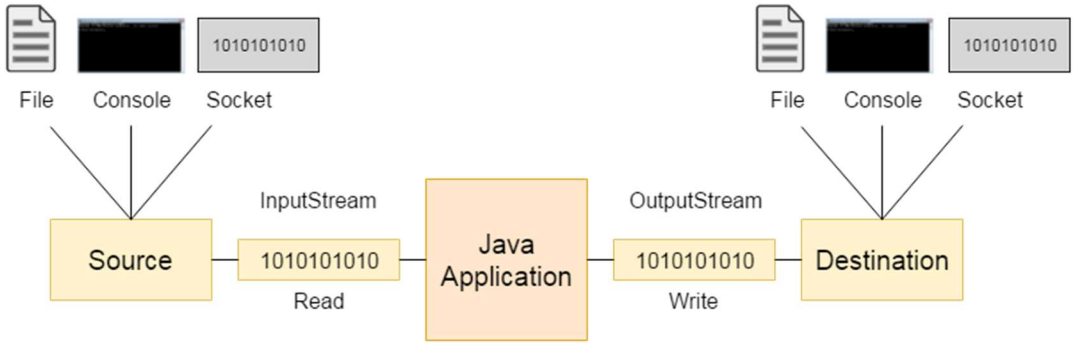
UNIT IV I/O, GENERICS, STRING HANDLING

I/O Basics – Reading and Writing Console I/O – Reading and Writing Files. Generics: Generic Programming – Generic classes – Generic Methods – Bounded Types – Restrictions and Limitations. Strings: Basic String class, methods ,String Buffer Class & StringBuilder class.

INPUT/OUTPUT BASICS

Java I/O (Input and Output) is used to process the input and produce the output. Java uses the concept of stream to make I/O operation fast. All the classes required for input and output operations are declared in java.io package.

A stream can be defined as a sequence of data. The Input Stream is used to read data from a source and the OutputStream is used for writing data to a destination.

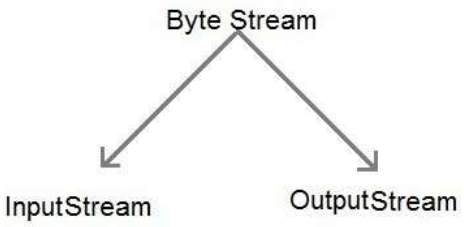


Java defines two types of streams. They are,

1. ***Byte Stream:*** It handles input and output of 8 bit bytes. The frequently used classes are FileInputStream and FileOutputStream.
2. ***Character Stream:*** It is used for handling input and output of characters. Character stream uses 16-bit Unicode. The frequently used classes are FileReader and File Writer.

Byte Stream Classes

The byte stream classes are topped by two abstract classes, InputStream and OutputStream.



InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

- The InputStrearn class is the superclass for all byte-oriented input stream classes.
- All the methods of this class throw an IOException.
- Being an abstract class, the InputStrearn class cannot be instantiated; hence, itssubclasses are used

Some of the Input Stream classes are listed below

Class	Description
Buffered Input Stream	Contains methods to read bytes from the buffer (memory area)
Byte Array Input Stream	Contains methods to read bytes from a byte array
Data Input Stream	Contains methods to read Java primitive data types
File Input Stream	Contains methods to read bytes from a file
Filter Input Stream	Contains methods to read bytes from other input streams which it uses as its basic source of data
Object Input Stream	Contains methods to read objects
Piped Input Stream	Contains methods to read from a piped output stream. A piped input stream must be connected to a piped output stream
Sequence Input Stream	Contains methods to concatenate multiple input streams and then read from the combined stream

Some of the useful methods of InputStream are listed below.

Method	Description
public abstract int read() throws IOException	Reads the next byte of data from the input stream. It returns -1 at the end of file.
public int available() throws IOException	Returns an estimate of the number of bytes that can be read from the current input stream.
public void close() throws IOException	Close the current input stream

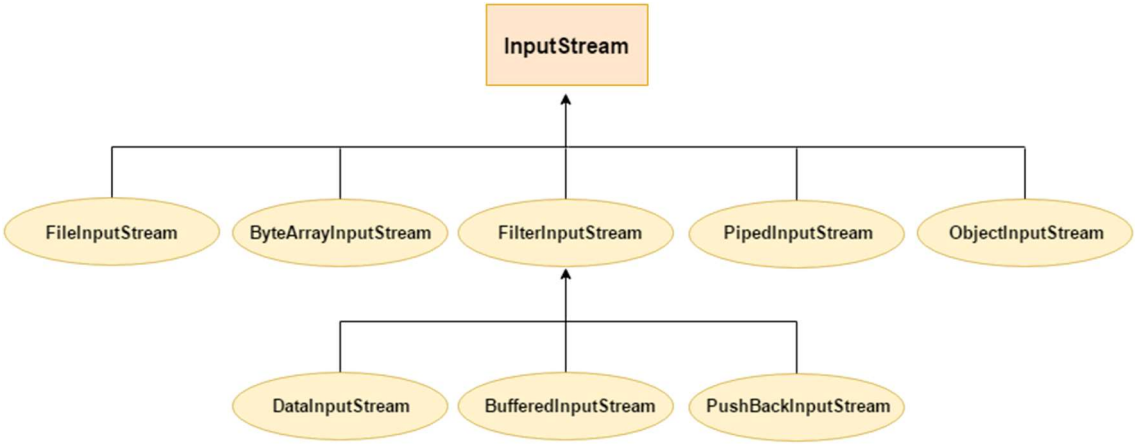


Fig. InputStream class Hierarchy

OutputStream class

OutputStream class is an abstract class. It is the super class of all classes representing anoutput stream of bytes. An output stream accepts output bytes and sends them to some sink.

Class	Description
Buffered Output Stream	Contains methods to write bytes into the buffer
Byte Array Output Stream	Contains methods to write bytes into a byte array
Data Output Stream	Contains methods to write Java primitive data types
File Output Stream	Contains methods to write bytes to a file
Filter Output Stream	Contains methods to write to other output streams
Object Output Stream	Contains methods to write objects
Piped Output Stream	Contains methods to write to a piped output stream
Print Stream	Contains methods to print Java primitive data types

Some of the useful methods of OutputStream class are listed below.

Method	Description
public void write(int)throws IO Exception	Write a byte to the current output stream.
public void write(byte[]) throws IO Exception	Write an array of byte to the current output stream.
public void flush()throws IO Exception	Flushes the current output stream.
public void close()throws IO Exception	close the current output stream.

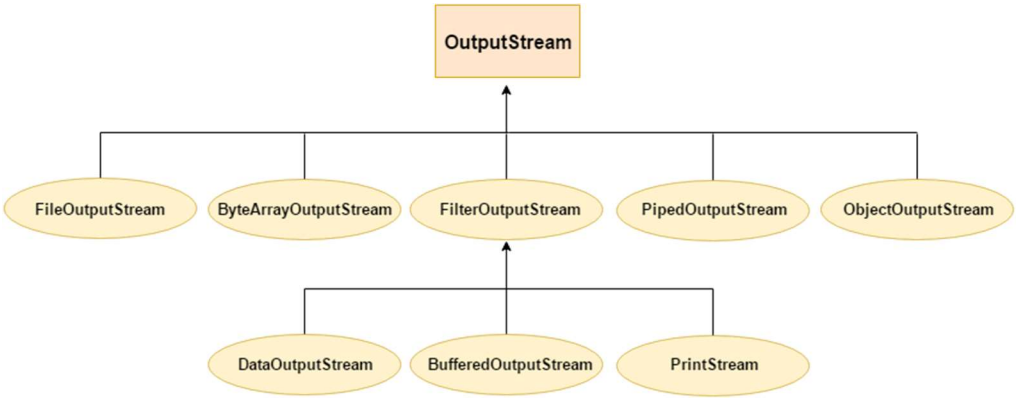
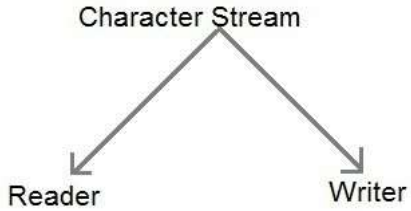


Fig. OutputStream class Hierarchy

Character Stream Classes

The character stream classes are also topped by two abstract classes Reader and Writer.



Some important Character stream reader classes are listed below.

Reader classes are used to read 16-bit unicode characters from the input stream.

- The Reader class is the superclass for all character-oriented input stream classes.
- All the methods of this class throw an IO Exception.
- Being an abstract class, the Reader class cannot be instantiated hence its subclasses are used.

Reader class	Description
BufferedReader	Contains methods to read characters from the buffer
FileReader	Contains methods to read from a file
InputStreamReader	Contains methods to convert bytes to characters
Reader	Abstract class that describes character stream input

The Reader class defines various methods to perform reading operations on data of aninput stream. Some of these methods are listed below.

Method	Description
int read()	returns the integral representation of the next available character of input. It returns -1 when end of file is encountered
int read (char buffer [])	attempts to read buffer. length characters into the buffer and returns the total number of characters successfully read. It returns -I when end of file is encountered
int read (char buffer [], int loc, int nChars)	attempts to read 'nChars' characters into the buffer starting at buffer [loc] and returns the total number of characters successfully read. It returns -1 when end of file is encountered
long skip (long nChars)	skips 'nChars' characters of the input stream and returns the number of actually skipped characters
void close ()	closes the input source. If an attempt is made to read even after closing the stream then it generates IOException

Some important Character stream writer classes are listed below.

Writer classes are used to write 16-bit Unicode characters onto an outputstream.

- The Writer class is the superclass for all character-oriented output stream classes.
- All the methods of this class throw an IOException.
- Being an abstract class, the Writer class cannot be instantiated hence, its subclasses are used.

Writer class	Description
BufferedWriter	Contains methods to write characters to a buffer
FileWriter	Contains methods to write to a file
OutputStreamReader	Contains methods to convert from bytes to character
PrintWriter	Output stream that contains print( ) and println( )
Writer	Abstract class that describes character stream output

The Writer class defines various methods to perform writing operations on output stream. Some of these methods are listed below.

Method	Description
void write ()	writes data to the output stream
void write (int i)	Writes a single character to the output stream
void write (char buffer [])	writes an array of characters to the output stream
void write(char buffer [],int loc, int nChars)	writes 'n' characters from the buffer starting at buffer [loc] to the output stream
void close ()	closes the output stream. If an attempt is made to perform writing operation even after closing the stream then it generates IOException
void flush ()	flushes the output stream and writes the waiting buffered output characters

Predefined Streams

Java provides the following three standard streams −

- Standard Input − refers to the standard InputStream which is the keyboard by default. This is used to feed the data to user's program and represented as **System.in**.
- Standard Output − refers to the standard OutputStream by default,this is console and represented as **System.out**.
- Standard Error − This is used to output the error data produced by the user's programand usually a computer screen is used for standard error stream and representedas **System.err**.

The System class is defined in java.lang package. It contains three predefined stream variables: in, out, err. These are declared as public and static within the system.

READING CONSOLE INPUT

Reading characters

The read() method is used with BufferedReader object to read characters. As this function returns integer type value has we need to use typecasting to convert it into char type.

Syntax:

int read() throws IOException

Example:

Read character from keyboard

```
import java.io.*;
class Main
{
 public static void main( String args[]) throws IOException
```

```
{
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 char c;
 System.out.println("Enter characters, @ to quit");
 do{
   c = (char)br.read();       //Reading character
    System.out.println(c);
 }while(c!='@');
}
}
```

Sample Output:

Enter characters, @ to quit

abcd23@

a

b

c

d

2

3

@

Example:

Read string from keyboard

The readLine() function with BufferedReader class's object is used to read string from keyboard.

Syntax:

String readLine() throws IOException

Example :

```
import java.io.*;
public class Main{
public static void main(String args[])throws Exception{
```

```
    InputStreamReader r=new InputStreamReader(System.in);
    BufferedReader br=new BufferedReader(r);
    System.out.println("Enter your name");
    String name=br.readLine();
    System.out.println("Welcome "+name);
  }
  }
```

Sample Output :

```
Enter your name
Priya
Welcome Priya
```

WRITING CONSOLE OUTPUT

- Console output is most easily accomplished with print( ) and println( ). These methods are defined by the class PrintStream (which is the type of object referenced by System. out).
- Since PrintStream is an output stream derived from OutputStream, it also implements the low-level method write( ).
- So, write( ) can be used to write to the console.

Syntax:

```
void write(int byteval)
```

This method writes to the stream the byte specified by byteval.

The following java program uses write( ) to output the character "A" followed by a new-line to the screen:

```
// Demonstrate System.out.write().
class WriteDemo
 {
public static void main(String args[])
{
 int b;
b = 'A';
```

```
System.out.write(b);
System.out.write('\n');
 }
 }
```

THE PRINTWRITER CLASS

- Although using System.out to write to the console is acceptable, its use is recommended mostly for debugging purposes or for sample programs.
- For real-world programs, the recommended method of writing to the console when using Java is through a PrintWriter stream.
- PrintWriter is one of the character-based classes.
- Using a character-based class for console output makes it easier to internationalize our program.
- PrintWriter defines several constructors.

Syntax:

PrintWriter(OutputStream outputStream, boolean flushOnNewline)Here,

- output Stream is an object of type OutputStream
- flushOnNewline controls whether Java flushes the output stream every time a println( ) method is called.
- If flushOnNewline is true, flushing automatically takes place. If false, flushing is not automatic.
- PrintWriter supports the print( ) and println( ) methods for all types including Object.
- Thus, we can use these methods in the same way as they have been used with System. out.
- If an argument is not a simple type, the PrintWriter methods call the object's toString( ) method and then print the result.
- To write to the console by using a PrintWriter, specify System.out for the output stream and flush the stream after each newline.

For example, the following code creates a PrintWriter that is connected to console output:

```
PrintWriter pw = new PrintWriter(System.out, true);
```

The following application illustrates using a PrintWriter to handle console output:

```
// Demonstrate PrintWriter
import java.io.*;
public class PrintWriterDemo
{
 public static void main(String args[])
{
PrintWriter pw = new PrintWriter(System.out, true);
pw.println("This is a string");
int i = -7; pw.println(i); double d = 4.5e-7;pw.println(d);
 }
 }
```

Sample Output:

```
This is a string
-7
 4.5E-7
```

READING AND WRITING FILES

In Java, all files are byte-oriented, and Java provides methods to read and write bytes fromand to a file.

Two of the most often-used stream classes are FileInputStream and FileOutputStream, which create byte streams linked to files.

File Input Stream

This stream is used for reading data from the files. Objects can be created using the key-word new and there are several types of constructors available.

The two constructors which can be used to create a FileInputStream object:

i) Following constructor takes a file name as a string to create an input stream object to read the file:

```
InputStream f = new FileInputStream("filename ");
```

ii) Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows:

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Methods to read to stream or to do other operations on the stream.

Method	Description
public void close() throws IOException{}	• Closes the file output stream. • Releases any system resources associated with the file. • Throws an IOException.
protected void finalize()throws IOException {}	• Ceans up the connection to the file. • Ensures that the close method of this file output stream is called when there are no more references to this stream. • Throws an IOException.
public int read(int r)throws IOException{}	• Reads the specified byte of data from the InputStream. • Returns an int. • Returns the next byte of data and -1 will be returned if it's the end of the file.
public int read(byte[] r) throws IOException{}	• Reads r.length bytes from the input stream into an array. • Returns the total number of bytes read. If it is the end of the file, -1 will be returned.
public int available() throws IOException{}	• Gives the number of bytes that can be read from this file input stream. • Returns an int.

File Output Stream

FileOutputStream is used to create a file and write data into it.

The stream would create a file, if it doesn't already exist, before opening it for output.

The two constructors which can be used to create a FileOutputStream object:

i) Following constructor takes a file name as a string to create an input stream object to write the file:

```
OutputStream f = new FileOutputStream("filename");
```

ii) Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows:

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Methods to write to stream or to do other operations on the stream

Method	Description
public void close() throws IO-Exception{}	• Closes the file output stream. • Releases any system resources associated with the file. • Throws an IOException.
protected void finalize()throws IOException {}	• Cleans up the connection to the file. • Ensures that the close method of this file output stream is called when there are no more references to this stream. • Throws an IOException.
public void write(int w)throws IOException{}	• Writes the specified byte to the output stream.
public void write(byte[] w)	• Writes w.length bytes from the mentioned byte array to the OutputStream.

Following code demonstrates the use of InputStream and OutputStream.

```
import java.io.*;
public class fileStreamTest
{
 public static void main(String args[])
{
try
{
        byte bWrite [] = {11,21,3,40,5};
        OutputStream os = new FileOutputStream("test.txt");
        for(int x = 0; x < bWrite.length ; x++)
        {
          os.write( bWrite[x] );  // writes the bytes
        }
        os.close();
        InputStream is = new FileInputStream("test.txt");int
        size = is.available();
        for(int i = 0; i < size; i++)
        {
          System.out.print((char)is.read() + "  ");
        }
```

```
                    is.close();
                }
    catch (IOException e)
    {
        System.out.print("Exception");
            }
        }
    }
```

The above code creates a file named test.txt and writes given numbers in binary format. The same will be displayed as output on the stdout screen.

Generics: Generic Programming

It would be nice if we could write a single sort method that could sort the elements in an Integer array, a String array, or an array of any type that supports ordering.

Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

Generic classes

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized

classes or parameterized types because they accept one or more parameters.

Example

Following example illustrates how we can define a generic class:

```
public class Box<T> {

  private T t;

  public void add(T t) {
    this.t = t;
  }

  public T get() {
    return t;
  }

  public static void main(String[] args) {
     Box<Integer> integerBox = new Box<Integer>();
     Box<String> stringBox = new Box<String>();

     integerBox.add(new Integer(10));
     stringBox.add(new String("Hello World"));

     System.out.printf("Integer Value :%d\n\n", integerBox.get());
     System.out.printf("String Value :%s\n", stringBox.get());
  }
}
```

This will produce the following result:

```
Integer Value :10
String Value :Hello World
```

Generic Methods

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods:

 All generic method declarations have a type parameter section delimited by angle

brackets (< and >) that precedes the method's return type (< E > in the next example).

 Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.

 The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.

 A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

Example

Following example illustrates how we can print an array of different type using a single Generic method:

```
public class GenericMethodTest
{
   // generic method printArray
   public static < E > void printArray( E[] inputArray )
   {
      // Display array elements
         for ( E element : inputArray ){
            System.out.printf( "%s ", element );
         }
         System.out.println();
    }

    public static void main( String args[] )
    {
        // Create arrays of Integer, Double and Character
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
```

```
        System.out.println( "Array integerArray contains:" );
        printArray( intArray  ); // pass an Integer array

        System.out.println( "\nArray doubleArray contains:" );
        printArray( doubleArray ); // pass a Double array

        System.out.println( "\nArray characterArray contains:" );
        printArray( charArray ); // pass a Character array
    }
}
```

This will produce the following result:

```
Array integerArray contains:
1 2 3 4 5 6

Array doubleArray contains:
1.1 2.2 3.3 4.4

Array characterArray contains:
H E L L O
```

Bounded Types parameters

There may be times when you'll want to restrict the kinds of types that are allowed to be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.

To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its upper bound.

Example

Following example illustrates how extends is used in a general sense to mean either "extends" (as in classes) or "implements" (as in interfaces). This example is Generic method to return the largest of three Comparable objects:

```
public class MaximumTest
{
   // determines the largest of three Comparable objects
   public static <T extends Comparable<T>> T maximum(T x, T y, T z)
   {
      T max = x; // assume x is initially the largest
      if ( y.compareTo( max ) > 0 ){
         max = y; // y is the largest so far
      }
      if ( z.compareTo( max ) > 0 ){
         max = z; // z is the largest now
      }

      return max; // returns the largest object
   }
   public static void main( String args[] )
   {
      System.out.printf( "Max of %d, %d and %d is %d\n\n",
                  3, 4, 5, maximum( 3, 4, 5 ) );

      System.out.printf( "Maxm of %.1f,%.1f and %.1f is %.1f\n\n",
                  6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );

      System.out.printf( "Max of %s, %s and %s is %s\n","pear",
         "apple", "orange", maximum( "pear", "apple", "orange" ) );
   }
}
```

This will produce the following result:

```
maximum of 3, 4 and 5 is 5

maximum of 6.6, 8.8 and 7.7 is 8.8

maximum of pear, apple and orange is pear
```

Restrictions and Limitations

1. Type Parameters Cannot Be Instantiated with Primitive Types
2. Runtime Type Inquiry Only Works with Raw Types
3. You Cannot Create Arrays of Parameterized Types
4. Varargs Warnings
5. You Cannot Instantiate Type Variables
6. You Cannot Construct a Generic Array
7. Type Variables Are Not Valid in Static Contexts of Generic Classes
8. You Cannot Throw or Catch Instances of a Generic Class
9. You Can Defeat Checked Exception Checking
10. Beware of Clashes after Erasure

Strings: Basic String class

Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects.

The Java platform provides the String class to create and manipulate strings.

The most direct way to create a string is to write:

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

Whenever it encounters a string literal in your code, the compiler creates a String object. with its value in this case, "Hello world!'.

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has 11 constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

```
public class StringDemo{

   public static void main(String args[]){
      char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};
      String helloString = new String(helloArray);
      System.out.println( helloString );
   }
}
```

This will produce the following result:

```
hello.
```

String methods

Here is the list of methods supported by String class:

Sr. No.	Methods with Description
1	**char charAt(int index)** Returns the character at the specified index.
2	**int compareTo(Object o)** Compares this String to another Object.
3	**int compareTo(String anotherString)** Compares two strings lexicographically.
4	**int compareToIgnoreCase(String str)** Compares two strings lexicographically, ignoring case differences.

5	**String concat(String str)** Concatenates the specified string to the end of this string.
6	**boolean contentEquals(StringBuffer sb)** Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	**static String copyValueOf(char[] data)** Returns a String that represents the character sequence in the array specified.
8	**static String copyValueOf(char[] data, int offset, int count)** Returns a String that represents the character sequence in the array specified.
9	**boolean endsWith(String suffix)** Tests if this string ends with the specified suffix.
10	**boolean equals(Object anObject)** Compares this string to the specified object.

11	**boolean equalsIgnoreCase(String anotherString)** Compares this String to another String, ignoring case considerations.
12	**byte getBytes()** Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	**byte[] getBytes(String charsetName)** Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
14	**void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** Copies characters from this string into the destination character array.

15	**int hashCode()** Returns a hash code for this string.
16	**int indexOf(int ch)** Returns the index within this string of the first occurrence of the specified character.
17	**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.
18	**int indexOf(String str)** Returns the index within this string of the first occurrence of the specified substring.
19	**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.
20	**String intern()** Returns a canonical representation for the string object.

21	**int lastIndexOf(int ch)** Returns the index within this string of the last occurrence of the specified character.
22	**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.
23	**int lastIndexOf(String str)** Returns the index within this string of the rightmost occurrence of the specified substring.
24	**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.

STRINGS IN JAVA

In java, string is basically an object that represents sequence of char values**. Java String** provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc.
In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.
String s="javatpoint";

There are two ways to create String object:
1. By string literal
2. By new keyword

1) String Literal
Java String literal is created by using double quotes. For Example:
String s="welcome";

2) By new keyword
String s=new String("Welcome");

char[] ch={'w','e','l','c','o','m','e'};
String s=new String(ch);

String methods:
1. char charAt(int index) returns char value for the particular index
2. int length() returns string length
3. static String format(String format, Object... args) returns formatted string
4. static String format(Locale l, Stringformat, Object... args) returns formatted string with given locale
5. String substring(int beginIndex) returns substring for given begin index
6. String substring(int beginIndex, int endIndex) returns substring for given begin index and end index
7. boolean contains(CharSequence s) returns true or false after matching the sequence of char value
8. static String join(CharSequence delimiter, CharSequence... elements) returns a joined string
9. static String join(CharSequence delimiter, Iterable<? Extends CharSequence> elements) returns a joined string
10. boolean equals(Object another) checks the equality of string with object
11. boolean isEmpty() checks if string is empty
12. String concat(String str) concatinates specified string
13. String replace(char old, char new) replaces all occurrences of specified char value
14. String replace(CharSequence old, CharSequence new) replaces all occurrences of specified CharSequence
15. static String equalsIgnoreCase(String another) compares another string. It doesn't check case.
16. String[] split(String regex) returns splitted string matching regex

17. String[] split(String regex, int limit) returns splitted string matching regex and limit
18. String intern() returns interned string
19. int indexOf(int ch) returns specified char value index
20. int indexOf(int ch, int fromIndex) returns specified char value index starting with given index
21. int indexOf(String substring) returns specified substring index
22. int indexOf(String substring, int fromIndex) returns specified substring index starting with given index
23. String toLowerCase() returns string in lowercase.
24. String toLowerCase(Locale l) returns string in lowercase using specified locale.
25. String toUpperCase() returns string in uppercase.
26. String toUpperCase(Locale l) returns string in uppercase using specified locale.
27. String trim() removes beginning and ending spaces of this string.
28. static String valueOf(int value) converts given type into string. It is overloaded.

Example:

```
public classstringmethod
{
public static void main(String[] args)
{
String string1 = new String("hello");
String string2 = new String("hello");
if (string1 == string2)
{
System.out.println("string1= "+string1+" string2= "+string2+" are equal");
}
else
{
System.out.println("string1= "+string1+" string2= "+string2+" are Unequal");
}
System.out.println("string1 and string2 is=
"+string1.equals(string2)); String a="information";
System.out.println("Uppercase of String a is= "+a.toUpperCase());
String b="technology";
System.out.println("Concatenation of object a and b is= "+a.concat(b));
System.out.println("After concatenation Object a is= "+a.toString());
System.out.println("Length of Object a is= "+a.length());
System.out.println("The third character of Object a is= "+a.charAt(2));
StringBuffer n=new StringBuffer("Technology");
StringBuffer m=new StringBuffer("Information");
System.out.println("Reverse of Object n is= "+n.reverse());
n= new StringBuffer("Technology");
System.out.println("Concatenation of Object m and n is= "+m.append(n));
System.out.println("After concatenation of Object m is= "+m);
}
}
```

Output:

```
string1= hello string2= hello are Unequal
string1 and string2 is= true
Uppercase of String a is= INFORMATION
Concatenation of object a and b is= informationtechnology
After concatenation Object a is= information
Length of Object a is= 11
The third character of Object a is= f
Reverse of Object n is= ygolonhceT
Concatenation of Object m and n is= InformationTechnology
```



```
public class StringMethodsDemo {
   public static void main(String[] args) {
      // Creating a string
      String str1 = "Hello, World!";
      String str2 = new String("Java Programming");

      // Length of a string
      int length1 = str1.length();
      int length2 = str2.length();

      System.out.println("str1 length: " + length1); // 13
      System.out.println("str2 length: " + length2); // 16

      // Concatenation
      String concat = str1 + " " + str2;
      System.out.println("Concatenation: " + concat);

      // Substring
      String sub1 = str1.substring(0, 5);
      String sub2 = str2.substring(5);
      System.out.println("Substring 1: " + sub1); // "Hello"
      System.out.println("Substring 2: " + sub2); // " Programming"

      // Searching
      int index1 = concat.indexOf("World");
      int index2 = concat.indexOf("Java");
      boolean containsJava = concat.contains("Java");

      System.out.println("Index of 'World': " + index1); // 7
      System.out.println("Index of 'Java': " + index2);  // 13
      System.out.println("Contains 'Java': " + containsJava); // true

      // Changing Case
      String upperCase = concat.toUpperCase();
      String lowerCase = concat.toLowerCase();
```

```
        System.out.println("Uppercase: " + upperCase);
        System.out.println("Lowercase: " + lowerCase);

        // String Comparison
        String str3 = "hello, world!";
        boolean isEqual = str1.equals(str3);
        boolean isEqualIgnoreCase = str1.equalsIgnoreCase(str3);

        System.out.println("str1 equals str3: " + isEqual); // false
        System.out.println("str1 equalsIgnoreCase str3: " + isEqualIgnoreCase); // true

        // Replacing Substrings
        String replaced = concat.replace("Java", "Python");
        System.out.println("Replaced: " + replaced);
    }
}
```

StringBuffer class in Java

StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time. StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

1. StringBuffer objects are mutable, meaning that you can change the contents of the buffer without creating a new object.
2. The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the ensureCapacity() method.
3. The append() method is used to add characters, strings, or other objects to the end of the buffer.
4. The insert() method is used to insert characters, strings, or other objects at a specified position in the buffer.
5. The delete() method is used to remove characters from the buffer.
6. The reverse() method is used to reverse the order of the characters in the buffer.

Important Constructors of StringBuffer class

- StringBuffer(): creates an empty string buffer with an initial capacity of 16.
- StringBuffer(String str): creates a string buffer with the specified string.
- StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

1. append() method

The append() method concatenates the given argument with this string.

2. insert() method

The insert() method inserts the given string with this string at the given position.

3. replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex-1.

4. delete() method

The delete() method of the StringBuffer class deletes the string from the specified beginIndex to endIndex-1

5. reverse() method

The reverse() method of the StringBuilder class reverses the current string.

6. capacity() method

- The capacity() method of the StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of characters increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.
- For instance, if your current capacity is 16, it will be (16*2)+2=34.

Example Programs

```
/*public class StringBufferExample {
   public static void main(String[] args)
   {
      StringBuffer sb = new StringBuffer();
         sb.append("Hello");
      sb.append(" ");
      sb.append("world");
         System.out.println(sb);

         String s=new String("Hello ");
         System.out.println(s.concat("World"));
         System.out.println(s);

   }
}*/

/*class StringBufferExample {
   public static void main(String args[])
   {
      StringBuffer sb = new StringBuffer("Hello ");
      sb.insert(1, "Java");
      // Now original string is changed
      System.out.println(sb);
   }
}*/

/*class StringBufferExample {
   public static void main(String args[])
   {
      StringBuffer sb = new StringBuffer("Hello");
```

```
        sb.replace(1, 3,"Welcome");
        System.out.println(sb);
    }
}*/


/*class StringBufferExample {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.delete(1, 3);
        System.out.println(sb);
    }
}*/


/*class StringBufferExample {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.reverse();
        System.out.println(sb);
    }
}*/

class StringBufferExample {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer();
        System.out.println(sb.capacity()); // default 16
        sb.append("Hello");
        System.out.println(sb.capacity()); // now 16
        sb.append("java is my favourite language");
        System.out.println(sb.capacity());
        // Now (16*2)+2=34     i.e (oldcapacity*2)+2
    }
}

class StringBufferDemo {
    public static void main(String[] args) {
        // Creating a StringBuffer object
        StringBuffer stringBuffer = new StringBuffer("Hello");

        // Append text to the StringBuffer
        stringBuffer.append(" World");
        System.out.println("After appending: " + stringBuffer);

        // Insert text at a specific position
        stringBuffer.insert(5, " Java");
        System.out.println("After inserting: " + stringBuffer);
```

```
        // Delete characters from the StringBuffer
        stringBuffer.delete(5, 10);
        System.out.println("After deleting: " + stringBuffer);

        // Reverse the contents of the StringBuffer
        stringBuffer.reverse();
        System.out.println("After reversing: " + stringBuffer);

        // Get the length of the StringBuffer
        int length = stringBuffer.length();
        System.out.println("Length of the StringBuffer: " + length);

        // Get the capacity of the StringBuffer
        int capacity = stringBuffer.capacity();
        System.out.println("Capacity of the StringBuffer: " + capacity);

        // Set the length of the StringBuffer
        stringBuffer.setLength(5);
        System.out.println("After setting length: " + stringBuffer);

        // Ensure the capacity of the StringBuffer
        stringBuffer.ensureCapacity(20);
        System.out.println("After ensuring capacity: " + stringBuffer);
    }
}
```

StringBuilder Class in Java

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 synchronisation. The StringBuilder class provides no guarantee of synchronisation 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). 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.

Constructors in Java StringBuilder Class

- **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.

```
public class StringBuilderDemo {
   public static void main(String[] argv)
   {
      StringBuilder str = new StringBuilder();
       str.append("HELLO");
       System.out.println("String = " + str);
          System.out.println("String capacity = "+ str.capacity());


      StringBuilder str1 = new StringBuilder("AAAABBBCCCC");
       System.out.println("String1 = " + str1);
         System.out.println("String1 capacity = "+ str1.capacity());


      StringBuilder str2 = new StringBuilder(10);
       System.out.println("String2 capacity = "+ str2.capacity());

         StringBuilder str3 = new StringBuilder(str);
       System.out.println("String3 = " + str3);
          System.out.println("String3 capacity = "+ str3.capacity());

       }
}
```

Methods in Java StringBuilder

StringBuilder append(X x)**: This method appends the string representation of the X type argument to the sequence.**

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. **IntStream chars()**: This method returns a stream of int zero-extending the char values from this sequence.
5. int codePointAt(int index): This method returns the character (Unicode code point) at the specified index.
6. int codePointBefore(int index): This method returns the character (Unicode code point) before the specified index.
7. int codePointCount(int beginIndex, int endIndex): This method returns the number of Unicode code points in the specified text range of this sequence.
8. **IntStream codePoints()**: This method returns a stream of code point values from this sequence.
9. StringBuilder delete(int start, int end): This method removes the characters in a substring of this sequence.

10. **StringBuilder deleteCharAt(int index)**: This method removes the char at the specified position in this sequence.
11. void ensureCapacity(int minimumCapacity): This method ensures that the capacity is at least equal to the specified minimum.
12. void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin): This method characters are copied from this sequence into the destination character array dst.
13. int indexOf(): This method returns the index within this string of the first occurrence of the specified substring.
14. **StringBuilder insert(int offset, boolean b)**: This method inserts the string representation of the boolean alternate argument into this sequence.
15. **StringBuilder insert()**: This method inserts the string representation of the char argument into this sequence.
16. int lastIndexOf(): This method returns the index within this string of the last occurrence of the specified substring.
17. int length(): This method returns the length (character count).
18. **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.
19. 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.
20. StringBuilder reverse(): This method causes this character sequence to be replaced by the reverse of the sequence.
21. void setCharAt(int index, char ch): In this method, the character at the specified index is set to ch.
22. void setLength(int newLength): This method sets the length of the character sequence.
23. CharSequence subSequence(int start, int end): This method returns a new character sequence that is a subsequence of this sequence.
24. String substring(): This method returns a new String that contains a subsequence of characters currently contained in this character sequence.
25. String toString(): This method returns a string representing the data in this sequence.
26. void trimToSize(): This method attempts to reduce storage used for the character sequence.

StringBuilder appendCodePoint() method

The appendCodePoint(int codePoint) method of StringBuilder class is the inbuilt method used to append the string representation of the codePoint argument to this sequence.

Syntax:

public StringBuilder appendCodePoint(int codePoint)

Parameters: This method accepts only one parameter **codePoint** which is int type value refers to a Unicode code point.

```
public class StringBuilderMethods {
   public static void main(String[] args)
   {
      StringBuilder str = new StringBuilder("Welcome");
      System.out.println("StringBuilder = "+ str);
       // Append 'C'(67) to the String
```

```
        str.appendCodePoint(67);
         // Print the modified String
        System.out.println("Modified StringBuilder = "+ str);
    }
}
```

public int codePointAt(int index)
Parameters: This method accepts one int type parameter **index** which represents index of the character whose unicode value to be returned.

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

        StringBuilder str = new StringBuilder();
          str.append("WELCOME");
           System.out.println("StringBuilder Object"+ " contains = " + str);
          int unicode = str.codePointAt(1);
          System.out.println(unicode);
          unicode = str.codePointAt(2);
          System.out.println(unicode);
    }
}
```

public int codePointBefore(int index)
Parameters: This method accepts one int type parameter **index** represents index of the character following the character whose unicode value to be returned. **Return Value:** This method returns **"unicode number"** of the character before the given index. **Exception:** This method throws **IndexOutOfBoundsException** when index is negative or greater than or equal to length().

```
class StringBuilderMethods{
    public static void main(String[] args)
    {
        // create a StringBuilder object
        // with a String pass as parameter
        StringBuilder str= new StringBuilder("Welcome Geeks");
            System.out.println(str);
         // get unicode of char at index 1
        // using codePointBefore() method
        int unicode = str.codePointBefore(2);
         System.out.println(unicode);

        // get unicode of char at index 10
        // using codePointBefore() method
        unicode = str.codePointBefore(11);
        System.out.println(unicode);
    }
}
```

public int codePointCount(int beginIndex, int endIndex)
Parameters: This method accepts two parameters

- **beginIndex:** index of the first character of the text range.
- **endIndex:** index after the last character of the text range.

Return Value: This method returns **the number of Unicode code points in the specified text range.**

```
class StringBuilderMethods{
   public static void main(String[] args)
   {
      StringBuilder str= new StringBuilder("WelcomeGeeks");
      System.out.println("String = " + str);

      // returns the codepoint count from index 2 to 8
      int codepoints = str.codePointCount(2, 8);
      System.out.println(codepoints);
         for(int i=2;i<8;i++)
         System.out.println((char)str.codePointAt(i));
   }
}
```

ensureCapacity(int minimumCapacity)
The **ensureCapacity(int minimumCapacity)** method of **StringBuilder class** helps us to ensures the capacity is at least equal to the specified minimumCapacity passed as the parameter to the method.

- If the current capacity of StringBuilder is less than the argument minimumCapacity, then a new internal array is allocated with greater capacity.
- If the minimumCapacity argument is greater than twice the old capacity, plus 2 then new capacity is equal to minimumCapacity else new capacity is equal to twice the old capacity, plus 2.
- If the minimumCapacity argument passed as parameter is not-positive, this method takes no action.

```
class StringBuilderMethods{
   public static void main(String[] args)
   {
      StringBuilder str = new StringBuilder();
        System.out.println(str.capacity());
        // apply ensureCapacity()
      str.ensureCapacity(18);
        // print string capacity
      System.out.println(str.capacity());
         str.ensureCapacity(35);
        // print string capacity
      System.out.println(str.capacity());
   }
}
```

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

The **getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** method of StringBuilder class **copies the characters starting at the given index:srcBegin to index:srcEnd-1 from String contained by StringBuilder into an array of char passed as parameter to function.**

- The characters are copied from StringBuilder into the array dst[] starting at index:dstBegin and ending at index:dstbegin + (srcEnd-srcBegin) – 1.
- The first character to be copied from StringBuilder to array is at index srcBegin and the last character to be copied is at index srcEnd-1.
- The total number of characters to be copied is equal to srcEnd-srcBegin.

Exception: This method throws StringIndexOutOfBoundsException if:

srcBegin < 0
dstBegin < 0
srcBegin > srcEnd
srcEnd > this.length()
dstBegin+srcEnd-srcBegin > dst.length

```
class StringBuilderMethods {
   public static void main(String[] args)
   {
       StringBuilder str= new StringBuilder("WelcomeGeeks");
       System.out.println("String = "+ str);
        char[] array = new char[7];
      str.getChars(0, 7, array, 0);
        // print char array after operation
      System.out.print("Char array contains : ");

      for (int i = 0; i < array.length; i++) {
          System.out.print(array[i] + " ");
      }
   }
}
```

trimToSize() method

The trimToSize() method of StringBuilder class is the inbuilt method used to trims the capacity used for the character sequence of StringBuilder object. If the buffer used by StringBuilder object is larger than necessary to hold its current sequence of characters, then this method is called to resize the StringBuilder object.

```
class StringBuilderMethods {
   public static void main(String[] args)
   {
       StringBuilder str = new StringBuilder("GeeksForGeeks");
       str.append("Contribute");
       System.out.println(str.capacity());
       str.trimToSize();
      System.out.println("String = " + str);
      System.out.println(str.capacity());
   }
}
```

****