JAVA PROGRAMMING

COURSE NAME: MCA – 5TH SEMESTER

COURSE CODE: MCA18501CR

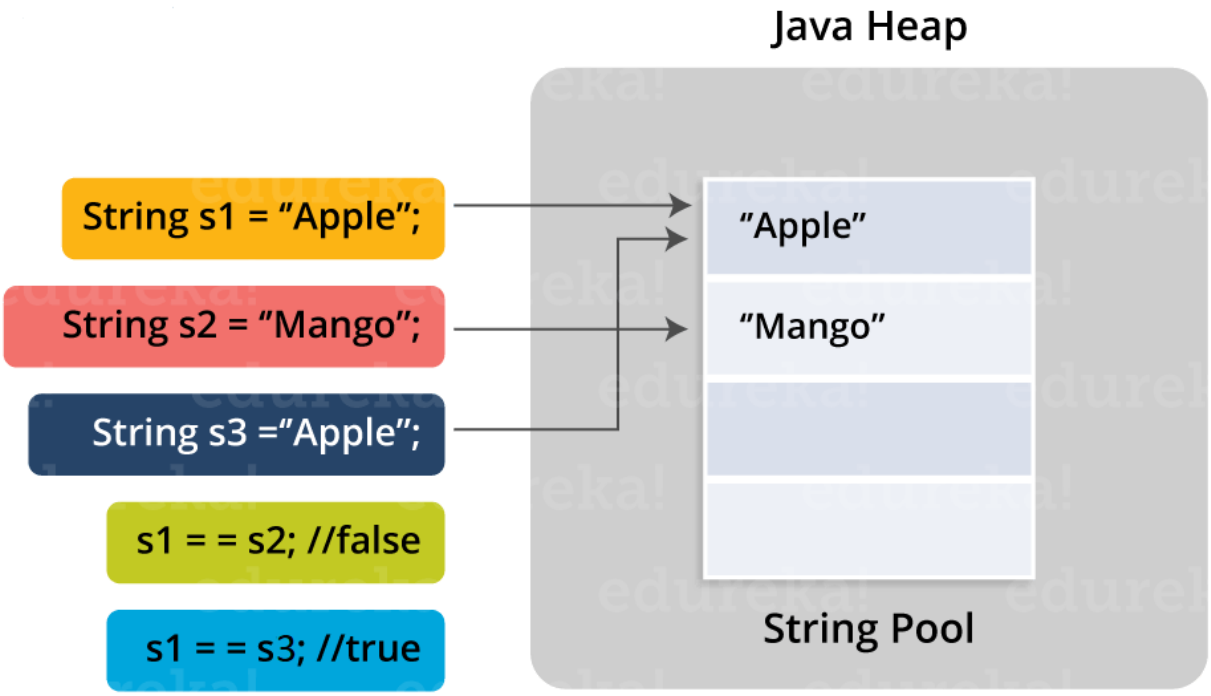
Teacher Incharge: *Dr. Shifaa Basharat*
Contact: *fazilishifaa@gmail.com*

STRINGS

Strings in Java are Objects that are backed internally by a char array. Since arrays are immutable (cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created. There are two ways to create a String object:

1. **By string literal**: Java String literal is created by using double quotes. For Example: String s="Welcome";
2. **By new keyword**: Java String is created by using a keyword "new". For example: String s=new String ("Welcome");
   It creates two objects (in String pool and in heap) and one reference variable where the variable 's' will refer to the object in the heap.

Java String Pool: Java String pool refers to collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned as depicted below:



In the above figure, two Strings are created using literal i.e "Apple" and "Mango". Now, when third String is created with the value "Apple", instead of creating a new object, the already present object reference is returned.

String Constructors

1. String() : Initializes a newly created String object so that it represents an empty character sequence.
2. String(byte[] bytes): Constructs a new String by decoding the specified array of bytes using the platform's default charset.
3. String(byte[] bytes, Charset charset) : Constructs a new String by decoding the specified array of bytes using the specified charset.
4. String(byte[] bytes, int offset, int length) : Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.
5. String(byte[] bytes, int offset, int length, Charset charset) : Constructs a new String by decoding the specified subarray of bytes using the specified charset.
6. String(byte[] bytes, int offset, int length, String charsetName) : Constructs a new String by decoding the specified subarray of bytes using the specified charset.
7. String(byte[] bytes, String charsetName) : Constructs a new String by decoding the specified array of bytes using the specified charset.
8. String(char[] value) : Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
9. String(char[] value, int offset, int count) : Allocates a new String that contains characters from a subarray of the character array argument.
10. String(int[] codePoints, int offset, int count) : Allocates a new String that contains characters from a subarray of the Unicode code point array argument.
11. 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.
12. String(StringBuffer buffer) : Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.
13. String(StringBuilder builder) : Allocates a new string that contains the sequence of characters currently contained in the string builder argument.

String Methods

Modifier and type	Method Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex)

	Returns the number of Unicode code points in the specified text range of this `String`.
int	**compareTo**(**String** anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(**String** str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(**String** str) Concatenates the specified string to the end of this string.
boolean	**contains**(**CharSequence** s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(**CharSequence** cs) Compares this string to the specified `CharSequence`.
boolean	**contentEquals**(**StringBuffer** sb) Compares this string to the specified `StringBuffer`.
static String	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static String	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
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.
static String	**format**(**Locale** l, **String** format, **Object**... args) Returns a formatted string using the specified locale, format string, and arguments.
static String	**format**(**String** format, **Object**... args)

	Returns a formatted string using the specified format string and arguments.
bytes[]	**getBytes**() Encodes this `String` into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
byte[]	**getBytes**(**Charset** `charset`) Encodes this `String` into a sequence of bytes using the given **charset**, storing the result into a new byte array.
byte[]	**getBytes**(**String** `charsetName`) 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`) Copies characters from this string into the destination character array.
int	**hashCode**() Returns a hash code 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.
String	**intern**() Returns a canonical representation for the string object.
boolean	**isEmpty**() Returns `true` if, and only if, **length()** is `0`.
int	**lastIndexOf**(`int ch`) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(`int ch, int fromIndex`)

	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`) Returns the index within this string of the last 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, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(**String** `regex`) Tells whether or not this string matches the given **regular expression**.
int	**offsetByCodePoints**(`int index, int codePointOffset`) Returns the index within this `String` that is offset from the given `index` by `codePointOffset` code points.
boolean	**regionMatches**(`boolean ignoreCase, int toffset,` **String** `other, int ooffset,int len`) Tests if two string regions are equal.
boolean	**regionMatches**(`int toffset,` **String** `other, int ooffset, int len`) Tests if two string regions are equal.
String	**replace**(`char oldChar, char newChar`) Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.
String	**replace**(**CharSequence** `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.
String	**replaceFirst**(**String** `regex,` **String** `replacement`) Replaces the first substring of this string that matches the given **regular expression** with the given replacement.
String[]	**split**(**String** `regex`) Splits this string around matches of the given **regular expression**.
String[]	**split**(**String** `regex, int limit`)

	Splits this string around matches of the given **regular expression**.
boolean	**startsWith**(**String** `prefix`) Tests if this string starts with the specified prefix.
boolean	**startsWith**(**String** `prefix, int toffset`) Tests if the substring of this string beginning at the specified index starts with the specified prefix.
CharSequene	**subSequence**(`int beginIndex, int endIndex`) Returns a new character sequence that is a subsequence of this sequence
String	**substring**(`int beginIndex`) Returns a new string that is a substring of this string.
String	**substring**(`int beginIndex, int endIndex`) Returns a new string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this `String` to lower case using the rules of the default locale.
String	**toLowerCase**(**Locale** `locale`) Converts all of the characters in this `String` to lower case using the rules of the given `Locale`.
String	**toString**() This object (which is already a string!) is itself returned.
String	**toUpperCase**() Converts all of the characters in this `String` to upper case using the rules of the default locale.
String	**toUpperCase**(**Locale** `locale`) Converts all of the characters in this `String` to upper case using the rules of the given `Locale`.
String	**trim**() Returns a copy of the string, with leading and trailing whitespace omitted.
static String	**valueOf**(`boolean b`) Returns the string representation of the `boolean` argument.
static String	**valueOf**(`char c`)

	Returns the string representation of the `char` argument.
static String	**valueOf**(`char[] data`) Returns the string representation of the `char` array argument.
static String	**valueOf**(`char[] data, int offset, int count`) Returns the string representation of a specific subarray of the `char` array argument.
static String	**valueOf**(`double d`) Returns the string representation of the `double` argument.
static String	**valueOf**(`float f`) Returns the string representation of the `float` argument.
static String	**valueOf**(`int i`) Returns the string representation of the `int` argument.
static String	**valueOf**(`long l`) Returns the string representation of the `long` argument.
static String	**valueOf**(**Object** `obj`) Returns the string representation of the `Object` argument.

Creating formatted strings:

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

```
System.out.printf("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);
```

we can use format() as shown in the program below:

```
class Demo
{
public static void main(String args[])
{
String fs;
double floatVar=3.4;
int intVar=4;
String stringVar="Hello";
```

```
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);
}
}
```

StringBuffer in Java

StringBuffer is a peer class of **String** that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable (mutable) and writable character sequences.

StringBuffer Constructors :

1. public StringBuffer()
   Constructs a string buffer with no characters in it and an initial capacity of 16 characters.

2. public StringBuffer(int capacity)
   Constructs a string buffer with no characters in it and the specified initial capacity.

   Throws:

   NegativeArraySizeException - if the capacity argument is less than 0.

3. public StringBuffer(String str)
   Constructs a string buffer initialized to the contents of the specified string. The initial capacity of the string buffer is 16 plus the length of the string argument.

   Parameters:

   str - the initial contents of the buffer.

   Throws:

   NullPointerException - if str is null

4. public StringBuffer(CharSequence seq)
   Constructs a string buffer that contains the same characters as the specified CharSequence. The initial capacity of the string buffer is 16 plus the length of the CharSequence argument. If the length of the specified CharSequence is less than or equal to zero, then an empty buffer of capacity 16 is returned.
   Parameters:
   seq - the sequence to copy.

   Throws:
   NullPointerException - if seq is null

StringBuffer Methods:

length( ) and capacity( ): The length of a StringBuffer can be found by the length( ) method, while the total allocated capacity can be found by the capacity( ) method. The general form is:

- public int length() :Returns the length of the sequence of characters currently represented by this object
- public int capacity() : Returns the current capacity. The capacity is the amount of storage available for newly inserted characters, beyond which an allocation will occur.

Example:

```
import java.io.*;
class Demo {
   public static void main(String[] args)
   {
      StringBuffer s = new StringBuffer("HelloWorld");
      int p = s.length();
      int q = s.capacity();
      System.out.println("Length of string =" + p);
      System.out.println("Capacity of string =" + q);
   }
}
//Output : Length of string =10
           Capacity of string=26
```

append( ): It is used to add text at the end of the existing text. The general forms include:

- StringBuffer append(String str) : The characters of the String argument are appended, in order, increasing the length of this sequence by the length of the argument.
- StringBuffer append(int num) : Appends the string representation of the int argument to this sequence.

Example:

```
import java.io.*;
class Demo {
   public static void main(String[] args)
   {
      StringBuffer s = new StringBuffer("Hello");
      s.append("World");
      System.out.println(s); // returns HelloWorld
      s.append(1);
      System.out.println(s); // returns HelloWorld1
   }
}
```

Output:

HelloWorld
HelloWorld1

insert( ): It is used to insert text at the specified index position. These general forms include:

- public StringBuffer insert(int offset, String str) : Inserts the string into this (already existing) character sequence.
  The characters of the String argument are inserted, in order, into this sequence at the indicated offset, moving up any characters originally above that position and increasing the length of this sequence by the length of the argument.

- public StringBuffer insert(int offset, char c) : Inserts the string representation of the char argument into this sequence.
  The overall effect is exactly as if the second argument were converted to a string by the method String.valueOf(char), and the character in that string were then inserted into this character sequence at the indicated offset.
  The offset argument must be greater than or equal to 0, and less than or equal to the length of this sequence.

- public StringBuffer insert(int offset, Object obj) : Inserts the string representation of the Object argument into this character sequence.
  The overall effect is exactly as if the second argument were converted to a string by the method String.valueOf(Object), and the characters of that string were then inserted into this character sequence at the indicated offset.
  The offset argument must be greater than or equal to 0, and less than or equal to the length of this sequence.

Example:

```
import java.io.*;
class Demo {
   public static void main(String[] args)
   {
      StringBuffer s = new StringBuffer("Hello");
      s.insert(5, "world");
      System.out.println(s);
      s.insert(0, 5);
      System.out.println(s);
      s.insert(3, true);
      System.out.println(s);
      s.insert(5, 41.35d);
      System.out.println(s);
```

```
        s.insert(8, 41.35f);
        System.out.println(s);
        char arr[] = { 'a', 'b', 'c', 'd', ‘e' };

        // insert character array at offset 9
        s.insert(2, arr);
        System.out.println(s);    }
}
```

Output:

```
Helloworld
5Helloworld
5Hetruelloworld
5Hetr41.35uelloworld
5Getr41.41.3535uelloworld
5Habcdeetr41.41.3535uelloworld
```

reverse( ): It can reverse the characters within a StringBuffer object using **reverse( ).**This method returns the reversed object on which it was called. The general form is:

- public StringBuffer reverse() :Causes this character sequence to be replaced by the reverse of the sequence.

Example:

```
import java.io.*;
class Demo {
   public static void main(String[] args)
   {
      StringBuffer s = new StringBuffer("Helloworld");
      s.reverse();
      System.out.println(s);
  }
}
```

Output:

```
dlrowolleH
```

delete( ) and deleteCharAt( ): We can delete characters within a StringBuffer by using the methods **delete( )** and **deleteCharAt( )**.The general forms include:

- public StringBuffer delete(int start, int end) :Removes the characters in a substring of this sequence. The substring begins at the specified start and extends to the character at

index end - 1 or to the end of the sequence if no such character exists. If start is equal to end, no changes are made.

- public StringBuffer deleteCharAt(int index) : Removes the char at the specified position in this sequence. This sequence is shortened by one char.

Example:

```
import java.io.*;
class Demo {
   public static void main(String[] args)
   {
      StringBuffer s = new StringBuffer("HellothereWorld");
      s.delete(0, 5);
      System.out.println(s);
      s.deleteCharAt(7);
      System.out.println(s);
   }
}
```

Output:

thereWorld

thereWold

replace( ): It can replace one set of characters with another set inside a StringBuffer object. The substring being replaced is specified by the indexes start Index and endIndex. The general form is:

- public StringBuffer replace(int start, int end, String str) :Replaces the characters in a substring of this sequence with characters in the specified String. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists. First the characters in the substring are removed and then the specified String is inserted at start. (This sequence will be lengthened to accommodate the specified String if necessary.)

Example:

```
import java.io.*;
class Demo{
   public static void main(String[] args)
   {
      StringBuffer s = new StringBuffer("Hello how for you");
      s.replace(10, 13, "are");
      System.out.println(s);
   }
}
```

Output:
Hello how are you

ensureCapacity( ): It is used to increase the capacity of a StringBuffer object. The general form is:

- public void ensureCapacity(int minimumCapacity) : Ensures that the capacity is at least equal to the specified minimum. If the current capacity is less than the argument, then a new internal array is allocated with greater capacity. The new capacity is the larger of:
  - → The minimumCapacity argument.
  - → Twice the old capacity, plus 2.

  If the minimumCapacity argument is nonpositive, this method takes no action and simply returns.

setLength(): This method sets the length of the character sequence. The general form is:

- public void setLength(int newLength) :The sequence is changed to a new character sequence whose length is specified by the argument. For every nonnegative index *k* less than newLength, the character at index *k* in the new character sequence is the same as the character at index *k* in the old sequence if *k* is less than the length of the old character sequence; otherwise, it is the null character '\u0000'. In other words, if the newLength argument is less than the current length, the length is changed to the specified length.
  If the newLength argument is greater than or equal to the current length, sufficient null characters ('\u0000') are appended so that length becomes the newLength argument.
  The newLength argument must be greater than or equal to 0.

substring(): A portion of StringBuffer can be obtained by calling substring() method. Its general form is:

- public String substring(int start) :Returns a new String that contains a subsequence of characters currently contained in this character sequence. The substring begins at the specified index and extends to the end of this sequence.
- public String substring(int start, int end) : Returns a new String that contains a subsequence of characters currently contained in this sequence. The substring begins at the specified start and extends to the character at index end - 1.

getchars(): We can copy a substring of a StringBuffer into a character array using this method. The general form is:

- public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) : Characters are copied from this sequence into the destination character array dst. The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1. The total number of characters to be copied is srcEnd-srcBegin. The characters are copied into the subarray of dst starting at index dstBegin and ending at index:

  dstbegin + (srcEnd-srcBegin) - 1

charAt() and setcharAt(): The value of a single character can be obtained from a StringBuffer via charAt() method. We can set the value of a character within a StringBuffer using setCharAt(). The general form is:

- public char charAt(int index) : Returns the char value in this sequence at the specified index. The first char value is at index 0, the next at index 1, and so on, as in array indexing. The index argument must be greater than or equal to 0, and less than the length of this sequence.

- public void setCharAt(int index, char ch) :The character at the specified index is set to ch. This sequence is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character ch at position index.
  The index argument must be greater than or equal to 0, and less than the length of this sequence.

Example:

```
class Demo{
   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));

   }
}
```

Output:
Buffer before=Hello
charAt(1) before= e
Buffer after=Hi
charAt(1) after= i

Modifier and type	Method and Description
StringBuffer	appendCodePoint(int codePoint) Appends the string representation of the codePoint argument to this sequence.
int	codePointAt(int index) Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index) Returns the character (Unicode code point) before the specified index.

int	codePointCount(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this sequence.
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(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	offsetByCodePoints(int index, int codePointOffset) Returns the index within this sequence that is offset from the given index by codePointOffset code points.
CharSequence	subSequence(int start, int end) Returns a new character sequence that is a subsequence of this sequence.
void	trimToSize() Attempts to reduce storage used for the character sequence.

StringBuilder in Java

The StringBuilder in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternative to String Class, as it creates a mutable sequence of characters. The function of StringBuilder is very much similar to the StringBuffer class, as both of them provide an alternative to String Class by making a mutable sequence of characters. However the StringBuilder class differs from the StringBuffer class on the basis of synchronization. The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does. Therefore this class is designed for use as a drop-in replacement for StringBuffer in places where the StringBuffer 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. Instances of StringBuilder are not safe for use by multiple threads. If such synchronization is required then it is recommended that StringBuffer be used.

StringBuilder Constructors

- **StringBuilder():** Constructs a string builder with no characters in it and an initial capacity of 16 characters.
- **StringBuilder(int capacity):** Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.

- **StringBuilder(CharSequence seq):** Constructs a string builder that contains the same characters as the specified CharSequence.
- **StringBuilder(String str):** Constructs a string builder initialized to the contents of the specified string.

StringBuilder Methods:

1. *StringBuilder appendCodePoint(int codePoint):* This method appends the string representation of the codePoint argument to this sequence.
2. *int capacity():* This method returns the current capacity.
3. *char charAt(int index):* This method returns the char value in this sequence at the specified index.
4. *int codePointAt(int index):* This method returns the character (Unicode code point) at the specified index.
5. *int codePointBefore(int index):* This method returns the character (Unicode code point) before the specified index.
6. *int codePointCount(int beginIndex, int endIndex):* This method returns the number of Unicode code points in the specified text range of this sequence.
7. *StringBuilder delete(int start, int end):* This method removes the characters in a substring of this sequence.
8. *StringBuilder deleteCharAt(int index):* This method removes the char at the specified position in this sequence.
9. *void ensureCapacity(int minimumCapacity):* This method ensures that the capacity is at least equal to the specified minimum.
10. *void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin):* This method characters are copied from this sequence into the destination character array dst.
11. *int indexOf():* This method returns the index within this string of the first occurrence of the specified substring.
12. *StringBuilder insert(int offset, boolean b):* This method inserts the string representation of the booalternatelean argument into this sequence.
13. *int lastIndexOf():* This method returns the index within this string of the last occurrence of the specified substring.
14. *int length():* This method returns the length (character count).
15. *int offsetByCodePoints(int index, int codePointOffset):* This method returns the index within this sequence that is offset from the given index by codePointOffset code points.
16. *StringBuilder replace(int start, int end, String str):* This method replaces the characters in a substring of this sequence with characters in the specified String.

17. *StringBuilder reverse():* This method causes this character sequence to be replaced by the reverse of the sequence.
18. *void setCharAt(int index, char ch):* In this method, the character at the specified index is set to ch.
19. *void setLength(int newLength):* This method sets the length of the character sequence.
20. *CharSequence subSequence(int start, int end):* This method returns a new character sequence that is a subsequence of this sequence.
21. *String toString():* This method returns a string representing the data in this sequence.
22. *void trimToSize():* This method attempts to reduce storage used for the character sequence.