String Object

- String is a sequence of characters.
- Unlike many other programming languages that implements string ascharacter arrays, Java implements strings as object of type String.
- This provides a full compliment of features that make string handling convenient. For example, Java String has methods to:

1) compare two strings
2) Search for a substring
3) Concatenate two strings and
4) Change the case of letters within a string
5) Can be constructed a number of ways making it easy to obtain a string when needed

String is Immutable

Once a String Object has been created, you cannot change the characters that comprise that string.

This is not a restriction. It means each time you need an altered version of an existing string, a new string object is created that contains the modification.

It is more efficient to implement immutable strings than changeable ones.

To solve this, Java provides a companion class to String called StringBuffer.

StringBuffer objects can be modified after they are created.

String Constructors 1

String supports several constrictors:

1) to create an empty String

```
String s = new String();
```

2) to create a string that have initial values

```
String(char chars[])
```

Example:

```
char chars[] = {‘a’,’b’,’c’};
String s = new String(chars);
```

String Constructors 2

3) to create a string as a subrange of a character array

String(char chars[], int startindex, int numchars)

Here, startindex specifies the index at which the subrange begins,and numChars specifies the number of characters to use.

Example:

```
char chars[] = {‘a’,’b’,’c’,’d’,’e’,’f’};
String s = new String(chars,2,3);
```

This initializes s with the characters cde.

String Constructors

4) to construct a String object that contains the same character sequence as another String object

String(String obj)

- Example

```
class MakeString {
    public static void main(String args[]) {
    char c[] = {'J', 'a', 'v', 'a'};
    String s1 = new String(c);
    String s2 = new String(s1);
    System.out.println(s1);
    System.out.println(s2);
    }
}
```

String Length

The length of a string is the number of characters that it contains.

To obtain this value call the length() method:

```
int length()
```

The following fragment prints “3”, since there are three characters in the string s.

```
char chars[] = {‘a’,’b’,’c’};
String s = new String(chars);
System.out.println(s.lenght());
```

String Operations

Strings are a common and important part of programming.

Java provides several string operations within the syntax of the language.

These operations include:

1) automatic creation of new String instances from literals
2) concatenation of multiple String objects using the + operator
3) conversion of other data types to a string representation.

There are explicit methods to perform all these functions, but Java does them automatically for the convenience of the programmer and to add clarity.

String Literals

Using String literals is an easier way of creating Strings Objects.

For each String literal, Java automatically constructs a String object.You can use String literal to initialize a String object.

- Example:

```
char chars[] = {‘a’,’b’,’c’};
String s1 = new String(chars);
```

Using String literals

```
String s2 = “abc”;
```

String Concatenation

Java does not allow operations to be applied to a String object.

The one exception to this rule is the + operator, which concatenates two strings producing a string object as a result.

With this you ca chain together a series of + operations.

Example:

```
String age = “9”;
String s = “He is ” + age + “ years old.”;
System.out.println(s);
```

Concatenation Usage

One practical use is found when you are creating very long strings.

Instead of letting long strings wrap around your source code, you can break them into smaller pieces, using the + to concatenate them.

Example:

```
class ConCat {
    public static void main(String args[]) {
      String longStr = "This could have been " +
      "a very long line that would have " +
      "wrapped around. But string concatenation "
      + "prevents this.";
      System.out.println(longStr);
    }
}
```

Concatenation & Other Data Type

You can concatenate Strings with other data types.

Example:

```
int age = 9;
String s = “He is ” + age + “ years old.”;
System.out.println(s);
```

The compiler will convert an operand to its string equivalent whenever the other operand of the + ia an instance of String.

Be careful:

```
String s = “Four:” + 2 + 2;
System.out.println(s);
```

Prints Four:22 rather than Four: 4.

To achieve the desired result, bracket has to be used.

```
String s = “Four:” + (2 + 2);
```

Conversion and toString() Method

- When Java converts data into its string representation during concatenation, it does so by calling one of its overloaded valueOf() method defined by String.

  valueOf() is overloaded for

  1) simple types – which returns a string that contains the human -readable equivalent of the value with which it is called.

  2) object types – which calls the toString() method of the object.

Example: toString() Method 1

- Override toString() for Box class

```
class Box {
        double width;
        double height;
        double depth;

        Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
}
```

Example: toString() Method 2

```
public String toString() {
        return "Dimensions are " + width + "
by " +    depth + " by " + height + ".";
    }
}
```

```
class toStringDemo {
        public static void main(String args[]) {
                Box b = new Box(10, 12, 14);
                String s = "Box b: " + b; // concatenate Box object
                System.out.println(b); // convert Box to string
                System.out.println(s);
        }
}
```

- Box's toString() method is automatically invoked when a Box object is used in a concatenation expression or in a Call to println().
- The output of this program is shown here:
- Dimensions are 10.0 by 14.0 by 12.0
- Box b: Dimensions are 10.0 by 14.0 by 12.0

Character Extraction

- String class provides a number of ways in which characters can be extracted from a String object.
- String index begin at zero.
- These extraction methods are:
    1) charAt()
    2) getChars()
    3) getBytes()
    4) toCharArray()
- Each will considered.

charAt()

- To extract a single character from a String.
- General form:

  char charAt(int where)

  where is the index of the character you want to obtain. The value of where must be nonnegative and specify allocation within the string.
- Example:

  char ch;

  ch = “abc”.charAt(1);
- Assigns a value of “b” to ch.

getChars()

- Used to extract more than one character at a time.
- General form:

```
void getChars(int sourceStart, int sourceEnd,
              char[]target, int targetStart)
```

- sourceStart – specifies the index of the beginning of the substring
- sourceEnd – specifies an index that is one past the end of the desired subString
- target – is the array that will receive the characters.
- targetStart – is the index within target at which the subString will be copied is passed in this parameter

getChars()

```
class getCharsDemo {
public static void main(String args[]) {
    String s = "This is a demo of the getChars
                                        method.";
    int start = 10;
    int end = 14;
    char buf[] = new char[end - start];
    s.getChars(start, end, buf, 0);
    System.out.println(buf);
  }
}
```

getBytes()

- Alternative to getChars() that stores the characters in an array of bytes. It

- uses the default character-to-byte conversions provided by the platform.
- General form:

byte[] getBytes()

- Usage:
- Most useful when you are exporting a String value into an environment thatdoes not support 16-bit Unicode characters.
- For example, most internet protocols and text file formats use 8-bit ASCII forall text interchange.

toCharArray()

To convert all the characters in a String object into character array.

It returns an array of characters for the entire string.

General form:

```
char[] toCharArray()
```

- It is provided as a convenience, since it is possible to use getChars() to achieve the same result.

String Comparison

- The String class includes several methods that compare strings or substrings within strings.
- They are:
    1) equals() and equalsIgnoreCase()
    2) regionMatches()
    3) startWith() and endsWith()
    4) equals() Versus = =
    5) comapreTo()
- Each will be considered.

equals()

- To compare two Strings for equality, use equals()
- General form:

  boolean equals(Object str)

- str is the String object being compared with the invoking String object.
- It returns true if the string contain the same character in the same order, and false otherwise.
- The comparison is case-sensitive.

equalsIgnoreCase()

- To perform operations that ignores case differences.
- When it compares two strings, it considers A-Z as the same as a-z.
- General form:

  boolean equalsIgnoreCase(Object str)

- str is the String object being compared with the invoking String object.
- It returns true if the string contain the same character in the same order, and false otherwise.
- The comparison is not case-sensitive.

equals and equalsIgnoreCase() 1

- Example:

```
class equalsDemo {
    public static void main(String args[]) {
        String s1 = "Hello";
        String s2 = "Hello";
        String s3 = "Good-bye";
        String s4 = "HELLO";
        System.out.println(s1 + " equals " + s2 +
                           "->"+s1.equals(s2));
        System.out.println(s1 + " equals " + s3 + "
                           -> " +s1.equals(s3));
```

equals and equalsIgnoreCase()

```
            System.out.println(s1 + " equals " + s4 +
                                        " -> " +s1.equals(s4));
            System.out.println(s1 + " equalsIgnoreCase
“            + s4 +" -> " + s1.equalsIgnoreCase(s4));
    }
```

- }
- The output from the program is shown here:
- Hello equals Hello -> true
- Hello equals Good-bye -> false
- Hello equals HELLO -> false
- Hello equalsIgnoreCase HELLO -> true

startsWith() and endsWith() 1

The startsWith() method determines whether a given string begins with a specified string.

- Conversely, endsWith() method determines whether the string in question ends with a specified string.
- General form:

  boolean startsWith(String str)

  boolean endsWith(String str)

- str is the String being tested. If the string matches, true is returned,otherwise false is returned.

startsWith() and endsWith() 1

- Example:
- “Foobar”.endsWith(“bar”);
- and
- “Foobar”.startsWith(“Foo”);
- are both true.

startsWith() and endsWith() 1

- A second form of startsWith(), let you specify a starting point:
- General form:

  boolean startWith(String str, int startIndex)

- Where startIndex specifies the index into the invoking string at which point the search will begin.
- Example:
- “Foobar”.startsWith(“bar”, 3);
- returns true.

equals() Versus ==

- It is important to understand that the two method performs different functions.
- 1) equals() method compares the characters inside a String object.
- 2) == operator compares two object references to see whether they refer to the same instance.

Example: equals() Versus ==

- class EqualsNotEqualTo {
  public static void main(String args[]) {
  String s1 = "Hello";
  String s2 = new String(s1);
  System.out.print(s1 + " equals " + s2 + " -> “);
  System.out.println(s1.equals(s2));
  System.out.print(s1 + " == " + s2 + " -> “)
  System.out.println((s1 == s2));
  }
}
- O/p : Hello equals Hello -> True
- Hello == Hello -> False

compareTo()

- It is not enough to know that two Strings are identical. You need to know which is less than, equal to, or greater than the next.

- A string is less than the another if it comes before the other in the dictionary order.

- A string is greater than the another if it comes after the other in the dictionary order.

The String method compareTo() serves this purpose

compareTo()

- General form:

  int compareTo(String str)

- str is the string that is being compared with the invoking String. The result of the comparison is returned and is interpreted as shown here:

Less than zero	The invoking string is less than str
Greater than zero	The invoking string is greater than str
Zero	The two Zero strings are equal

Example: compareTo()

```
class SortString {
    static String arr[] = {"Now", "is", "the", "time", "for", "all",
        "good,” “men","to", "come", "to", "the", "aid", "of “,
        "their", "country"};
    public static void main(String args[]) {
        for(int i = 0;i < arr.length; i++) {
            for(int j =i + 1;j < arr.length;j++) {
            if(arr[i].compareTo(arr[j])> 0)
                {
                String t = arr[i]
                arr[i] = arr[j];
                arr[j] = t;
                }
            }
}
```

Searching String

- String class provides two methods that allows you search a string for a specified character or substring:
- 1) indexOf() – Searches for the first occurrence of a character or substring.
- 2) lastIndexOf() – Searches for the last occurrence of a charater or substring.
- These two methods are overloaded in several different ways. In all cases, the methods return the index at which the character or substring was found, or -1 on failure.

Searching String

- To seach for the first occurrence of a character, use

  int indexOf(int ch)

- To search for the last occurrence of a character, use

  int lastIndexOf(int ch)

- To search for the first and the last occurence of a substring, use

  int indexOf(String str)

  int lastIndexOf(String str)

- Here str specifies the substring.

Searching String

- You can specify a starting point for the search using these forms:
- int indexOf(int ch, int startIndex)
- int lastIndexOf(int ch, int startIndex)

- int indexOf(String str, int startIndex)
- int lastIndexOf(String str, int startIndex)
- startIndex – specifies the index at which point the search begins.
- For indexOf(), the search runs from startIndex to the end of the string.
- For lastIndexOf(), the search runs from startIndex to zero.

Example: Searching String 1

```
class indexOfDemo {
    public static void main(String args[]) {
        String s = "Now is the time for all good men to come
                                        to the aid of their country.";
        System.out.println(s);
        System.out.println("indexOf(t) = " +    s.indexOf('t'));
        System.out.println("lastIndexOf(t)="+s.lastIndexOf('t'));
        System.out.println("indexOf(the)="s.indexOf("the"));
        System.out.println("lastIndexOf(the) = " +
        s.lastIndexOf("the"));
```

```
System.out.println("indexOf(t, 10) = " +s.indexOf('t', 10));
System.out.println("lastIndexOf(t, 60) = "
                                        +s.lastIndexOf('t', 60));
System.out.println("indexOf(the, 10) = " +
                                        s.indexOf("the", 10));
System.out.println("lastIndexOf(the, 60) = " +
                                    s.lastIndexOf("the", 60));
    }
}
```

Modifying a String

- String object are immutable.
- Whenever you want to modify a String, you must either copy it into a StringBuffer or use the following String methods,, which will construct a new copy of the string with your modification complete.
- They are:
    1) subString()
    2) concat()
    3) replace()
    4) trim()
- Each will be discussed.

subString() 1

- You can extract a substring using subString().
- It has two forms:

String substring(int startIndex)

- startIndex specifies the index at which the substring will begin. This form returns a copy of the substring that begins at startIndex and runs to the end of the invoking string.

subString() 2

- The second form allows you to specify both the beginning and ending index of the substring.

String substring(int startIndex, int ensIndex)

- startIndex specifies the index beginning index, and
- endIndex specifies the stopping point.

- The string returned contains all the characters from the beginning index, upto, but not including, the ending index.

Example: subString()

```
class StringReplace {
    public static void main(String args[]) {
    String org = "This is a test.This is, too.";
    String search = "is";
            String sub = "was";
            String result = "";
            int i;
            do { // replace all matching substrings
            System.out.println(org);
            i = org.indexOf(search);
            if(i != -1) {
            result = org.substring(0, i);
```

```
            result = result + sub;
        result = result + org.substring(i +search.length());
            org = result;
            }
        } while(i != -1);
        }
}
```

concat()

- You can concatenate two string using concat()
- General form:

  String concat(String str)

- This method creates a new object that contains the invoking string with the contents of str appended to the end.
- concat() performs the same function as +.
- Example:

  String s1 =“one”;

  String s2 = s1.concat(“two”);

- Or

  String s2 = s1 + “two”;

replace()

- Replaces all occurences of one character in the invoking string with another character.

  General form:

  String replace(char original, char replacement)

- original – specifies the character to be replaced by the character specified by replacement. The resulting string is returned.

- Example:

  String s = “Hello”.replace(‘l’,’w’);
  Puts the string “Hewwo” into s.

trim()

- Returns a copy of the involving string from which any leading and trailing whitespace has been removed.

- General form:

  String trim();

- Example:

  String s = “ Hello world “.trim();

- This puts the string “Hello world” into s.
- It is quite useful when you process user commands.

Example: trim() 1

```
import java.io.*;
class UseTrim {
public static void main(String args[]) throws IOException{
BufferedReader br = new BufferedReader(new
                                    InputStreamReader(System.in));
        String str;
        System.out.println("Enter 'stop' to quit.");
        System.out.println("Enter State: ");
do {
        str = br.readLine();
```

```
        str = str.trim(); // remove whitespace
    if(str.equals("Illinois"))
        System.out.println("Capital is pringfield.");
    else if(str.equals("Missouri"))
        System.out.println("Capital is Jefferson City.");
```
-
```
    else if(str.equals("California"))
        System.out.println("Capital is Sacramento.");
    else if(str.equals("Washington"))
        System.out.println("Capital is Olympia.");
    } while(!str.equals("stop"));
}
}
```

Case of Characters

- The method toLowerCase() converts all the characters in a string fromuppercase to lowercase.
- The toUpperCase() method converts all the characters in a string fromlowercase to uppercase.
- Non-alphabetical characters, such as digits are unaffected.
- General form:

String toLowercase()

String toUppercase()

Example: Case of Characters

```
class ChangeCase {
public static void main(String args[]) {
String s = "This is a test.";
System.out.println("Original: " + s);
String upper = s.toUpperCase();
String lower = s.toLowerCase();
System.out.println("Uppercase: " + upper);
System.out.println("Lowercase: " + lower);
}
}
```

StringBuffer

- StringBuffer is a peer class of string that provides much of the functionality of Strings.
- String is immutable. StringBuffer represents growable and writable character sequence.
- StringBuffer may have characters and substring inserted in the middle or appended to the end.
- StringBuffer will automatically grow to make room for such additions and often has more characters pre allocated than are actually needed, to allow room for growth.

StringBuffer Constructors

- Defines three constructors:

  1) StringBuffer() – default and reserves room for 16 characters without reallocation

  2) StringBuffer(int size) – accepts an integer argument that explicitly sets the size of the buffer

  3) StringBuffer(String str) – accepts a String argument that initially sets the content of the StringBuffer Object and reserves room for more16 characters without reallocation.

Length() and capacity()

- Current length of a StringBuffer can be found via the length() method, while the total allocated capacity can be found throught the capacity() method.
- General form:

  int length()

  Int capacity()

Example: Length() and capacity()

```
class StringBufferDemo {
public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        System.out.println("buffer = " + sb);
        System.out.println("length = " + sb.length());
        System.out.println("capacity = " + sb.capacity());
        }
}
```

ensureCapacity()

- Use ensureCapacity() to set the size of the buffer in order to pre allocate room for a certain number of characters after a StringBuffer has been constructed.
- General form:

  void ensureCapacity(int capacity)

- Here, capacity specifies the size of the buffer.
- Usage:
- Useful if you know in advance that you ill be appending a large number of small strings to a StringBuffer.

setLength()

- To set the length of the buffer within a StringBuffer object.
- General form:

```
void setlength(int len)
```

- Here, len specifies the length of the buffer.
- Usage:
- When you increase the length of the buffer, null characters are added to the end of the existing buffer. If you call setLength() with a value less than the current value returned by length(), then the characters stored beyond the new length will be lost.

charAt() and setCharAt()

- To obtain the value of a single character, use CharAt().
- To set the value of a character within StringBuffer, use setCharAt().
- General form:

```
char charAt(int where)
void setCharAt(int where, char ch)
```

- For charAt(), where specifies the index of the characters being obtained.
- For setCharAt(), where specifies the index of the characters being set,and ch specifies the new value of that character.where must be non negative and must not specify a location beyond the endof the buffer.

Example:charAt() and setCharAt()

```
class setCharAtDemo {
public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        System.out.println("buffer before = " + sb);
        System.out.println("charAt(1) before = " +
                                                sb.charAt(1));
        sb.setCharAt(1, 'i');
        sb.setLength(2);
        System.out.println("buffer after = " + sb);
        System.out.println("charAt(1) after = " +
        sb.charAt(1));
    }
}
```

getChars()

- To copy a substring of a StringBuffer into an array.
- General form:

  void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

- Where:
- srcBegin - start copying at this offset.
- srcEnd - stop copying at this offset.
- dst - the array to copy the data into.
- dstBegin - offset into dst.

append()

- Concatenates the string representation of any other type of data to the end of the invoking StringBuffer object.
- General form:
- StringBuffer append(Object obj)
- StringBuffer append(String str)
- StringBuffer append(int num)
- String.valueOf() is called for each parameter to obtain its string representation. The result is appended to the current StringBuffer object.

Example: append()

```
class appendDemo {
    public static void main(String args[])   String s;
            int a = 42;
    StringBuffer sb = new StringBuffer(40);
    s=sb.append("a “).append(a).append("!").toString();
    System.out.println(s);
    }
}
```

insert()

- Inserts one string into another. It is overloaded to accept values of all the simple types, plus String and Objects.
- General form:

```
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)
```

- Here, index specifies the index at which point the String will be inserted into the invoking StringBuffer object.

Example: insert()

```
class insertDemo {
    public static void main(String args[]) {
    StringBuffer sb = new StringBuffer("I Java!");
    sb.insert(2, "like ");
    System.out.println(sb);
    }
}
```

reverse()

- To reverse the character within a StringBuffer object.
- General form:

  StringBuffer reverse()
- This method returns the reversed on which it was called.
- For example:
- ```
  class ReverseDemo {
      public static void main(String args[]) {
      StringBuffer s = new StringBuffer("abcdef");
          System.out.println(s);
          s.reverse();
          System.out.println(s);
      }
  }
  ```
  ```

replace()

- Replaces one set of characters with another set inside a StringBuffer object.
- General form:

```
StringBuffer replace(int startIndex, String
                     endIndex,    String str)
```

- The substring being replaced is specified by the indexes startIndex and endIndex. Thus, the substring at startIndex through endIndex-1 is replaced. The replacement string is passed in str. The resulting StringBuffer object is returned.

Example: replace()

```
class replaceDemo {
public static void main(String args[]) {
      StringBuffer sb = new StringBuffer("This is a
                                                test.");
      sb.replace(5, 7, "was");
      System.out.println("After replace: " + sb);
      }
}
```

substring()

- Returns a portion of a StringBuffer.
- General form:

  String substring(int startIndex)

  String substring(int startIndex, int endIndex)
- The first form returns the substring that starts at startIndex and runs tothe end of the invoking StringBuffer object.
- The second form returns the substring that starts at startIndex and runs through endIndex-1.
- These methods work just like those defined for String that were described earlier.