Chapter - 10

String, StringBuffer and StringBuilder classes

String, StringBuffer and StringBuilder classes

Strings

In Java a string can be defined as a sequence of characters. Strings can be stored in Java using **String** class objects. The **String** class object can store a string of any length. **String** objects are **immutable** objects i.e. constant objects. A String object once created is initialized with characters and whose string contents can not be modified in the same object, but modification to the contents of String object results to a new String object. Modification to the contents in the same object performance is slower than constructing the new object, that is why, any modification to the strings results to new object. The older objects are automatically collected by garbage collector. Stings can be stored into string objects which are defined to String class of **java.lang** package. The String class contains various methods that perform string operations such as finding the length of string, converting lower case to upper case and vice versa, etc.

The constructors and methods present in the **String class** are listed in the below tables.

Constructors	Meaning
String()	Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes)	Construct a new String by converting the specified array of bytes using the platform's default character encoding.
String(byte[] bytes, int offset, int length)	Construct a new String by converting the specified subarray of bytes using the platform's default character encoding.
String(char[] value)	Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
String(char[] value, int offset, int count)	Allocates a new String that contains characters from a subarray of the character array argument.

String(String original)	Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
String(StringBuffer buffer)	Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

Methods	**Meaning**
char **charAt**(int index)	Returns the character at the specified index.
int **compareTo**(Object o)	Compares this String to another Object.
int **compareTo**(String anotherString)	Compares two strings lexicographically.
int **compareToIgnoreCase**(String str)	Compares two strings lexicographically, ignoring case considerations.
String **concat**(String str)	Concatenates the specified string to the end of this string.
static String **copyValueOf**(char[] data)	Returns a String that is equivalent to the specified character array.
static String **copyValueOf**(char[] data, int offset, int count)	Returns a String that is equivalent to the specified character array.
boolean **endsWith**(String suffix)	Tests if this string ends with the specified suffix.
boolean **equals**(Object anObject)	Compares this string to the specified object.
boolean **equalsIgnoreCase**(String anotherString)	Compares this String to another String, ignoring case considerations.
byte[] **getBytes**()	Convert this String into bytes according to the platform's default character encoding, storing the result into a new byte array.

void **getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin)	Copies characters from this string into the destination character array.
int **hashCode**()	Returns a hashcode for this string.
int **indexOf**(int ch)	Returns the index within this string of the first occurrence of the specified character.
int **indexOf**(int ch, int fromIndex)	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)	Returns the index within this string of the first occurrence of the specified substring.
int **indexOf**(String str, int fromIndex)	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
int **lastIndexOf**(int ch)	Returns the index within this string of the last occurrence of the specified character.

A program on constructing string using constructors

Bin>edit string1.java

```
class string1
{
   public static void main(String argv[])
   {
         String s1=new String();
         char name[]={'k','u','m','a','r'};
         byte b[]={65,66,67,68,69,70};
         String s2=new String(name);
         String s3=new String(name,2,3);
         String s4=new String(s2);
         String s5=new String(b);
         String s6=new String(b,2,3);
         String s7="Apex";  // initializing string literals
         System.out.print("\ns1="+s1+"\ns2="+s2+"\ns3="+
          s3+"\ns4="+s4+"\ns5="+s5+"\ns6="+s6+"\ns7="+s7);
   }
}
```

Bin>javac string1.java
Bin>java string1

```
s1=
s2=kumar
s3=mar
s4=kumar
s5=ABCDEF
S6=CDE
S7=Apex
```

A program to demonstrate some methods of String class

Bin>edit string2.java

```
class string2
{
   public static void main(String argv[])
   {
        String s1="Apex Computers";
        String s2,s3;
        s2=s1;
        s1=s1.toUpperCase();
        s3=s1.substring(3,6);
        int len=s1.length();
        char ch=s1.charAt(3);
        System.out.print("\n s1="+s1+"\ns2="+s2+
             "\ns3="+s3+"\nlen="+len+"\nch="+ch);

   }
}
```

Bin>javac string2.java

Bin>java string2

```
s1=APEX COMPUTERS
s2=Apex Computers
s3=X C
len=14
ch=X
```

Another program to demonstrate some String class methods

Bin>edit string3.java

```
class string3
{
   public static void main(String argv[])
   {
        String s1="apex";
        String s2="APEX";
```

```
        String s3="   apex   ";
        System.out.print("\n s1.equals(s2)="+
                         s1.equals(s2));
        System.out.print("\n s1.equalsIgnoreCase(s2)="+
                         s1. equalsIgnoreCase (s2));

        s1=s1.replace('p','1');
        System.out.print("\n after replacement s1 = " +s1)
        int n=s2.indexOf("PE")
        System.out.print("\n n= "+n);
        s3=s2.trim();
        System.out.print("\ns3= "+s3);

    }
}
```

Bin>javac string3.java
Bin>java string3

s1.equals(s2) =false
s1.equalsIgnoreCase(s2) =true
after replacement s1 = a1ex
n=1
s3=apex

Note: Other methods you try on your own.

StringBuffer class

The **String** class objects are immutable objects whose contents cannot be modified in the same object. If we want to modify the contents in the same object then we can go with **StringBuffer** class. The objects of **StringBuffer** class are mutable objects i.e. these object contents can be modified in the same object. The **StringBuffer** object is allocated with extra **16** free elements of characters so that the size of the **StringBuffer** can increase wherever needed. This helps the **StringBuffer** object to grow by 16 more characters without any other process. If the string grows beyond the free **16** character space, the **StringBuffer** is relocated to a new memory space with the required size. So, memory management for handling **StringBuffer** will not be efficient as that of the fixed-length string objects.

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.

The principal operations on a **StringBuffer** are the append and insert methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string buffer. The append method always adds these characters at the end of the buffer; the insert method adds the characters at a specified point.

For example, if z refers to a string buffer object whose current contents are "**start**", then the method call **z.append("le")** would cause the string buffer to contain "**startle**", whereas **z.insert(4, "le")** would alter the string buffer to contain "**starlet**".

In general, if sb refers to an instance of a **StringBuffer**, then **sb.append(x)** has the same effect as **sb.insert(sb.length(), x)**.

Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger.

The constructors and methods present in the **StringBuffer** class are listed in the below tables.

Constructors	Meaning
StringBuffer()	Constructs a string buffer with no characters in it and an initial capacity of 16 characters.
StringBuffer(int length)	Constructs a string buffer with no characters in it and an initial capacity specified by the length argument.
StringBuffer(String argument)	Constructs a string buffer so that it represents the same sequence of characters as the string argument; plus space for 16 more characters.

Methods	Meaning
StringBuffer **append**(boolean b)	Appends the string representation of the boolean argument to the string buffer.
StringBuffer **append**(char c)	Appends the string representation of the char argument to this string buffer.
StringBuffer **append**(char[] str)	Appends the string representation of the char array argument to this string buffer.
StringBuffer **append**(char[] str, int offset, int len)	Appends the string representation of a subarray of the char array argument to this string buffer.
StringBuffer **append**(double d)	Appends the string representation of the double argument to this string buffer.
StringBuffer **append**(float f)	Appends the string representation of the float argument to this string buffer.
StringBuffer **append**(int i)	Appends the string representation of the int argument to this string buffer.
StringBuffer **append**(long l)	Appends the string representation of the long argument to this string buffer.
StringBuffer **append**(Object obj)	Appends the string representation of the Object argument to this string buffer.
StringBuffer **append**(String str)	Appends the string to this string buffer.
StringBuffer **append**(StringBuffer sb)	Appends the specified StringBuffer to this StringBuffer.
int **capacity**()	Returns the current capacity of the String buffer.
char **charAt**(int index)	The specified character of the sequence currently represented by the string buffer, as indicated by the index argument, is returned.
StringBuffer **delete**(int start, int end)	Removes the characters in a substring of this StringBuffer.

StringBuffer **deleteCharAt**(int index)	Removes the character at the specified position in this StringBuffer (shortening the StringBuffer by one character).
Void **ensureCapacity**(int minimu mCapacity)	Ensures that the capacity of the buffer is at least equal to the specified minimum.
void **getChars**(int srcBegin, int srcEnd, char[] dst,	Characters are copied from this string buffer into the destination character array dst.
int **indexOf**(String str)	Returns the index within this string of the first occurrence of the specified substring.
int **indexOf**(String str, int fromIndex)	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
StringBuffer **insert**(int offset, boolean b)	Inserts the string representation of the boolean argument into this string buffer.
StringBuffer **insert**(int offset, char c)	Inserts the string representation of the char argument into this string buffer.
StringBuffer **insert**(int offset, char[] str)	Inserts the string representation of the char array argument into this string buffer.
StringBuffer **insert**(int index, char[] str, int offset, int len)	Inserts the string representation of a subarray of the str array argument into this string buffer.
int **lastIndexOf**(String str)	Returns the index within this string of the rightmost occurrence of the specified substring.
int **lastIndexOf**(String str, int fromIndex)	Returns the index within this string of the last occurrence of the specified substring.
int **length**()	Returns the length (character count) of this string buffer.
StringBuffer **replace**(int start, int end, String str)	Replaces the characters in a substring of this StringBuffer with characters in the specified String.

StringBuffer **reverse**()	The character sequence contained in this string buffer is replaced by the reverse of the sequence.
void **setCharAt**(int index, char ch)	The character at the specified index of this string buffer is set to ch.
Void **setLength**(int newLength)	Sets the length of this String buffer.
String **substring**(int start)	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.
String **substring**(int start, int end)	Returns a new String that contains a subsequence of characters currently contained in this StringBuffer.
String **toString**()	Converts to a string representing the data in

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.

The principal operations on a **StringBuffer** are the append and insert methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string buffer. The append method always adds these characters at the end of the buffer; the insert method adds the characters at a specified point.

For example, if z refers to a string buffer object whose current contents are "**start**", then the method call **z.append("le")** would cause the string buffer to contain "**startle**", whereas **z.insert(4, "le")** would alter the string buffer to contain "**starlet**".

In general, if **sb** refers to an instance of a **StringBuffer**, then **sb.append(x)** has the same effect as **sb.insert(sb.length(), x)**.
Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger.

A program to demonstrate constructors, capacity(), length(), ensureCapacity() and setLength() methods of StringBuffer class.

Bin>edit string4.java

```
class string4
{
   public static void main(String argv[])
   {
         StringBuffer s1=new StringBuffer();
         StringBuffer s2=new StringBuffer(30);
         StringBuffer s3=new StringBuffer("apex");
         System.out.print("\n s1.length="+
         s1.length()+"\ts1.capacity="+s1.capacity());
         System.out.print("\ns2.length="+s2.length()+
           "\ts2.capacity=" +s2.capacity());
         System.out.print("\ns3.length="+s3.length()+
          "\ts3.capacity=" +s3.capacity());
         s1.ensureCapacity(70);
         System.out.print("\nafter ensurecapacity
         s1.capacity="+s1.capacity());
         System.out.print("\n="+s3);
         s3.setLength(2);
         System.out.print("\n s3 after setLength() ="+s3);
   }
}
```

Bin>javac string4.java
Bin>java string4

```
s1.length=0  s1.capacity=16
s2.length=0  s2.capacity=30
s3.length=4  s3.capacity=20
after ensurecapacity s1.capacity=70
```

s3=apex
s3 after setLength() =ap

A program demonstrate append(), insert(),reverse() methods of StringBuffer Class.

Bin> edit string5.java

```
class string5
{
   public static void main(String argv[])
   {
         StringBuffer s1=new StringBuffer("apex");
         StringBuffer s2=new StringBuffer("computers");
         s1.append(" warangal");
         s1.insert(5,"computers");
         s2.reverse();
         System.out.print("\n s1="+s1+"\ns2="+s2);
   }
}
```

Bin>javac string5.java
Bin>java string5

s1=apex computerswarangal
s2=sretupmoc

Note: Other methods you try on your own.

Differences between String class and StringBuffer class.

StringBuilder

A **mutable** sequence of characters. This class provides an **API** compatible with **StringBuffer**, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for **StringBuffer** in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to **StringBuffer** as it will be faster under most implementations.

The principal operations on a **StringBuilder** are the append and insert methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string builder. The append method always adds these characters at the end of the builder; the insert method adds the characters at a specified point.

For example, if z refers to a string builder object whose current contents are "**start**", then the method call **z.append("le")** would cause the string builder to contain "**startle**", whereas **z.insert(4, "le")** would alter the string builder to contain "starlet".

In general, if **sb** refers to an instance of a **StringBuilder**, then **sb.append(x)** has the same effect as **sb.insert(sb.length(), x)**. Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger.

Instances of **StringBuilder** are not safe for use by multiple threads. If such synchronization is required then it is recommended that **StringBuffer** be used.

Note:

1. Constructors and Method of **StringBuilder** class are similar to **StringBuffer** class.
2. Execute String buffer programs replacing **StringBuffer** class name with **StringBuilder** name.