class String

class String

- Strings are a sequence of characters.
- Strings are objects.

Creating Strings

- Strings are a sequence of characters.
- Strings are objects.
- The most direct way to create a string

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

- "Hello world!" is a string literal that is enclosed in double quotes.
- Another way to create String objects by using the **new** keyword and a constructor.

String Length

- We can use length() method, which returns the number of characters contained in the string object.
- String str = "Hello world!";
- str.length()
- length() method returns the number of characters contained in the str object that is 12.

String is immutable

- The String class is immutable, so that once it is created a String object cannot be changed.
- The String class has a number of methods, that appear to modify strings. Since strings are immutable, what these methods really do is create and return a new string that contains the result of the operation.
- When a modifiable string is desired, Java provides two options: **StringBuffer** and **StringBuilder**.
- Both hold strings that can be modified after they are created.
- Both classes are defined in java.lang.

Concatenating Strings

- The String class includes a method for concatenating two strings:

  string1.concat(string2);

- This returns a new string that is string1 with string2 added to it at the end.

  “Ram ".concat(“Chandar");

- Strings are more commonly concatenated with the + operator

  “Ram" + " Chandar”

- which results in “Ram Chandar”
- The + operator is widely used in print statements.

  String string1 = “Ram";

  System.out.println(“Sita " + string1 + “ Chandar");

Concatenating Strings

- You can concatenate strings with other types of data.

```
String s = "four: " + 2 + 2;
System.out.println(s);
```

- The result of concatenation is “**four: 22”**
- Operator precedence causes the concatenation of "four" with the string equivalent of 2 to take place first. This result is then concatenated with the string equivalent of 2 a second time.
- To complete the integer addition first, you must use parentheses, like this:

```
String s = "four: " + (2 + 2);
```

Now s contains the string “**four: 4”**

Converting Numbers to Strings

- Sometimes we need to convert a number to a string because you need to operate on the value in its string form.
- There are several easy ways to convert a number to a string:
- int i;
- // Concatenate "i" with an empty string; conversion is handled for you.

  String s1 = "" + i;
- The valueOf(primitive types) method.

  String s2 = String.valueOf(i);

Converting Numbers to Strings

- Each of the **Number** subclasses includes a class method, toString(), that will convert its primitive type to a string.

```
int i;
double d;
String s3 = Integer.toString(i);
String s4 = Double.toString(d);
```

- toString() method converts a number to a string.

String Constructors

- The String class supports several constructors.
- You will want to create strings that have initial values. The String class provides a variety of constructors to handle this.
- To create a String initialized by an array of characters, use the constructor shown here:

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

```
char chars[] = { 'a', 'b', 'c' };
```

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

- This constructor initializes s with the string "abc".

String Constructors

- You can specify a subrange of a character array as an initializer using the following constructor:

  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.

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

- This initializes s with the characters cde.

String Constructors

- You can construct a String object that contains the same character sequence as another
- String object using this constructor:
- String(String strObj)

```
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 Constructors

- You can construct a String object that contains the same character sequence as another
- String object using this constructor:
- String(String strObj)

```
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);
    }                          Output: Java
}                                      Java
```

String Constructors

- We can construct a String from a **StringBuffer** by using the constructor shown here:

  String(StringBuffer strBufObj)

- We can construct a String from a **StringBuilder** by using this constructor:
- String(StringBuilder strBuildObj)

Getting Characters and Substrings by Index

- We can get more than one consecutive character from a given string.
- We can use the substring() method.

  String substring(int beginIndex, int endIndex)

- Returns a new string that is a substring of this string. The substring begins at the specified *beginIndex* and extends to the character at index *endIndex* - 1.

  String substring(int beginIndex)

- Returns a new string that is a substring of this string. The integer argument specifies the index of the first character. Here, the returned substring extends to the end of the original string.

Methods for Manipulating Strings

- We can use the split() method.

  String[] split(String regex)

  String[] split(String regex, int limit)

- Searches for a match as specified by the string argument *regex* and splits this string into an array of strings accordingly.
- The integer argument specifies the maximum size of the returned array.

  String trim()

- Returns a copy of this string with leading and trailing white space removed.

Methods for Manipulating Strings

String toLowerCase()

String toUpperCase()

- Returns a copy of this string converted to lowercase or uppercase.

Searching for Characters and Substrings

- String class has some methods for finding characters or substrings within a string.

  int indexOf(int ch)

  int lastIndexOf(int ch)

- Returns the index of the first (last) occurrence of the specified character.

  int indexOf(int ch, int fromIndex)

  int lastIndexOf(int ch, int fromIndex)

- Returns the index of the first (last) occurrence of the specified character, searching forward from the specified index.

Searching for Characters and Substrings

int indexOf(String str)

int lastIndexOf(String str)

- Returns the index of the first (last) occurrence of the specified substring.

int indexOf(String str, int fromIndex)

int lastIndexOf(String str, int fromIndex)

- Returns the index of the first (last) occurrence of the specified substring, searching forward from the specified index.

boolean contains(CharSequence s)

- Returns true if the string contains the specified character sequence.

Replacing Characters and Substrings

String replace(char oldChar, char newChar)

- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

String replace(CharSeqence target,
CharSequence replacement)

- Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.

String replaceAll(String regex,
String replacement)

- Replaces each substring of this string that matches the given regular expression with the given replacement.

Comparing Strings and Portions of Strings

boolean endsWith(String suffix)

boolean startsWith(String prefix)

- Returns true if this string ends with or begins with the substring specified as an argument to the method.

boolean startsWith(String prefix, int offset)

- Considers the string beginning at the index offset, and returns true if it begins with the substring specified as an argument.

Comparing Strings and Portions of Strings

int compareTo(String anotherString)

- Compares two strings lexicographically. Returns an integer indicating whether this string is greater than (result is > 0), equal to (result is = 0), or less than (result is < 0) the argument.

int compareToIgnoreCase(String str)

- Compares two strings lexicographically, ignoring differences in case. Returns an integer indicating whether this string is greater than (result is > 0), equal to (result is = 0), or less than (result is < 0) the argument.

Comparing Strings and Portions of Strings

boolean equals(Object anObject)

- Returns true if and only if the argument is a String object that represents the same sequence of characters as this object.

boolean equalsIgnoreCase(String anotherString)

- Returns true if and only if the argument is a String object that represents the same sequence of characters as this object, ignoring differences in case.