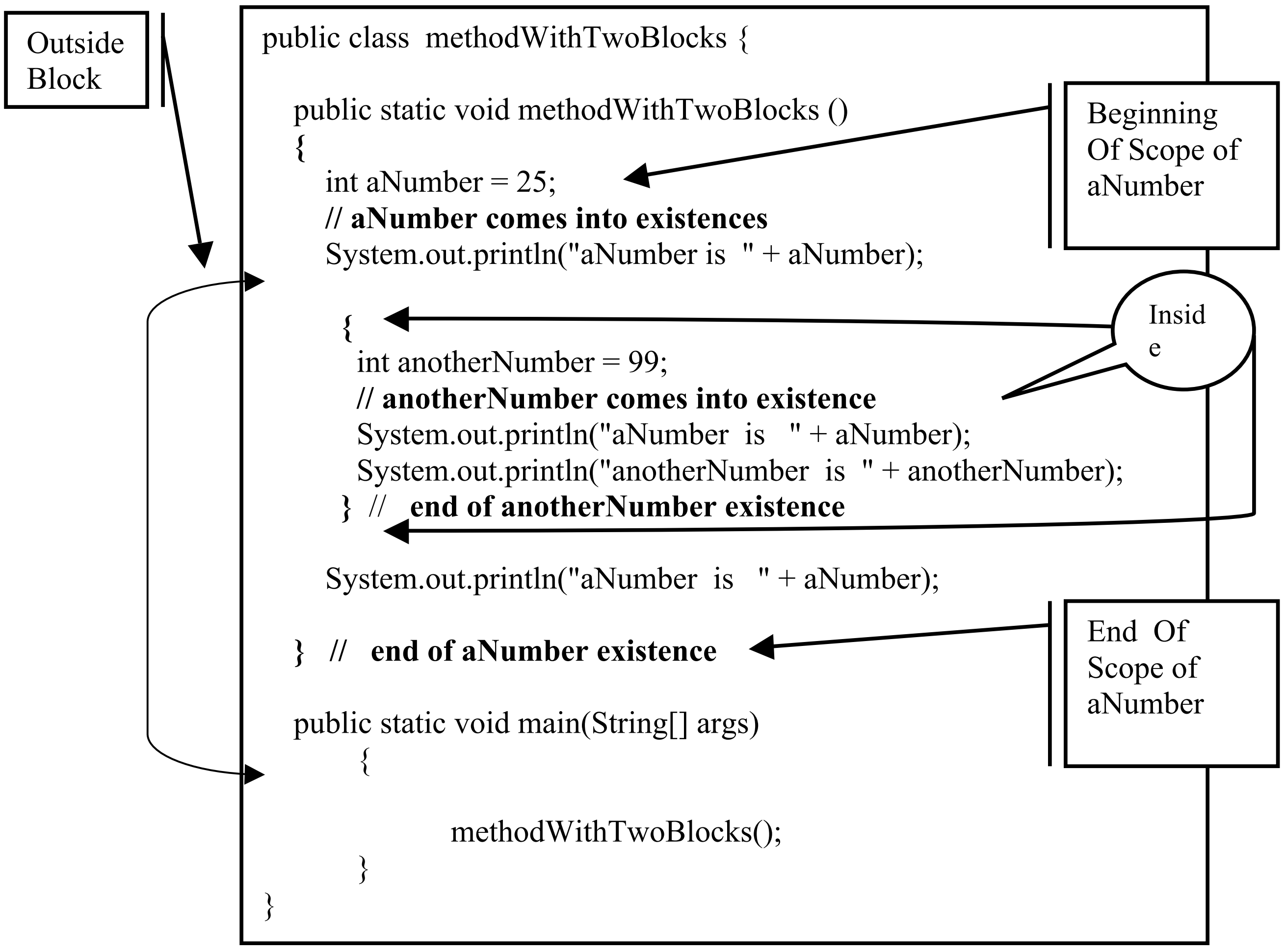
Advanced Object Concepts

Understanding Blocks

- **Blocks** - Appears within any class or method, the code between a pair of curly braces
  - Outside block- The first block, begins immediately after the method declaration and ends at the end of the method
  - Inside block- The second block, contained within the second pair of curly braces
  - The inside block is nested within the outside block



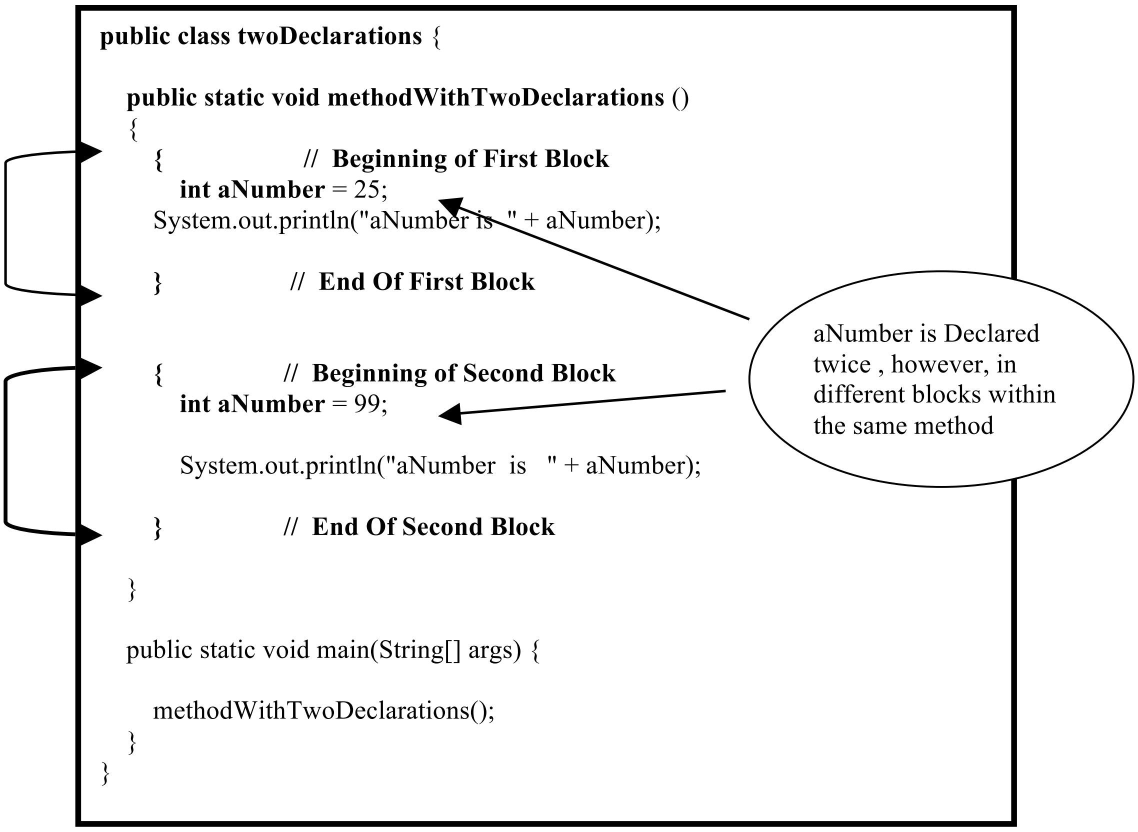
```
public class  methodWithTwoBlocks {

   public static void methodWithTwoBlocks ()
   {
      int aNumber = 25;
      // aNumber comes into existences
      System.out.println("aNumber is  " + aNumber);

       {
         int anotherNumber = 99;
         // anotherNumber comes into existence
         System.out.println("aNumber  is   " + aNumber);
         System.out.println("anotherNumber  is  " + anotherNumber);
       }  //   end of anotherNumber existence

      System.out.println("aNumber  is   " + aNumber);

   }  //   end of aNumber existence

   public static void main(String[] args)
         {

                 methodWithTwoBlocks();
         }
}
```

Understanding Scope

- The portion of a program within which you can reference a variable
- A variable comes into existence, or comes into scope, when you declare it
- A variable ceases to exist, or goes out of scope, at the end of the block in which it is declared

Declaring variables multiple times

Within a method, you can declare a variable with the same name multiple times, **as long as** each declaration is in its own, ***none overlapping*** block.



```
public class twoDeclarations {

  public static void methodWithTwoDeclarations ()
  {
    {               // Beginning of First Block
      int aNumber = 25;
    System.out.println("aNumber is  " + aNumber);

    }               // End Of First Block


    {               // Beginning of Second Block
      int aNumber = 99;

      System.out.println("aNumber  is   " + aNumber);

    }               // End Of Second Block

  }

  public static void main(String[] args) {

    methodWithTwoDeclarations();
  }
}
```

aNumber is Declared twice , however, in different blocks within the same method

The above declaration is valid because each variable is contained within its own block. The first instance of aNumber has gone out of scope before the second instance comes into scope.

Note : you can not declare the same variable more than once within a block.

If you declare a variable within a class, and use the same variable name within a method of the class, the variable used inside the method take precedence, or **overrides**, the first variable.

```
public class Employee {

    private int aNum = 44;
    private int aDept = 55;

   public  void empMethod ()
   {
      int aNum = 88;     //  aNum Overrides the class variable name
      System.out.println("aNum in empMethod   is   " + aNum);
      System.out.println("aDept  is   " + aDept);
   }

   public  void anotherEmpMethod ()
   {
      System.out.println("aNum in anotherEmpMethod  is  "  +  aNum);
      System.out.println("aDept  is   " + aDept);
   }

}
```

```
public class TheMainClass {

    public static void main(String[] args) {

      Employee Admin = new Employee ();
      Admin.empMethod();
      Admin.anotherEmpMethod();
   }


}
```

Note : *What happens if I want to use the class variable aNum that is outside the method empMethod () ??????*

Overloading a Method

Overloading:

- Involves using one term to indicate diverse meanings
- Writing multiple methods with the same name, but with different arguments
- Overloading a Java method means you write multiple methods with a shared name

```
public class OverLoad1 {

   public static void simpleInterest ( double bal , double rate ) {
        double interest ;
        interest = bal * rate;
        System.out.print("Interest On  " + bal + " at  " + rate  );
        System.out.println(" interest rate is  " + interest);
   }

}
```

Has 2 double arguments

Signature

```
public static void simpleInterest ( double bal , int rate ) {
        double interest , rateAsPercent;
        rateAsPercent  = rate / 100.0;
        interest = bal * rateAsPercent;
        System.out.print("Interest On  " + bal + " at  " + rate  );
        System.out.println(" interest rate is  " + interest);
   }
```

Has a double and an integer arguments

```
public static void main(String[] args) {

        simpleInterest ( 1000.0 , 0.04);
        simpleInterest ( 1000.0 , 4);

   }
```

Learning about Ambiguity

- When you overload a method you run the risk of ambiguity
  - An ambiguous situation is one in which the compiler cannot determine which method to use.

```
public static void main(String[] args) {

      simpleInterest ( 1000.0 , 0.04);
      simpleInterest ( 1000.0 , 4);
      simpleInterest ( 1000 , 4);


   }
```

This is calling a method with (int , int) arguments. There is no method that matchs the (in , int) arguments. Which method will be executed, :-
public static void simpleInterest (double bal , int rate) or
public static void simpleInterest (double bal , double rate)

Sending Arguments to Constructors

- Java automatically provides a constructor method when you create a class
- Programmers can write their own constructor classes
- Programmers can also write constructors that receive arguments
  - Such arguments are often used for **initialization** purposes when values of objects might vary

```
class Chap3EventSite  {

   private int siteNumber;
   private double usageFee;
   private String managerName;
   Chap3EventSite (int siteNum){
        siteNumber = siteNum;
       managerName = "ZZZ";
   }
// getManagerName() gets managerName
   public String getManagerName()  {     return managerName;  }

// getSiteNumber() gets the siteNumber
   public int getSiteNumber()  {     return siteNumber;   }

// getUsageFee() gets the usageFee
   public double getUsageFee() {     return usageFee;  }

// setManagerName() assigns a name to the manager
   public void setManagerName(String name)  {     managerName = name;
}

// setSiteNumber() assigns a site number
   public void setSiteNumber(int n)  {  siteNumber = n; }
//setUsageFee() assigns a value to the usageFee figure
   public void setUsageFee(double amt) {   usageFee = amt; }


//setUsageFee() assigns a value to the usageFee figure
   public void setUsageFee(double amt) {   usageFee = amt; }


}
```

Initialization the value with 100

```
public class Chap3SetUpSite {

   public static void main(String args[])
{
   Chap3EventSite oneSite = new Chap3EventSite(100);
   System.out.println(oneSite.getSiteNumber());

   oneSite.setUsageFee(32508.65);
   oneSite.setManagerName("Jefferson");
   oneSite.setSiteNumber(678);

   System.out.print("The number of the event site is ");
   System.out.println(oneSite.getSiteNumber());
   System.out.println("Usage fee " + oneSite.getUsageFee());
   System.out.println("Manager is " + oneSite.getManagerName());
}

}
```

Overrides the initial value

Overloading Constructors

- If you create a class from which you instantiate objects, Java automatically provides a constructor
- But, if you create your own constructor, the automatically created constructor no longer exists
- As with other methods, you can overload constructors
  - Overloading constructors provides a way to create objects with or without initial arguments, as needed

```
class Chap3EventSite  {

   private int siteNumber;
   private double usageFee;
   private String managerName;

   Chap3EventSite ( int siteNum ){
        siteNumber = siteNum;
       managerName = "ZZZ";
  }

  Chap3EventSite ( ) {
        siteNumber = 999;
       managerName = "ZZZ";
  }

// getManagerName() gets managerName
   public String getManagerName()  {     return managerName;  }

// getSiteNumber() gets the siteNumber
   public int getSiteNumber()  {     return siteNumber;   }

// getUsageFee() gets the usageFee
   public double getUsageFee() {    return usageFee;  }

// setManagerName() assigns a name to the manager
   public void setManagerName(String name)  {     managerName = name;
}

……….
………
………..

}
```

Overloaded Constructor

```
public class Chap3SetUpSite {

   public static void main(String args[])
{
   Chap3EventSite oneSite = new Chap3EventSite();
   System.out.println(oneSite.getSiteNumber());

   oneSite.setUsageFee(32508.65);
   oneSite.setManagerName("Jefferson");
   oneSite.setSiteNumber(678);

   System.out.print("The number of the event site is ");
   System.out.println(oneSite.getSiteNumber());
   System.out.println("Usage fee " + oneSite.getUsageFee());
   System.out.println("Manager is " + oneSite.getManagerName());
}

}
```

Java stores only one copy of every method of a class, although many objects may be instantiated from that class. One copy of a class method is stored and used by all instantiated objects. When an object method is called you specify the object name, a dot and the method name. You are referring to the shared copy of the method stored for all objects. When you access a field of the object, however, you are referring to that objects individual copy of the data member.

The compiler knows which object's data member is being referred to, because an implicit reference to the object, the `this` reference, is automatically passed. The keyword `this` is a reserved word in Java. You normally do not need to refer to the `this` reference within the methods that you write.

Static methods, or class methods, do not have a `this` reference. In addition to static methods it is possible to create static variables, or class variables, that are shared by every instantiation of a class.

Learn about the this Reference

```
public class Employee {

    private int aNum = 44;
    private int aDept = 55;

   public  void empMethod ()
   {
      int aNum = 88;     //  aNum Overrides the class variable name
      System.out.println("aNum in empMethod   is   " + aNum);
      System.out.println("aDept  is    " + aDept);
   }

   public  void anotherEmpMethod ()
   {
      System.out.println("aNum in anotherEmpMethod  is  "  +  aNum);
      System.out.println("aDept  is    " + aDept);
   }

}
```

```
System.out.println("aNum in empMethod  using this is
" + this.aNum);
```

- How would you reference the field *aNum* that is outside the method empMethod() from inside the method empMethod() ??

- **how would you reference a field of an object after it has been instantiated from a class?????**

Java stores only one copy of every method of a class, although many objects may be instantiated from that class. One copy of a class method is stored and used by all instantiated objects. When an object method is called you specify the object name, a dot and the method name. You are referring to the shared copy of the method stored for all objects. When you access a field of the object, however, you are referring to that objects individual copy of the data member.

The compiler knows which object's data member is being referred to, because an implicit reference to the object, the **`this` reference**, is automatically passed. The keyword `this` is a reserved word in Java. You normally do not need to refer to the `this` reference within the methods that you write.

Static methods, or class methods, do not have a `this` reference. In addition to static methods it is possible to create static variables, or class variables, that are shared by every instantiation of a class.

```
public class Chap4EventSite8 {

   private int siteNumber;
   static final public String HEADQUARTERS = "Crystal Lake, IL ";
  // getSiteNumber() gets the siteNumber
  public int getSiteNumber()
  {
       return siteNumber;
  }
  // setSiteNumber() assigns a site number
  public void setSiteNumber(int n)
  {
       siteNumber = n;
  }

}
```

return this.siteNumber

Working with Constants

Class variables- Variables that are shared by every instantiation of a class

Constant variable:

- A variable or data field that should not be changed during the execution of a program
    - To prevent alteration, use the keyword **final**
- ***Constant fields are written in all uppercase letters***
    - For example:
        - COMPANY_ID

```
public class Employee {

        static **final** private int **COMPANY_ID** =  12345;
…………
……….
}
```

Example

```
public class Chap4EventSite8 {

  private int siteNumber;
  static final public String HEADQUARTERS = "Crystal Lake, IL ";
 // getSiteNumber() gets the siteNumber
 public int getSiteNumber()
 {
     return siteNumber;
 }
 // setSiteNumber() assigns a site number
 public void setSiteNumber(int n)
 {
     siteNumber = n;
 }

}
```

HEADQUARTERS is a constant variable

```
public class Chap4SetUpSite8 {
   public static void main(String args[])
  {
       Chap4EventSite8 oneSite = new Chap4EventSite8();
       Chap4EventSite8 anotherSite = new Chap4EventSite8();
       int number;
       oneSite.setSiteNumber(101);
       anotherSite.setSiteNumber(102);

       System.out.print("The number of one site is ");
       System.out.println(oneSite.getSiteNumber());
       System.out.println("Headquarters located at  " + oneSite.HEADQUARTERS);
       System.out.print("The number of another site is ");
       System.out.println(anotherSite.getSiteNumber());
       System.out.println("Headquarters located at  " + anotherSite.HEADQUARTERS);

  }

}
```

Using Automatically Imported, Prewritten Constants and Methods

- The creators of Java created nearly 500 classes
  - For example:
    - System, Character, Boolean, Byte, Short, Integer, Long, Float, and Double are classes
- These classes are stored in a package, or a library of classes, which is a folder that provides a convenient grouping for classes

- **java.lang** – The package that is **implicitly** imported into every Java program and contains fundamental classes, or basic classes
- Fundamental classes include:
  - System, Character, Boolean, Byte, Short, Integer, Long, Float, and Double
- Optional classes – Must be explicitly named

- To use any of the prewritten classes (other than java.lang):
  - Use the entire path with the class name **OR**
  - Import the class **OR**
  - Import the package which contains the class you are using

- To import an entire package of classes use the wildcard symbol
  - * For example: **import java.util.*;**
  - Represents all the classes in a package

Examples

```
//import java.lang.*;

   public class MyMath {
   public static void main(String[] args) {
   final int MyVal=10;
   System.out.println("The My Value Is :"+MyVal);
   System.out.println("The Absolute Value  Is :"+java.lang.Math.abs(-1*MyVal));
   System.out.println("The PI Value Is :"+Math.PI);
   System.out.println("The PI Value Is :"+Math.abs(-MyVal));
   System.out.println("The Sine Value Is :"+Math.sin(MyVal));
   System.out.println("The Cosine Value Is :"+Math.cos(Math.PI/3));
   System.out.println("The ceiling Value Is :"+Math.ceil(33.01));
   System.out.println("The ceiling Value Is :"+Math.ceil(2.00000000001));
   System.out.println("The ceiling Value Is :"+Math.ceil(2.0));
   System.out.println("The Exp Value Is :"+Math.exp(1.0));
   System.out.println("The Floor Value Is :"+Math.floor(2.0000001));
   System.out.println("The Floor Value Is :"+Math.floor(2.999999999));
   System.out.println("The Floor Value Is :"+Math.floor(3.0));
   System.out.println("The The log Value Is :"+Math.log(1));
   System.out.println("The max Value Is :"+Math.max(1,3));
   System.out.println("The min Value Is :"+Math.min(1,3));
   System.out.println("The pow Value Is :"+Math.pow(2,10));
   System.out.println("The random Value Is :"+Math.random());
   System.out.println("The random Value Is :"+(5000+  (  Math.ceil(Math.random()*2001))));
   System.out.println("The rint Value Is :"+Math.rint(3.5));
   System.out.println("The round Value Is :"+Math.round(3.5));
   System.out.println("The rint Value Is :"+Math.rint(3.499999));
   System.out.println("The round Value Is :"+Math.round(3.499999));
   System.out.println("The square root Value Is :"+Math.sqrt(16));

  }
}
```

import an entire package of classes

Use the entire path with the class name

Method within a class

Answer

```
The My Value Is :10
The Absolute Value  Is :10
The PI Value Is :3.141592653589793
The PI Value Is :10
The Sine Value Is :-0.5440211108893698
The Cosine Value Is :0.5000000000000001
The ceiling Value Is :34.0              ( Smallest int value not <  X )
The ceiling Value Is :3.0
The ceiling Value Is :2.0
The Exp Value Is :2.7182818284590455
The Floor  Value Is :2.0                ( Largest int value not > X )
The Floor Value Is :2.0
The Floor Value Is :3.0
The The log Value Is :0.0
The max Value Is :3
The min Value Is :1
The pow Value Is :1024.0
The random Value Is :0.6413030055657253
The random Value Is :6324.0
The rint Value Is :4.0
The round Value Is :4
The rint Value Is :3.0
The round Value Is :3
The square root Value Is :4.0
```

Learning about the Gregorian Calendar

- The Gregorian calendar is the calendar used in most of the western world
  - There are seven constructors for GregorianCalendar objects
  - The default creates a calendar with the current date and time in the default locale
  - You can use other constructors to specify the year, month, day, hour, minute, and second
  - You create a calendar object with the default constructor

```
GregorianCalendar calendar = new GregorianCalendar();
```

Information such as the day, month, and year can be retrieved from a GregorianCalendar object by using a class get() method, and then specifying what you want as an argument. Some of the possible arguments to the get() method are shown in Table 4-2.

Note : You need to add the following line

import java.util.;*

Examples

```
import java.util.*;

public class MyDate {


  public static String convertNumberToAMORPM(int a){
    String TBReturnd="PM";
    if(a==0) {TBReturnd="AM";}
    return TBReturnd;
  }


 public static void main(String[] args) {

   Date a = new Date();

   System.out.println("Today is :"+a);

   GregorianCalendar ACal=new GregorianCalendar();

   System.out.println("Year  =  "+ACal.get(ACal.YEAR));
   System.out.println("Year  =  "+ACal.get(GregorianCalendar.YEAR));
   System.out.println("Month =  "+ACal.get(java.util.GregorianCalendar.MONTH));
   System.out.println("Day Of Month  =  "+ACal.get(GregorianCalendar.DAY_OF_MONTH));
   System.out.println("Day Of Week  =  "+ACal.get(GregorianCalendar.DAY_OF_WEEK));
   System.out.println("Day Of Year  =  "+ACal.get(GregorianCalendar.DAY_OF_YEAR));
   System.out.println("Hour Min  Sec   Am  or PM =  "+ACal.get(GregorianCalendar.HOUR)
                   +":"+ ACal.get(GregorianCalendar.MINUTE)
                   +":"+ ACal.get(GregorianCalendar.SECOND)
                   +"  "+ MyDate.convertNumberToAMORPM(
                        ACal.get(GregorianCalendar.AM_PM)));

 }
}
```

Answer

```
Today is :Sat Mar 13 12:33:00 CST 2010
Year  =  2010
Year  =  2010
Month =  2
Day Of Month  =  13
Day Of Week  =  7
Day Of Year  =  72
Hour Min  Sec   Am  or PM =  0:33:0  PM
BUILD SUCCESSFUL (total time: 0 seconds)
```

Class StringBuffer

A string buffer is like a `String`, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

String buffers are safe for use by multiple threads. The methods are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.

String buffers are used by the compiler to implement the binary string concatenation operator +. For example, the code:

```
     x = "a" + 4 + "c"
```

is compiled to the equivalent of:

```
 x = new StringBuffer().append("a").append(4).append("c")
                           .toString()
```

which creates a new string buffer (initially empty), appends the string representation of each operand to the string buffer in turn, and then converts the contents of the string buffer to a string. Overall, this avoids creating many temporary strings.

Some of the functions that are used are :-

append()
This is the append() function used for the concatenate the string in string buffer. This is better to use for dynamic string concatenation. This function works like a simple string concatenation such as : String str = str + "added string";.

insert()
This is the insert() function used to insert any string or character at the specified position in the given string.

reverse()
This is the reverse() function used to reverse the string present in string buffer.

setCharAt()
This is the setCharAt() function which is used to set the specified character in buffered string at the specified position of the string in which you have to set the given character.

charAt()
This is the charAt() function which is used to get the character at the specified position of the given string.

substring()
This is the substring() function which is used to get the sub string from the buffered string from the initial position to end position (these are fixed by you in the program).

deleteCharAt()
This is the deleteCharAt() function which is used to delete the specific character from the buffered string by mentioning that's position in the string.

length()
This is the length() function is used to finding the length of the buffered string.

delete()
This is the delete() function is used to delete multiple character at once from n position to m position (n and m are will be fixed by you.) in the buffered string.

capacity()
This is the capacity() function is used to know about the current characters kept which is displayed like : number of characters + 16.

```
package ch6stringobjectsasparameters;

/**
 *
 * @author husaingholoom
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {


        String str = "Hello";

        System.out.println("str before "
                         + "calling the method "
                         + "stringParameter: "+ str);  // Hello

        stringParameter(str);

        System.out.println("str after "
                         + "calling the method "
                         + "stringParameter: "+ str);  //  Hello
    } //end main

    public static void stringParameter(String pStr)
    {
        System.out.println("In the method "
                         + "stringParameter");
        System.out.println("pStr before "
                         + "changing its value: "
                         + pStr);                     // Hello

        pStr = "Sunny Day";
        System.out.println("pStr after "
                         + "changing its value: "
                         + pStr);             // Sunny Day

    }

}
```

```
package ch6stringbufferobjectsasparameters;

/**
 *
 * @author husaingholoom
 */
public class Main {

    /**
     * @param args the command line arguments
     */
   public static void main(String[] args)
    {
        StringBuffer str = new StringBuffer("Hello");

        System.out.println("str before "
                         + "calling the method "
                         + "stringBufferParameter: "
                         + str);              // Hello

        stringBufferParameter(str);

        System.out.println("str after "
                         + "calling the method "
                         + "stringBufferParameter: "
                         + str); //Hello There, How are you doing
    } //end main

    public static void stringBufferParameter
                                (StringBuffer pStr)
    {
       System.out.println("In the method "
                        + "stringBufferParameter ");
       System.out.println("pStr before "
                        + "changing its value: "
                        + pStr);       // Hello

       pStr.append(" There, ");
       pStr.append("   How are you doing");

       System.out.println("pStr after "
                        + "changing its value: "
                        + pStr);  //Hello There, How are you doing
    } //end stringBufferParameter
}
```

```
package ch6stringbuffer;
/**
 *  Exaple of StringBuffer Functions
 * @author Husain Gholoom
 */
import  java.util.* ;
public class Main {
   static Scanner console = new Scanner (System.in);
    public static void main(String[] args) {

      System.out.print("Enter your name: ");
      String  str = console.next();
      str += ", This is the example of SringBuffer class and it's functions.";

      //Create a object of StringBuffer class
      StringBuffer strbuf = new StringBuffer();
      System.out.println("capcity   "+strbuf.capacity());  // Capcity is 16
      strbuf.append(str);
      System.out.println(strbuf);
      strbuf.delete(0,str.length());
      System.out.println(strbuf);     //  length is   zero
      //append()
      strbuf.append("Hello");
      strbuf.append("World");             //print HelloWorld
      System.out.println(strbuf);
      //insert()
      strbuf.insert(5,"_Java ");           //print Hello_Java World
      System.out.println(strbuf);
      //reverse()
      strbuf.reverse();
      System.out.print("Reversed string : ");
      System.out.println(strbuf);           //print dlroW avaJ_olleH
      strbuf.reverse();
      System.out.println(strbuf);           //print Hello_Java World
      //setCharAt()
      strbuf.setCharAt(5,' ');
      System.out.println(strbuf);           //print  Hello Java World
      //charAt()
      System.out.print("Character at 6th position : ");
      System.out.println(strbuf.charAt(6));      //print J
      //substring()
      System.out.print("Substring from position 3 to 6 : ");
      System.out.println(strbuf.substring(3,7));    //print lo J
      //deleteCharAt()
      strbuf.deleteCharAt(3);
      System.out.println(strbuf);           //print Helo java World
      //capacity()
      System.out.print("Capacity of StringBuffer object : ");
      System.out.println(strbuf.capacity());      //print the capacity
      //length()
      System.out.println(strbuf.length());      //print the length (15)
      //delete() and length()
      strbuf.delete(6,strbuf.length());
      System.out.println(strbuf);           //no anything
  }   }
```