```
public class ThreadGroupImpDemo {
        public static void main(String args[]) throws InterruptedException {

                ThreadGroup grpA = new ThreadGroup("Group A");
                ThreadGroup grpB = new ThreadGroup("Group B");

                ThreadGroupImp th1 = new ThreadGroupImp(grpA, "Thread1");
                ThreadGroupImp th2 = new ThreadGroupImp(grpA, "Thread2");
                ThreadGroupImp th3 = new ThreadGroupImp(grpB, "Thread3");
                ThreadGroupImp th4 = new ThreadGroupImp(grpB, "Thread4");


                grpA.list();
                grpB.list();

                System.out.println("Suspending GroupA...");
                grpA.suspend();

                Thread.sleep(2000);

                System.out.println("Resuming GroupA...");
                grpA.resume();

                // Waiting for the threads to end
                th1.t.join();
                th2.t.join();
                th3.t.join();
                th4.t.join();

                System.out.println("Main thread exiting...");
        }

}
```

Two different models of applications

Based on the number of threads used by an application, applications are divided into two types

1. Single thread model application- An application that uses single thread to execute multiple tasks is called single thread model application. It executes all tasks sequentially.

2. Multithread model application- An application that uses more than one thread to execute all tasks is called multithread model application. It executes all tasks concurrently. This application model is recommended only if all tasks are independent to each other.

Q) What is an "inline" thread?
A) A thread that is created with anonymous thread object is called inline thread. Anonymous thread object means a thread that is created without extending from Thread class or implementing from Runnable interface explicitly is call "anonymous" thread object. The other name for anonymous thread object is "inline" thread.

Note: It is not the official word given by SUN Microsystems. It is invented by our textbook authors or software industry experts.

How can we create Thread without extending from Thread class or implementing from Runnable interface explicitly?
Using anonymous inner class

```
//AnonymousThread.java – creates inline thread with Thread object
class AnonymousThread {
        public static void main(String[] args) {

                ( new Thread()
                  {
                        public void run(){
                                for (int i = 1; i<= 10; i++){
                                        System.out.println("run: "+i);
                                }
                        }
                  }
                ).start();
        }
}
//AnonymousRunnable.java – create inline thread with Runnable object
class AnonymousRunnable {
        public static void main(String[] args) {

                ( new Thread(
                        new Runnable()
                        {
                                public void run(){
                                        for (int i = 1; i<= 10; i++){
                                                System.out.println("run: "+i);
                                        }
                                }
                        }
                  )
                ).start();
        }
}
```

What is multithreading?
The process of creating multiple threads in a single application is called multithreading.

Multithreading is a technique by which a single set of code can be used by several threads at different stages of execution.

Unlike most other computer languages, Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum. This is especially important for the interactive, networked environment in which Java operates, because idle time is common. For example, the transmission rate of data over a network is much slower than the rate at which the computer can process it. Even local file system resources are read and written at a much slower pace than they can be processed by the CPU. And, of course, user input is much slower than the computer. In a traditional, single-threaded environment, your program has to wait for each of these tasks to finish before it can proceed to the next one—even though the CPU is sitting idle most of the time. Multithreading lets you gain access to this idle time and put it to good use.

Difference between multitasking and Multithreading
Multitasking:
Multitasking allow to execute more than one tasks at the same time, a task being a program. In multitasking only one CPU is involved but it can switches from one program to another program so quickly that's why it gives the appearance of executing all of the programs at the same time. Multitasking allow processes (i.e. programs) to run concurrently on the program. For Example running the spreadsheet program and you are working with word processor also. Multitasking is running heavyweight processes by a single OS.

Multithreading:
Multithreading is a technique that allows a program or a process to execute many tasks concurrently (at the same time and parallel). It allows a process to run its tasks in parallel mode on a single processor system

In the multithreading concept, several multiple lightweight processes are run in a single process/task or program by a single processor. For Example, when you use a word processor you perform a many different tasks such as printing, spell checking and so on. Multithreaded software treats each process as a separate program.

Advantages of multithreading over multitasking:

- Reduces the computation time.
- Improves performance of an application.
- Threads share the same address space so it saves the memory.
- Context switching between threads is usually less expensive than between processes.
- Cost of communication between threads is relatively low.

Chapter 27

String Handling

- In this chapter, You will learn
  - Need of String, StringBuffer, StringBuilder
  - Differences between String, StringBuffer and StringBuilder
  - String Pooling
  - Why String class is immutable?
  - Why StringBuffer is mutable?
  - Why StringBuilder class is given?
  - Limitation of String?
  - Special operations we can perform on StringBuffer
  - Controlling StringBuffer capacity
  - Converting String to StringBuffer and vice versa.
- By the end of this chapter- you will be comfortable in working with string data by using all above three classes.

Interview Questions

By the end of this chapter you answer all below interview questions

The **String** class

- Define string?
- In how many ways we can store string data?
- Why String class is given when char array is already available?
- What are the different ways to construct Sting Object?
- What is the difference in String object creation from the String literal and the constructor?
- Explanation of String Pooling.
- What is immutable and why String object is immutable?
- What do you mean by immutable and mutable objects?
- Can we develop immutable object?
- Why StringBuffer class is given when we have String class to store string data?
- Why StringBuilder is given when we have StringBuffer?
- Is String thread safe?
- Definition of String, StringBuffer, and StringBuilder
- Can we assign String literal directly to StringBuffer or StringBuilder type variables?

- Printing String object
- Finding is string empty or not
- How can we find length of the String?
- What is the difference between length & length()?

- The limitation of String class.
- What is meant by concatenation, how Strings can be concatenated?
- How can we convert string case in the String?
- How can we replace a character or substring in this String?
- How can we trim string leading and trailing spaces?

- How String Objects must be compared for equality?
- What is meant by comparing string objects lexicographically, how it can be done?
- How can we read characters from the String?
- How can we find the character case in the String object?
- How can we find the position of a character or sub String?
- Searching for a character or a substring?
- Finding string startsWith or endsWith
- How can we retrieve substring from the String?
- How can we convert primitive values and objects to Strings?
- Splitting string into tokens

- What are the operations we cannot perform on string using String object?

The **StringBuffer, StringBuilder** classes

- Define StringBuffer.
- When StringBuffer class must be chosen?
- What are the special operations can be performed on StringBuffer; those cannot be applied on String?
- Ways of creating StringBuffer object creation?
- Appending, Inserting, Deleting, reversing, and overriding characters in the StringBuffer.
- Finding StringBuffer capacity and length
- Controlling StringBuffer capacity
- Difference between String and StringBuffer, and StringBuffer and StringBuilder.
- Ways of converting String to StringBuffer and vice versa.

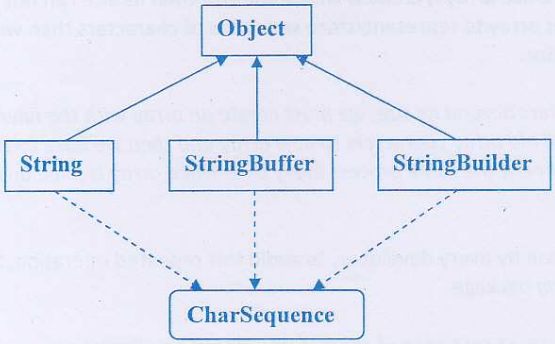
string is a sequence of characters placed in double quote (" ").
Performing different operations on string data is called String Handling.

In *java.lang* package we have below three classes to store and perform different operations on string of characters.

1. String
2. StringBuffer
3. StringBuilder

StringBuilder class was introduced in Java 5, as a *non-thread safe* class. Except this all its operations are exactly same as StringBuffer.

All these three classes are siblings, and they are subclasses of ***CharSequence*** interface. This interface also was introduced in Java 5. Below diagram shows the inheritance relation of these three classes,



Note:

- All these three classes are ***final*** classes
- These three classes are also subclasses of *java.io.Serializable* interface
- String class is a subclass of *Comparable* interface, but StringBuffer and StringBuilder are not

Any data that is place in " " is treated by compiler and JVM as an instance of java.lang.String
For example:
"abc" is an object of type java.lang.String

Find CE in the below list

```
String        s1    = "abc";   ✓
StringBuffer  sb1   = "abc";   X
StringBuilder sb2   = "abc";   X
```

We cannot assign string literal directly to StringBuffer and StringBuilder variables, it leads to CE: **in compatible types**

Q) How can we store string data in StringBuffer & StringBuilder class objects?
A) Through constructors and methods of these two classes. These constructors and methods perform copying operation. It means they store given string characters in this SB/SB object.

The **String** class

The ***String*** class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Definition
In general String can be defined as it is a sequence of characters. Strings are constant; their values cannot be changed in the same memory after they are created,

So String is defined as it is an ***immutable sequence of characters***.

Why String class is given when *char array* is available to represent sequence of characters?
String class is given to store characters dynamically without size limitation and also to perform different common operations of string data.
Explanation:
The limitation on arrays is size. Once array is created with some size then its size can not be modified. So if we use character array to represent/store sequence of characters then we can only store characters up to its size.

If we want to store new characters beyond its size, we must create an array with the new required size and should copy all old array characters to new array and then we have to store new values at end. We should repeat the same process every time when array is filled and want to add new characters.

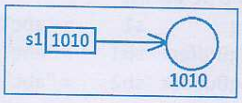
Since this operation must be done by every developer, to avoid this repeated operation, SUN has given String class in *java.lang* package.

Hence using String class we can store sequence of characters without size limitation.

What are the possible ways to create String object?
String object can be created in two ways
1. By assigning string literal directly => String s = "abc";
2. By using any one of the string class constructors

String class has totally 15 constructors among them below are the 8 important constructors, check *API documentation* for more details

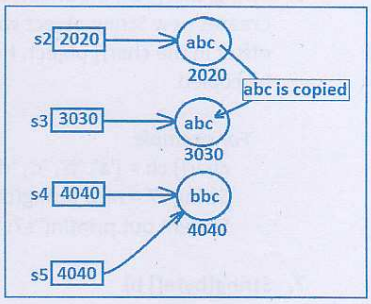
1. **String()**
   Creates empty string object, not null string object
   For example:
   ```
   String s1 = new String();
   System.out.println(s1); //no output
   ```
   string object memory structure



2. **String(String value)**
   Creates String object with given String object characters.
   It performs String copy operation

For example:

```
String s2 = "abc";

//string copy
String s3 = new String(s2);

//creating string with direct string literal
String s4 = new String("bbc");

//String assignment
String s5 = s4;

System.out.println("s2: "+s2);//abc
System.out.println("s3: "+s3);//abc
System.out.println("s4: "+s4);//bbc

System.out.println(s2 == s3);//false
System.out.println(s4 == s5);//true
```



Note:

- **In String copy** two different String objects are created with same data, it is like a file copy operations in OS.
- **In String assignment** current object reference is copied to destination variable, new object is not created.

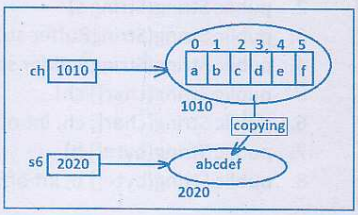
3. **String(StringBuffer sb)**
   Creates new String object with the given StringBuffer object' data.
   Performs string copy operation from StringBuffer object to String object

4. **String(StringBuilder sb)**
   Creates new String object with the given StringBuilder object content
   Performs string copy operation from StringBuilder object to String object

5. **String(char[] ch)**
   Creates String object with the given char array values
   Performs string copy operation from char[] object to String object

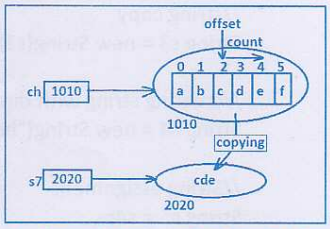
For example:

```
char[] ch = {'a', 'b', 'c', 'd', 'e', 'f'};
String s6 = new String(ch);
System.out.println("s6: "+s6);//abcdef
```



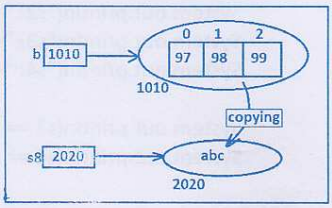
6. **String(char[] ch, int offset, int count)**
   Creates new String object with the given count number of characters from the given offset in the char[] object. Here offset is the starting index from which characters must be copied.

   For example:
   ```
   char[] ch = {'a', 'b', 'c', 'd', 'e', 'f'};
   String s7 = new String(ch, 2, 3);
   System.out.println("s7: "+s7);//cde
   ```



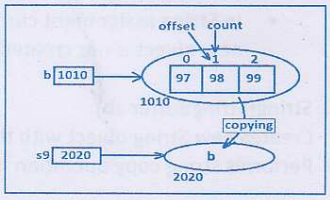
7. **String(byte[] b)**
   Creates new String object by copying the given byte[] numbers by converting them into their ASCII characters.

   For example:
   ```
   byte[] b = {97, 98, 99};
   String s8 = new String(b);
   System.out.println(b);//abc
   ```



8. **String(byte[] b, int offset, int count)**
   Creates new String object with the given count number of bytes from the given offset in the byte[]. All bytes are stored in their ASCII character form

   For example:
   ```
   byte[] b = {97, 98, 99};
   String s9 = new String(b, 1, 1);
   System.out.println("s9: "+s9);//b
   ```



Note:
Write all above object creation statements in main method and compile and execute.

3 important rules
We have there 3 important rules on String constructors, so recollect once all above 8 constructors

1. public String()
2. public String(String s)
3. public String(StringBuffer sb)
4. public String(StringBuilder sb)
5. public String(char[] ch)
6. public String(char[] ch, int offset, int count)
7. public String(byte[] b)
8. public String(byte[] b, int offset, int count)

All these constructors are overloaded constructors with siblings.

Rule #1:
For char[] and byte[] parameter constructors the give offset and count argument value must be within the range of [0 to string length -1], else it leads to
RE: ***java.lang.StringIndexOutOfBoundsException***

```
byte[] b = {97, 98, 99}
String s10 = new String(b, 1, 4);   × RE: SIOBE: 4
        //we do not have 4 bytes in array
```

Rule #2:
We cannot pass ***null*** as argument directly to constructor, it leads CE: ***ambiguous error*** because it is matched with all parameters of String class constructors.

```
String s11 = new String(null); × CE: ambiguous error
```

Rule #3:
We cannot create String object with ***null*** it leads to RE: ***java.lang.NullPointerException***

```
String s1 = null;
String s2 = new String(s1);   × RE: NPE
String s3 = new String( (StringBuffer) null);  ×  RE: NPE
```

Identify the operation I am doing in the below statements

```
String s4 = "";
String s5 = new String();
String s6 = new String("");
```
⇐ **Empty string object creation**

```
String s7 = null;
```
⇐ ***null*** **string referenced variable creation**

```
//String s8 = new String(null); CE: ambiguous error
//String s8 = new String((String)null); RE: NPE
//String s8 = new String(s7); RE: NPE
```
⇐ **We cannot create *null* String object**

```
String s = new String("null");
```
⇐ **"null" is not *null* value. It is a string with characters *n u l l*. So string object is created with characters "null"**

Note: We can create null String referenced variable (ex: s7), but we cannot create String object with null. It leads to NPE

```
String s8 = new String("A");
String s9 = new String(s8);
```
⇐ **String copy - Creating String object from another String object**

```
String s10 = new String("B");
String s11 = s10;
```
⇐ **assigning one string object reference to another String reference variable**

Why String object is immutable?
Because of String pooling concept

String pooling
Q) What is the difference in creating String objects *using constructor* and by *assigning literal*?
A) String pooling

String pooling means pooling strings, it means grouping string objects.

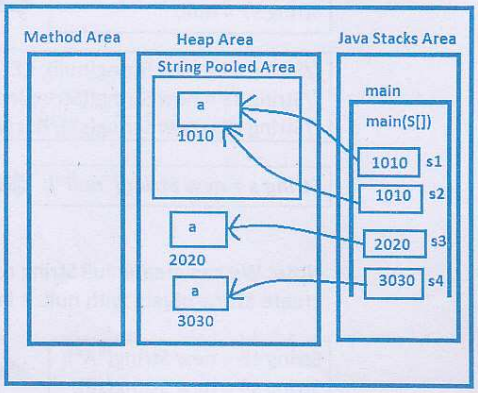
If we create String objects by assigning same string literal, only one object is created and all referenced variables are initialized with that same String object reference. Then all referenced variables are pointing to same String object. This is reason why String object become immutable. It means string data modification is not stored in same memory.

Since a string object is pointing by multiple referenced variables if we change that String value using one referenced variable, all other referenced variables those are pointing to that objects are also affected. To solve this problem SUN decided to create String as immutable.

This String pooling concept is not applicable for the String objects those are created using "*new keyword and constructor*". They have separate object and those string objects are created in heap area directly. The string objects created with expression (+ operator), method returned String objects are also does not come under string pooling. So, all these objects are created in heap area, not in string constant pooled area.

Below diagram shows String Pooling with JVM architecture

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

        String s1 = "a";
        String s2 = "a";

        System.out.println(s1 == s2);          //true
        System.out.println(s1.equals(s2));     //true


        String s3 = new String("a");
        String s4 = new String("a");

        System.out.println(s3 == s4);          //false
        System.out.println(s3.equals(s3)); //true
    }
}
```



Q) Why this String pooling?
A) To improve performance by saving memory.

String expression calculation

If an expression has only String literals that expression is calculated by compiler and places result as literal. So JVM creates that String object in stringpooled area. If expression has a variable compiler does not calculate it, it is calculated by JVM and creates that object in heap area.

Q) In below code how many string objects are created?

```
String s1 = "a";        String s2 = "a";
String s3 = "a";        String s4 = "b";

String s5 = new String("a");
String s6 = new String("a");

String s7 = "ab";
String s8 = s1 + "b";
String s9 = "a"+"b";

System.out.println(s1 == s2);// true
System.out.println(s5 == s6);// false
System.out.println(s7 == s8);// false
System.out.println(s7 == s9);// true
```

A) 6 objects, they are s1, s4, s5, s6, s7, s8

String Common Operations:-

1. Checking String is empty or not
2. Finding length of the String
3. Printing String objects
4. Compare String objects normally
5. Comparing String objects lexicographically
6. Reading character at the given index
7. Finding index of given character
8. Searching for the given String
9. Checking the String starts with
10. Checking the String ends with
11. Retrieving substring
12. String objects concatenation
13. Removing trailing and leading spaces
14. Changing all string characters to lowercase
15. Changing all string characters to uppercase
16. Replacing old character with new character
17. Replacing old String with new String
18. Converting String into character array and byte array
19. Preparing String object from a given character String
20. Split the String into tokens
21. Converting any data type, including primitive types, into String type.

Below program shows using all String class methods

```
class Example{}

class StringOperations {

        public static void main(String[] args) {

                String s = "Java Programming Language";

                //finding string is empty or not
                System.out.println(s.isEmpty());

                //finding length of the string
                System.out.println(s.length());

                String s1 = "";
                String s2 = " ";
                String s3 = new String("");
                System.out.println(s1.isEmpty());
                System.out.println(s2.isEmpty());
                System.out.println(s3.isEmpty());

                System.out.println(s1.length());
                System.out.println(s2.length());
                System.out.println(s3.length());

                //printing string object
                System.out.println(s);

                //comparing two strings
                String s1 = new String("abc");
                String s2 = new String("abc");
                String s3 = new String("Abc");

                System.out.println(s1 == s2);        //false    => different references
                System.out.println(s1.equals(s2)); //true     => both objects has same state

                System.out.println(s2 == s3);        //false => different references
                System.out.println(s2.equals(s3)); //false => both objects has different state

                //comaring strings without case, below method is defined in String class
                //public boolean equalsIgnoreCase(String s)

                String s4 = new String("a");
                String s5 = new String("A");
```

```
System.out.println(s4.equals(s5));              //false => compares with case
System.out.println(s4.equalsIgnoreCase(s5)); //true => compares without case

//comparing strings lexicographically, means after comparison method should
//return difference between string content,
//public int compareTo(String s)              <= compares Strings with case
//public int compareToIgnoreCase(String s)  <= compares Strings without case

String s6 = new String("a");
String s7 = new String("A");
System.out.println(s6.compareTo(s7)); //32              => a - A => 97 - 65   =>32
System.out.println(s6.compareToIgnoreCase(s7));//0  => a - a => 97 - 97   =>0

String s8 = "abcdef";
String s9 = "abc";
System.out.println(s8.compareTo(s9)); //3   =>s8.length() – s9.length() => 3

String s10 = “abcdef”;
System.out.println(s9.compareTo(s10)); //-3 =>s9.length() – s10.length() => -3

String s11 = “adc”;
System.out.println(s11.compareTo(s9)); //2  => d – b => 100 – 98 => 2

//startsWith() / endsWith()
String s = "Java Programming Language";
System.out.println(s.startsWith("Java"));
System.out.println(s.startsWith("java"));
System.out.println(s.startsWith("hari"));

System.out.println(s.endsWith("hari"));
System.out.println(s.endsWith("Language"));

//print character of the given index, string index starts from ZERO, because its
internal object is char array, should use below method.
//public char chatAt(int index)

String s1 = new String("Java Programming Language");
System.out.println("character at 10th index in s1 string: "+ s1.charAt(10));

//print all characters in given string with index, we should write our own logic
with charAt() and length() methods
for (int i = 0; i < size ; i++ ){
        System.out.println("character at index "+ i + "is : "+ s1.charAt(i));
}
```

```
//Note: If we pass index out of range (index < 0 || index >= string.length()) to
charAt() method it leads StringIndexOutOfBoundsException
//In the above for loop if we keep condition i <= size, after printing all 25
characters JVM throws SIOBE for 25th index.

System.out.println(s1.charAt(0)); //=> J
System.out.println(s1.charAt(10)); //=> a
//System.out.println(s1.charAt(-10)); //=> SIOBE
//System.out.println(s1.charAt(26)); //=> SIOBE

//Finding index of given character of String in the given string,
use below methods

//below methods return first occurrence of given character or String
        //public int indexOf(char ch)
        //public int indexOf(String str)

//below methods return last occurrence of given character or String
        //public int lastIndexOf(char ch)
        //public int lastIndexOf(String str)

//below methods return first occurrence of given character or String by
searching it from the "givenIndex" not from ZERO index.
        //public int indexOf(char ch, int fromIndex)
        //public int indexOf(String str, int fromIndex)

//below methods return first occurrence of given character or String by
searching it from the "givenIndex" not from END index.
        //public int lastIndexOf(char ch, int fromIndex)
        //public int lastIndexOf(String str, int fromIndex)

System.out.println(s1.indexOf('a')); //=> 1
System.out.println(s1.lastIndexOf('a')); //=> 22
System.out.println(s1.indexOf('a', 4)); //=> 10
System.out.println(s1.lastIndexOf('a', 22)); //=> 22
System.out.println(s1.lastIndexOf('a', 21)); //=> 18

System.out.println(s1.indexOf("Programming")); //=> 5

//Focus: the given char or string is not available in the current string, above
methods return -1.
System.out.println(s1.indexOf("hari")); //=> -1
System.out.println(s1.indexOf('k')); //=> -1
```

```
//write a program to find the given string or char available or not
//until 1.4 version, we should write a condition using indexOf() method to find
string is available or not
//In 1.5 version, SUN introduced a new method called "contains(String s)" to
find string is available or not
//It returns true if substring is available in current string , else returns false.

//check below methods definition given at end of the program
findWithIndexOf("abcdef");
findWithContains("abcdef");

findWithIndexOf("asdfasdfsadf hari krishna asdfasdfsadf");
findWithContains("asdfasdfsadf hari krishna asdfasdfsadf");

//substring()
String s = "Java Programming Language";
//Note: if (begin index == end index) this method returns empty string

System.out.println(s1.substring(5));
System.out.println(s1.substring(5, 15));
System.out.println(s1.substring(5, 16));
//      System.out.println(s1.substring(-6));
System.out.println(s1.substring(5, 25));
//      System.out.println(s1.substring(5, 26));
System.out.println(s1.substring(24, 24));
System.out.println(s1.substring(24, 25));
System.out.println(s1.substring(23, 24));
System.out.println(s.substring(5,16));
System.out.println(s.substring(s.indexOf("P"),s.indexOf(" L")));
System.out.println(s.substring(s.indexOf("Language")));
System.out.println(s.substring(16,5));
System.out.println(s.substring(-1,5));

//concatenating new string,
//adding given string characters at end of current string characters and placing
//the result in new object is called concatenation
//We can do it in two ways
        //1. using + operator
        //2. using concat() method
                //public String concat(String newString)

//concatenation using + operator
String s1 = "a";
String s2 = s1 + "b";
```

```
System.out.println(s1);//a
System.out.println(s2);//ab
System.out.println(s1 == s2); //false

//concatenation using concat() method
String s3 = "a";
String s4 = s3.concat("b");

System.out.println(s3);//a
System.out.println(s4);//ab
System.out.println(s3 == s4); //false

//String limitation
//If we perform modification on string using String class methods, and if that
object is changed as a result of that method call, new String object is created
with that result. If we store that new object reference in reference variable, it is
reachable else it is garbage collected.

//In the below statement string is modified => so new String object is created =>
but its reference is not stored => hence s4 state is still ab only

s4.concat("c");
System.out.println(s4);//ab

// Due to a String method call, if String object is not changed, then the current
//String object reference is returned

//What is the output from the below program?
String s5 = s4.concat("");
System.out.println(s4); // ab
System.out.println(s5); // ab
System.out.println(s4 == s5); //true

Note: '+' operator always creates and returns new string object irrespective of
the object is changed or not.

String s6 = "a";
String s7 = s6 + "";
System.out.println(s6 == s7); //false

//converting all characters in string to upper or lower case
//public String toUpperCase()
//public String toLowerCase()

String s1 = "Hari Krishna";
```

```
System.out.println(s1.toUpperCase()); //HARI KRISHNA
System.out.println(s1.toLowerCase()); //hari krishna
System.out.println(s1); //Hari Krishna

String s1 = "hari";
String s2 = s1.toLowerCase();
String s3 = s1.toUpperCase();
System.out.println(s1); //hari
System.out.println(s2); //hari
System.out.println(s3); //HARI
System.out.println(s1 == s2); //true
System.out.println(s1 == s3); //false

//replacing a character with new character, use below methods of String class
//public String replace(char oldChar, char newChar)
//public String replace(CharSequence oldString, CharSequence newString)
//public String replaceAll(String oldString, String newString)
//public String replaceFirst(String oldString, String newString)

String s1 = "Java Programming Language";
String s2 = s1.replace('J', 'K');

System.out.println("s1 String: "+s1);
System.out.println("s2 String: "+s2);

String s3 = s1.replace('a', 'A');
System.out.println("s3 String: "+s3);

String s4 = s1.replace("Programming", "Object-Oriented Programming");
System.out.println("s4 String: "+s4);

String s5 = "Ha Ha Ha";
String s6 = s5.replace("Ha", "Hello");
System.out.println("s5 String: "+s5);
System.out.println("s6 String: "+s6);

String s7 = s6.replace("hello", "Hi");
System.out.println("s7 String: "+s7);

//due to replace method call if there is no change in current string, it returns
same current string object reference.
System.out.println(s6 == s7);//true
```

```
        //deleting trailing and leading spaces of a String
        //public String trim()  => it will not remove middle spaces

        String s1 = new String(" hari krishna ");
        String s2 = s1.trim();
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s1 == s2);

        System.out.println(s1.length());
        System.out.println(s2.length());

        //What is the output from below program?
        String s3 = new String("Naresh Technologies");
        String s4 = s3.trim();

        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s3 == s4);

        //split()
        String[] sarray = s.split(" ");
        int size = sarray.length;
        for (int i = 0; i < size ; i++ ){
                System.out.println(i + " tocken is "+ sarray[i]);
        }
    }
    static void findWithIndexOf(String originalString){

        if(originalString.indexOf("hari") != -1){
                System.out.println("The String hari is available ");
        }
        else{
                System.out.println("The String hari is not available ");
        }
    }
    static void findWithContains(String originalString){
        if(originalString.contains("hari")){
                System.out.println("The String hari is available ");
        }
        else{
                System.out.println("The String hari is not available ");
        }
    }
}
```

Q) How can we remove middle spaces?
A) We must use replace method. Remove not middle spaces also begin and end spaces.
For example:

```
String s4 = " abc abc  ";
String s5 = s4.replace( " ", "" );
System.out.println(s4);
System.out.println(s5);
System.out.println(s4.length());
System.out.println(s5.length());
```

Assignment

Write a program to reverse the words in the String objects
InputString: How are you?
Ouput String: you? are How

```
//StringWordsReverse.java
public class StringWordsReverse{

        public static String reverseStringWords(String s){

                String[] stringWords = s.split(" ");

                String resultString = "";

                for (int i = stringWords.length-1; i >= 0; i--){

                        resultString += stringWords[i] + " ";

                }

                return resultString;
        }
}
```

Just you count how many objects are created in the above program.

It leads to big performance issue. we should use *StringBuffer* or *StringBuilder* class to import performance by reducing memory usage.

```
//Test.java
import java.util.*;

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

                Scanner s = new Scanner(System.in);
                System.out.print("Enter string: ");

                String str = s.nextLine();
                String result = StringWordsReverse.reverseStringWords (str);

                System.out.println("original string: "+str);
                System.out.println("reverse string: "+result);
        }
}
```

The **StringBuffer** class

Definition
It is a thread-safe, mutable sequence of characters. A string buffer is like a String, but can be modified in the same memory location.

How StringBuffer object can be created?
StringBuffer object can be created only in one way
1. By using its available constructor

In StringBuffer class we have below 4 constructors

1. **public StringBuffer()**
   Creates empty StringBuffer object with default capacity 16 buffers, it means it has 16 empty locations

   For example:
   ```
   StringBuffer sb = new StringBuffer();
   System.out.println("sb: "+sb); //sb:
   System.out.println("sb1 capacity: "+sb1.capacity()); //16
   ```

2. **public StringBuffer(int capacity)**
   Creates empty StringBuffer object with the given capacity

   For example:
   ```
   StringBuffer sb2 = new StringBuffer(3);
   System.out.println("sb2: "+sb2);//sb2:
   System.out.println("sb2 capacity: "+sb2.capacity());//3
   ```

   Rule: capacity value should be >=0, if we pass negative number, JVM throws "*java.lang.NegativeArraySizeException*"
   ```
   StringBuffer sb = new StringBuffer(-5);   RE: NASE
   ```

3. **public StringBuffer(String s)**
   Creates StringBuffer object with given String object data. It performs string copy from String to StringBuffer object. The default capcity is [16 + s.length()]

   For example:
   ```
   StringBuffer sb3 = new StringBuffer("abc");
   System.out.println("sb3: "+sb3); //sb3: abc
   System.out.println("sb3 capacity: "+sb3.capacity()); //19
   ```

4. **public StringBuffer(CharSequence cs)**
   Creating new StringBuffer object with the give CS object characters, it also performs string copy from CS object to StringBuffer object. Its capacity is [16 + cs.length()]

For example:

```
StringBuffer sb4 = new StringBuffer("abc");
StringBuffer sb5 = new StringBuffer(sb4);

System.out.println("sb4: "+sb4); //abc
System.out.println("sb5: "+sb5); //abc
System.out.println("sb4 capacity: "+sb4.capacity()); //19
System.out.println("sb5 capacity: "+sb5.capacity()); //19

System.out.println(sb4 == sb5); //false
```

Rule:
We cannot create SB object by passing null. It leads to RE: NPE
In StringBuffer case CE: ambiguous error is not raised for null argument because its constructors are overloaded with super and subclasses CharSequence and String.

For null argument String parameter constructor is called.

```
StringBuffer sb = new StringBuffer(null); RE: NPE
```

StringBuilder constructors
StringBuilder class functionality is exactly same as StringBuffer class. So it also has same StringBuffer parameter constructors as shown below

1. public StringBuilder()
2. public StringBuilder(int capacity)
3. public StringBuilder(String s)
4. public StringBuilder(CharSequence cs)

Q) What is the difference between StringBuffer and StringBuilder?
A) StringBuffer is synchronized and where as StringBuilder is not synchronized

StringBuilder class is given in Java 5 version to develop non synchronized StringBuffer object. Except this difference StringBuffer and StringBuilder operations are exactly same.

So,

- StringBuffer object must be used in Multithread programming model applications
- StringBuilder object must be used in Single Thread programming model applications

Find out valid objects creation in the below list

```
String s1 = "abc";
String s2 = new String("abc");

StringBuffer sb1 = new StringBuffer("abc");
StringBuffer sb2 = "abc";
```

Special operations those we can perform only on StringBuffer and StringBuilder classes

Since StringBuffer is mutable we can perform below 5 operations specially on StringBuffer

1. Append
2. Insert
3. Delete
4. Reverse
5. Replacing character at given index

In StringBuffer class, we have below methods to perform above 5 operations

1. public StringBuffer ***append***(xxx data)
2. public StringBuffer ***insert***(int index, xxx data)
   - where xxx is java data types
   - above methods are overloaded to take all possible java datatype values
3. public StringBuffer ***deleteCharAt***(int index)
   public StringBuffer ***delete***(int start, int end)
4. public StringBuffer ***reverse***()
5. public StringBuffer ***setCharAt***(int index, char ch)

All above methods *return current StringBuffer object reference* after modification is completed

Rule: index value must be with range of ["0" to sb.length() -1], else it leads to "*java.lang.StringIndexOutOfBoundsException*"

Below program shows performing all above operations

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

        //append
        StringBuffer sb1  = new StringBuffer("abc");
        StringBuffer sb2  = sb1.append("d");

        System.out.println(sb1);//abcd
        System.out.println(sb2);//abcd
        System.out.println(sb1 == sb2);//true

        //insert
        StringBuffer sb3 = new StringBuffer("2942011");

        System.out.println(sb3);// 2942011

        sb3.insert(2, '/');
        System.out.println(sb3);// 29/42011

        sb3.insert(4, '/');
        System.out.println(sb3);// 29/4/2011
```

```
        //delete
        StringBuffer sb4 = new StringBuffer("Hari xyz Krishna");
        System.out.println(sb4);//Hari xyz Krishna

        sb4.delete(sb4.indexOf("xyz") , sb4.indexOf("xyz") + 3);
        //sb.delete(5, 5 +3 );   //sb.delete(5, 8);
        System.out.println(sb4); //Hari  Krishna

        sb4.deleteCharAt(4);
        sb4.deleteCharAt(4);
        System.out.println(sb4); //HariKrishna

        //reverse
        StringBuffer sb5 = new StringBuffer("abc");
        System.out.println(sb5); //abc

        sb5.reverse();
        System.out.println(sb5); //cba

        //capacity() and length()
        //public int capacity()
        //public int length()

        StringBuffer sb6 = new StringBuffer();
        System.out.println(sb6);//
        System.out.println(sb6.capacity());//16
        System.out.println(sb6.length());//0

        sb6.insert(0,"abc");
        System.out.println(sb6);//abc
        System.out.println(sb6.capacity());//16
        System.out.println(sb6.length());//3

        StringBuffer sb7 = new StringBuffer("abc");
        System.out.println(sb7);//abc
        System.out.println(sb7.capacity());//19
        System.out.println(sb7.length());//3
    }
}
```

Assignment

Write a program to perform append, and insert operations

Take a string "object language". Change this string as "Object-Oriented Programming Language is secure"

Controlling StringBuffer capacity

We have below three methods to control StringBuffer capacity

1. public void ensureCapacity(int minimumCapacity)
2. public void trimToSize()
3. public void setLength(int newLength)

enusureCapacity

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 non positive, this method takes no action and simply returns, the capacity remains the same current capacity.

What is the output from below program?

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());     //16

sb.ensureCapacity(17);                 //16*2+2 => 34 => (34 > 17)
System.out.println(sb.capacity());     //34

sb.ensureCapacity(100);                //34*2+2 => 70 => (70 < 100)
System.out.println(sb.capacity());     //100

sb.ensureCapacity(-3);
System.out.println(sb.capacity());     //100
```

Q) What is the capacity of length of SB object in the below program?

```
StringBuffer sb = new StringBuffer();

for (int i = 1; i<= 17 ; i++){
        sb.append(i);
}

System.out.println(sb.capacity());
System.out.println(sb.length());
```

trimToSize
It reduces SB capacity to its size. The prototype is: *public void trimToSize()*

What is the output from the below program?

```
StringBuffer sb1 = new StringBuffer("abc");
System.out.println(sb1.capacity());//19
System.out.println(sb1.length());//3

sb1.trimToSize();

System.out.println(sb1.capacity());//3
System.out.println(sb1.length());//3
```

setLength
Sets the length of the SB to current given length. Its prototype is:

```
public void setLength(int newLength)
```

Rule: The newLength argument must be greater than or equal to 0.
Else this method throws "*java.lang.IndexOutOfBoundsException*".

Case #1: If the passed length is greater than current length, it places empty spaces
What is the output from below program?

```
StringBuffer sb2 = new StringBuffer("abcdefg");
System.out.println(sb2.capacity()); //23
System.out.println(sb2.length()); //7
System.out.print(sb2); //adbcdefg
System.out.println("hari"); //adbcdefghari

sb2.setLength(9);
System.out.println(sb2.capacity()); //23
System.out.println(sb2.length()); //9
System.out.print(sb2); //adbcdefg
System.out.println("hari"); //adbcdefg  hari
```

Case # 2: If the passed length is less than current length,
it removes all characters > this length

What is the output from below program?

```
sb2.setLength(4);
System.out.println(sb2.capacity()); //23
System.out.println(sb2.length()); //4
System.out.print(sb2); //abcd
```

Diff #2: Creating String, StringBuffer objects
String object can be created in two ways
1. By assigning string literal => String s1 = "a";
2. By using its available constructors

StringBuffer objects are creating only in one way
1. By using its available constructor.

Diff #3: Capacity
String does not have default capacity, the passed string length is its capacity.

StringBuffer has default capacity 16 buffers. It increases its capacity automatically when size reached its capacity: formula is: (current capacity * 2) + 2.
To control string buffer capacity or to create string buffer with user given capacity we have
1. Constructor - to assign capacity at the of creation
2. method - to change its capacity after its creation

Diff #4: Concatenation
Using String object, we can concat new string to the current string in two ways
1. using + operator
2. using concat() method

Using StringBuffer we can perform concat operation only in one way
1. using append() method

Important point to be remembered
Due to concatenation operation by using concat() method, if current string object is modified, this method creates new object with this result and returns that object reference. If it is not modified it returns the same current object reference. But when we use + operator whether current string object is modified or not modified, JVM always creates new String object.

Where as the append() always stores modification result in the current SB object and returns the current SB object

What is the output from the below program?

```
String s1 = new String("a");
String s2 = s1.concat("b");
System.out.println(s1);
System.out.println(s2);
System.out.println(s1 == s2);
s2.concat("c");
System.out.println(s2);
```

```
StringBuffer sb1 = new StringBuffer("a");
StringBuffer sb2 = sb1.append("b");
System.out.println(sb1);
System.out.println(sb2);
System.out.println(sb1 == sb2);
sb2.append("c");
System.out.println(sb2);
```

Diff #5: Special operations those we can only perform on StringBuffer,

1. append
2. insert
3. delete
4. reverse

Since String is immutable we cannot perform these operations on String.

Diff #6: Comparison

Equals method is overridden in String class to compare String objects with state So string objects are compared with reference by using == operator and with sate by using equals() method..

In StringBuffer and StringBuilder classes equals() method is not overridden, so using == operator and equals() method their objects are compared with reference.

In all three class toString() method is overridden to print the object state.

What is the output from below program?

```
String s1 = "a";
String s2 = "a";
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));

String s3 = new String("a");
String s4 = new String("a");

System.out.println(s3 == s4);
System.out.println(s3.equals(s4));

StringBuffer sb1 = new StringBuffer("a");
StringBuffer sb2 = new StringBuffer("a");

System.out.println(sb1 == sb2);
System.out.println(sb1.equals(sb2));
```

How many String objects are created in the below program?

```
String s1 = “a”;
String s2 = “a”;
String s3 = “b”;

String s4 = “ab”;
String s5 = “a” + “b”;
String s6 = s1 + “b”;
String s7 = s1 + s2;
```

Q) What is the best design in defining a method to take and return string data as argument?
A) The parameter and return type should be java.lang.String, not StringBuffer

The problem if we take StringBuffer as parameter or return type is: user should convert that string data as StringBuffer object, because StringBuffer will not accept string data directly.

For Example:

```
void m1(String s){

}

void m1(StringBuffer sb){

}
```

```
m1("abc");

m1(new StringBuffer("abc"));
//It is not the convenient way for user.
```

Project development procedure:

- In projects we store string data by using java.lang.String object.
- To perform modifications on that string data we convert/copy that string into StringBuilder object.
- After modification we convert/copy back the result string to String object and return the result.

Q) What are the different ways for converting String to StringBuffer and vice versa.

- Using *StringBuffer(String)* constructor we can convert String to StringBuffer.
- Using *String(StringBuffer)* constructor and *StringBuffer.toString()* we can convert StringBuffer to String.

Write a program to reverse the string words?
Input: How Are You.
Output: You Are How

```
class  StringWordsReverse{

        public static String reverseStringWords(String s){

                String[]        stringWords    = s.split(" ");
                int             noOfWords      = stringWords.length;

                StringBuilder resultBuffer = new StringBuilder ();

                for (int i = noOfWords -1; i >= 0 ; i--){
                        resultBuffer.append(stringWords[i]);
                        resultBuffer.append(" ");
                }

                //removing last space character
                return resultBuffer.toString().trim();
        }

        public static void main(String[] args){
                System.out.print(reverseStringWords("How Are You"));
        }
}
```

Immutable objects and their creation

Immutable objects are simply objects whose state (the object's data) cannot be changed after construction. Examples of immutable objects from the JDK include String and Integer. Immutable objects greatly simplify your program, since they:

- are simple to construct, test, and use
- are automatically thread-safe and have no synchronization issues
- do not need a copy constructor
- do not need an implementation of clone
- allow hashCode to use lazy initialization, and to cache its return value
- do not need to be copied defensively when used as a field
- make good Map keys and Set elements (these objects must not change state while in the collection)
- have their class invariant established once upon construction, and it never needs to be checked again
- always have "failure atomicity" (a term used by Joshua Bloch) : if an immutable object throws an exception, it's never left in an undesirable or indeterminate state

Immutable objects have a very compelling list of positive qualities. Without question, they are among the simplest and most robust kinds of classes you can possibly build. When you create immutable classes, entire categories of problems simply disappear.

Make a class immutable by following these guidelines :

- *Make the class final, or use static factories and keep constructors private* to ensure the class cannot be overridden
- *Make fields private and final* to ensure the variable is not accessible from outside of class and also to ensure it is not modified unknowingly in the same class from its mutator methods.
- Define parameterized constructor to initialize the state of the object to ensure callers are constructing an object completely in a single step, instead of using a no-argument constructor combined with subsequent calls to mutator methods
- Do not provide mutator methods not to allow changes *or*, if we want to allow changes define mutator method but store the result in new object and return it.

For example

- String is an immutable object with mutator methods that is it allows changes on store string data but result is stored in new String object and returns that object
- Wrapper classes -they are - Integer, Character, Boolean, etc are immutable object without mutator methods that is they do not allow changes even new object.

Chapter 28

IO Streams

- In this chapter, You will learn
  - Storing and reading binary and text data from a flat file
  - Storing and reading Primitive Data from a flat file
  - Storing and reading objects from a flat file
    - Serialization and deserialization
  - Different ways to create objects in Java
  - Logical concatenation of streams
  - Default streams created in JVM
  - Importance of PrintStream class
  - Clear understanding on System.out.println() statement
  - Reading data from key board
  - Introduction to reader and writer classes
  - Importance of BufferedReader class
  - Creating and deleting files & directories from a program
- By the end of this chapter- you will be in a position to perform file based persistence operations.

Interview Questions

By the end of this chapter you answer all below interview questions

- Need of this chapter
- Define Stream
- Types of Streams
- Stream creation in Java

[Binary Streams]

- InputStream, OutputStream class hierarchy
- InputStream and OutputStream class methods
- Reading, writing, copying file data using FileInputStream and FileOutputStream
- Reading and writing data as primitive types using DataInputStream and DataOutputStream
- Reading and writing object using ObjectInputStream and ObjectOutputStream
- Understating Serialization and De-serialization
- Use of serialVersionUID variable, transient keyword in Serialization and Deserialization
- Serialization and Deserialization with IS-A and HAS-A relation objects
- Use of SequenceInputStream, BufferedInputStream, BufferedOutputStream
- Understating PrintStream class
- Default streams created in JVM
- Understanding System.out.println() method call structure
- How can we redirect STDOUTPUT data to a file using Sopln().
- Need of File class

[Character Streams]

- Need of Character Streams
- Reader and Writer classes hierarchy
- Reader and Writer classes methods
- Need of FileReader and FileWriter
- Need of InputStreamReader, and OutputStreamWriter
- Need of BufferedReader, and BufferedWriter
- How can we read one line at a time either from keyboard or from a file?
- How can we read input from end-user

So far you have learnt how to store data in variables. In this chapter you will learn how to store the data in files permanently.

Introduction – Need of this chapter

The basic idea of IOStreams chapter is

- storing and reading data from files and
- also reading data from keyboard

In this chapter, we cover the methods required for handling files and directories and also the methods for writing data to different destinations and reading data from different sources.

This chapter also shows you the *object serialization* mechanism that lets you store objects as easily as you can store text or numeric data in files.

First let us understand some basic terminology

Persistence media
The environment that allows us to store data permanently is called persistent media. We can store data permanently in three places.

- File
- Database
- Remote Computer (Socket)

Persistence
The process of storing data permanently in a persistence media is called persistence.

Persistence Logic
The logic that persist data in a persistence media is called persistence logic.
Ex: IOStreams based logic, JDBC based logic, Networking based logic

Persistence Technologies
The technology that provides API to develop persistence logic is called persistence technology.

Well know persistence technologies are
IOStreams – to persist data in flat files
JDBC, EJB, Hibernate – to persist data in DB
Networking – to persist data in remote computer.

In real-time projects the basic operation that we do using this chapter, is to perform persisting the data in files. Persisting the data means storing the data permanently.

Where can we store data or result permanently?
In persistence medias either in files or in Databases.

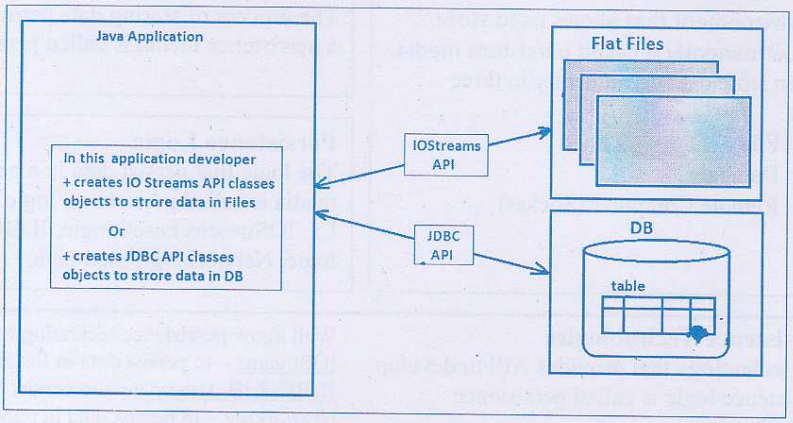
Storing data in variables and arrays is temporary. Data will be lost when a local variable goes out of scope or when the program terminates.

Programmers use files or DBs for long-term storage of large amounts of data. It is available even after termination of the program. We refer to data maintained on files as persistent data, because the data exists beyond the duration of program execution.

To store data in files and DBs SUN has given in-built API. We all need to do is creating the particular class object and calling methods on that object for storing and reading data from that persistence media.

IOStreams API is given to store and read data from files
JDBC API is given to store and read data from DBs.

Check below diagram



In this chapter we will learn storing data in files using IOStreams API.
In Advanced Java in JDBC chapter we will learn storing data in DB using JDBC API.

Note: In either of the approaches we just create object for predefined classes and we will call appropriate methods to store and read data from files and DBs.

So we can conclude IOStreams API and JDBC API is given not for developing Business logic rather they have been given to develop persistence logic.

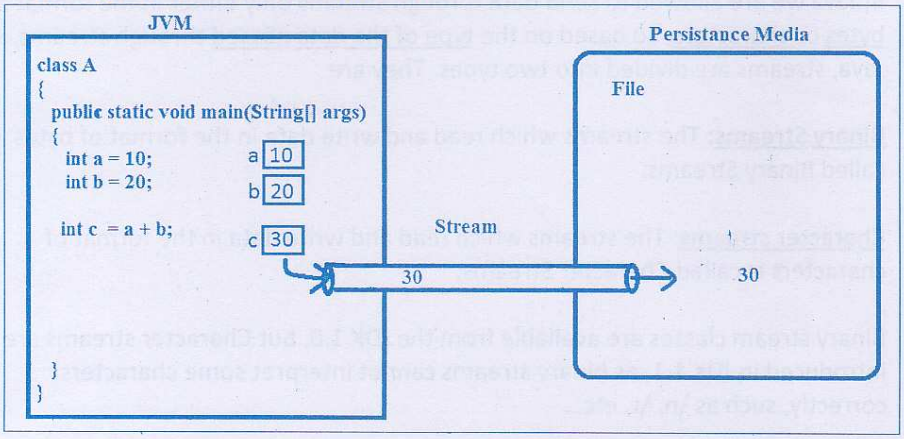
How a Java Application can store or read data from a file?
Using **stream** object.

Introduction to Streams
Stream is a logical connection between Java program and a file.

To store the data in the persistent media there should be a way to connect to persistent media from java application either physically or logically. Stream provides logical connection.

Stream can be defined as "*it is a continuous flow of data* between java program and persistence media".

Below diagram shows stream architecture



This diagram shows storing the result 30 in a file via a stream connection. In this chapter we learn how to create this stream object.

What is the direction of the stream flow?
Stream has a direction and its direction depends on the viewer's view. In Java the viewer is Java Application. If you look from Java application data is sending out from Java application.

Types of streams

Generally streams are divided into two types based on data flow direction.

InputStream: the stream that allows data to *come into* the java application from the persistent media is called Input Stream.

OutputStream: the stream that allows data to *send out* from the java application to be stored into the persistent media is called Output Stream.

Basically *InputStreams* are used to *read* data from a persistence media, and *OutputStreams* are used to *write or store* data in a persistence media from a java application.

Types of Java streams

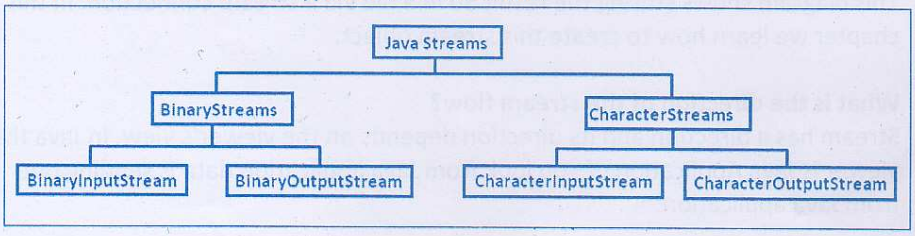
In Java we are allowed to send data through streams only either in the format of bytes or characters. So based on the type of the data passed through streams, in Java, streams are divided into two types. They are

Binary Streams: The streams which read and write data in the format of bytes is called Binary Streams.

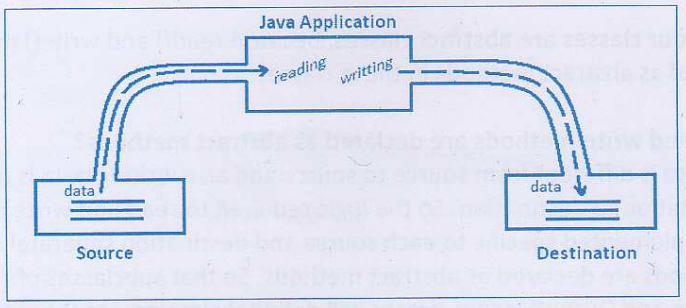
Character streams: The streams which read and write data in the format of characters is called Character Streams.

Binary stream classes are available from the JDK 1.0, but Character streams are introduced in JDK 1.1, as binary streams cannot interpret some characters correctly, such as \n, \t, etc...

Below diagram shows the types of JAVA streams.



What are the different sources and destinations to read and write data?
The data can be read from and written into different sources and destinations. From sources Java application reads data, and it writes data to destinations.



Below are the commonly available sources and destinations:
The **sources** can be a Keyboard, Mouse, File, Database, Socket (Computer), object, Array, String, StringBuffer.

The **destinations** can be a Monitor, File, Database, Socket (Computer), object, Array, String, StringBuffer.

How can we create stream in Java specific to source and destination?
In **java.io** package we have several classes to create input or output streams specific to source and destination.

So the technical definition of stream would be "Stream is a java object that allows reading and writing data to a persistence media".

What is the super class for all types of input and output stream classes?
InputStream is the super class for all *binary input stream* classes.
OutputStream is the super class for all *binary output stream* classes.

Reader is the super class for all *character input stream* classes.
Writer is the super class for all *character input stream* classes.

InputStream and Reader classes has **read()** method to read data from a source. In InputStream class it returns one byte at a time and in Reader class it returns one character at a time.

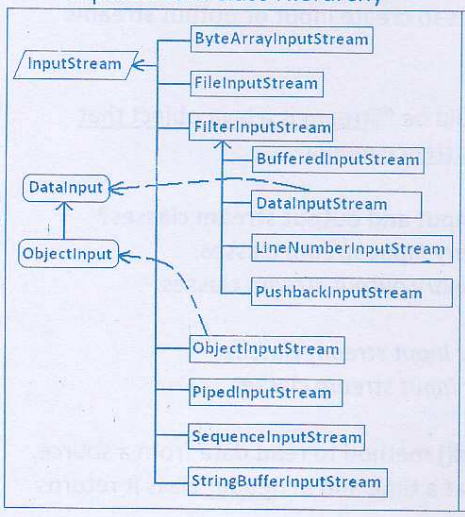
OutputStream and Writer classes has **write()** method to write data to a destination. In OutputStream class it writes one byte at a time and in writer class it writes one character at a time.

All above four classes are abstract classes, because read() and write() methods are declared as abstract methods in these classes.

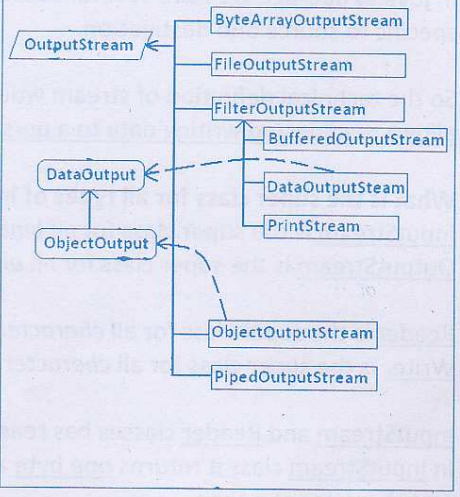
Why read and write methods are declared as abstract methods?
Reading data is different from source to source and also writing data is different from destination to destination. So the logic required to read and write data must be implemented specific to each source and destination separately. Hence these methods are declared as abstract methods. So that subclasses of InputStream and OutputStream classes will definitely implement these two methods.

InputStream and OutputStream class hierarchy:
For each type of source and destination in java.io package sun has given a separate class. Below diagram shows InputStream and OutputStream classes' hierarchy

InputStream class Hierarchy



OutputStream class Hierarchy



Explanation on above important classes

FileInputStream & FileOutputStream classes are used to read and write data as bytes from File. Basically these two streams are used as basic data source and destination for other streams.

DataInputStream & DataOutputStream classes are used to read and write data as primitive data. Basically these two streams are used to add capability to another input stream and output stream to read and write data as primitive types.

ObjectInputStream & ObjectOutputStream classes are used to read and write data as Object. Basically these two streams perform object Serialization and De-serialization.

BufferedInputStream & BufferedOutputStream classes are used to read and write data as buffers. Basically these two streams are used to improve reading and writing performance of other streams.

SequenceInputStream class is used to read data from multiple InputStreams sequentially.

PrintStream class is the most continent output stream class to write ta.

What is the procedure to perform read and write operations using streams?
Below is the basic procedure to perform IO operations

- Create stream class object based on source and destination
- call *read()* method to read data from the source
- call *write()* method to write data to the destination.

For instance, for performing IO operations on File we must create FileInputStream, FileOutputStream or FileReader, FileWriter stream classes object.

InputStream class methods:

Since InputStream is the super class for all binary input stream classes, it has below method with general implementation for all subclasses. As I said before no-arg read method is abstract method, it is implemented in its all subclasses.

InputStream class methods

Book Reading	InputStream Reading
Buy a book	Create stream object using its available constructors
1. checking available pages	Checking avaialbe bytes to read `public int available() throws IOException`
2. reading word by word	reading byte - by - byte `public abstract int read() throws IOException`
3. reading all words	reading all bytes `public int read(byte[] b) throws IOException`
4. reading important words	reading selected range of bytes `public int read(byte[] b, int off, int len) throws IOException`
5. Checking marksupported or not	checking is this stream supports marking `public boolean markSupported()`
6. marking important lines	marking current position in stream `public void mark(int readLimit)`
7. re-reading marking lines	placing the control at marked place `public void reset() throws IOException`
8. reading only required pages	skip reading given range of bytes `public long skip(long n) throws IOException`
9. closing the book	closing the stream `public void close() throws IOException`

What is the difference in read() and read(byte[]) methods return value?

read() method returns the actual byte value, where as read(byte[]) method returns number of bytes read from the stream into byte array.

OutputStream class methods:

Since OutputStream is the super class for all binary output stream classes, it has below method with general implementation for all subclasses. As I said before "int parameter" write method is abstract method, it is implemented in its all subclasses.

OutputStream class Methods

storing data in our mind	writing data to an output stream
writing data word by word	writing one by at time to an outputstream public abstract void write(int i) throws IOException
writing all words	writing stream of bytes to an outputstream public void write(byte[] b) throws IOException
writing important words	writing give range of stream of bytes to an outputstream public void write(byte[] b, int off, int len) throws IOException
store all words in permanent mind	flush all bytes to destination from outputstream public void flush() throws IOException
close mind	close outputstream public void close() throws IOException

Note: flush() method must be called for BufferedOutputStream, and BufferedWriter. Else data will not be stored in destination. For all other streams calling flush method is optional. But it is good practice if we call flush() before stream close.

Let us start developing applications for reading and writing data from file.

Application #1: Reading the data from a file
FileInputStream class must be used to read data from a file.

It is two steps process.

1. Create FileInputStream class object by using its available constructors
2. Then call read() method on this object till control reach end of the file. Means read() method must be called in loop, because it returns only one byte at a time. Hence it must be called in loop for each byte available in the file till it returns "-1". It returns -1 if control reached end of the file.

FileInputStream class has below **constructors** to create its Object.

1. public FileInputStream(String name) throws FileNotFoundException.
2. public FileInputStream(File file) throws FileNotFoundException
3. public FileInputStream(FileDescriptor fd)

Rule: To create FileInputStream class object the file must be existed with the passed argument name, else this constructor throws **java.io.FileNotFoundException.**

FileInputStream class constructor throws this exception in below situations

- If the file is not existed with the passed name.
- Passed file name is a directory rather than is a regular file.
- If file does not have reading permissions.

Below program shows reading data from a file
First create a file with the name **test.txt** and store data **abcd** in this file.

Below **is the sample code** to read data from this file

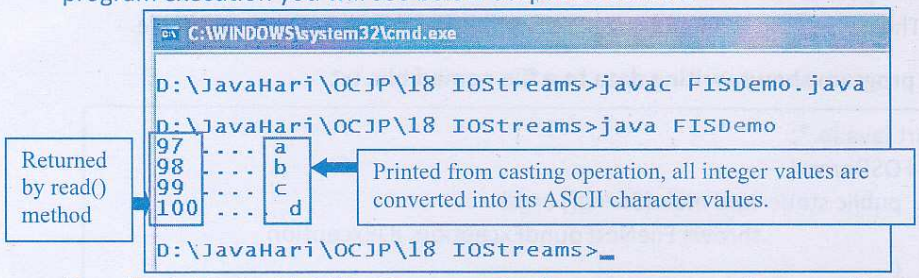
```
//1. Creating stream object connecting to test.txt file
 FileInputStream fis = new FileInputStream("test.txt");

//2. Reading all bytes from the file till control reaches end of the file.
      int i = fis.read() ; <- this statement reads only first character "a".
printing returned character
      System.out.println(i); <- It prints 97, 'a' ASCII integer number.

Converting it to character to print 'a'
      System.out.println((char)i); <- now it prints 'a'
```

Below is the complete program to read complete data from abc.txt file.

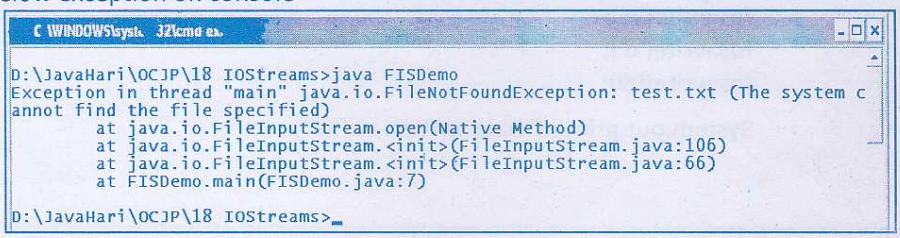
```
import java.io.*;

class FISDemo
{
        public static void main(String[] args) throws FileNotFoundException, IOException
        {
                FileInputStream fis = new FileInputStream("test.txt");

                int i;
                while( ( i = fis.read() ) != -1)
                {
                        System.out.println(i + " .... ");
                        System.out.print((char)i);

                }
        }
}
```

Find the difference in executing above program with and without file

Case #1: Assume **test.txt** file is available with the data **abcd**. After the above program execution you will see below output on console



Case #2: delete **test.txt** file, then execute above program again; you will see below exception on console



Application #2: Writing data to a file
FileOutputStream class must be used to write data to a file.

It is also two steps process

1. Create FileOutputStream object by using its available constructors
2. Then call write() method to write either single or multiple bytes.

FileOutputStream class has below **constructors** to create its Object.

1. public FileOutputStream(String name) throws FileNotFoundException
2. public FileOutputStream(File name) throws FileNotFoundException.
3. public FileOutputStream(String name, boolean append) throws FileNotFoundException.
4. public FileOutputStream(File file, boolean append) throws FileNotFoundException

Note*:** Unlike FileInputStream, FileOutputStream class constructor ***does not throw FileNotFoundException if file is not existed, instead it creates empty file with the given name. Then it writes the given data to that file.

It throws above exception in the below cases.

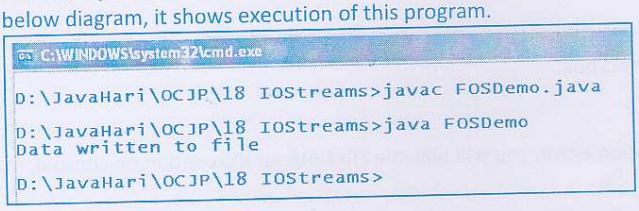
- File doesn't exist but can't be created.
- The passed file name is a directory rather than a regular file.
- The passed file is a Read Only File, writing permissions aren't available.

Below program shows writing data to a file named bbc.txt.

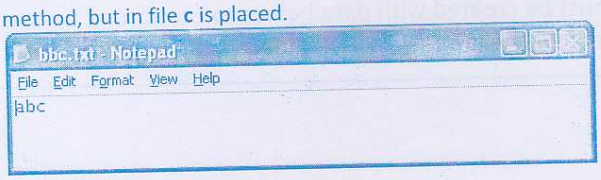
```
import java.io.*;
class FOSDemo {
        public static void main(String[] args)
                        throws FileNotFoundException, IOException
        {
                FileOutputStream fos = new FileOutputStream("bbc.txt");

                fos.write('a');
                fos.write('b');
                fos.write(99);

                System.out.println("Data written to file");
        }
}
```

Follow below test cases to run this application:

Case #1: execute above application without creating **bbc.txt** file. It is executed fine without FNFE exception. FileOutputStream class creates this file when its object is created. Check below diagram, it shows execution of this program.

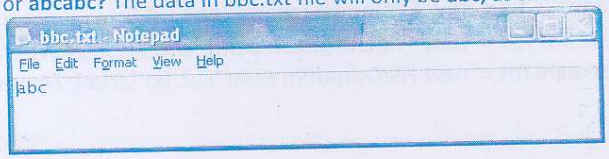


Below is the bbc.txt file with its data. Notice its data, in the application we passed **99** to **write()** method, but in file **c** is placed.



So we can conclude that **write()** method always writes **ASCII character data** and **read()** method always returns **ASCII integer data**.

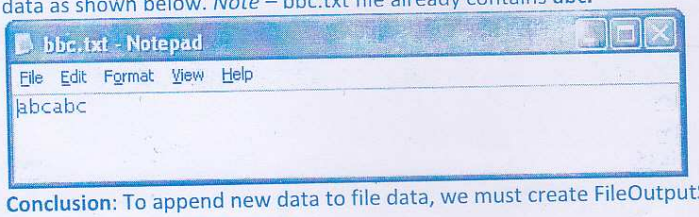
Case #2: Execute this application again, and observe bbc.txt file data. Find out whether it is **abc** or **abcabc?** The data in bbc.txt file will only be **abc,** as shown below



Actually FileOutputStream ***by default overrides the file data*** with new data.

Case #3:
In the above program, replace **FileOutputStream fos = new FileOutStream("bbc.txt");** with
FileOutputStream fos = new FileOutStream("bbc.txt", true);

Then compile and execute the program again. Now observe bbc.txt file, you can find **abcabc** data as shown below. *Note* – bbc.txt file already contains **abc.**



Conclusion: To append new data to file data, we must create FileOutputStream class object with second argument as **true**.

Case #4: Change bbc.txt file permission to read-only.
Setting read-only permission:

1. right click on file
2. click properties
3. Check *Read –only* checkbox.
4. Click Ok

Now execute above application again, you will observe FileNotFoundException on console.

Application #3: File copy
Source file: **abc.txt** file it must be created with data before execution.
Destination file: **cbc.txt** file it will be created by FOS object.

```
import java.io.*;

public class FileCopy
{
	public static void main(String[] args)
		throws FileNotFoundException, IOException
	{
		FileInputStream	fis = new FileInputStream("abc.txt");
		FileOutputStream fos =  new FileOutputStream("cbc.txt"); //overrides
		//FileOutputStream fos =  new FileOutputStream("cbc.txt",true); //appends

		int i;
		while( ( i = fis.read() ) != -1)
		{
			fos.write(i);
		}

		System.out.println("Data has written");
	}
}
```

Below application shows copying file data by reading filenames at runtime. It also shows handling exception properly.

```
//FileCopy.java
import java.io.*;
public class FileCopy
{
        //This method has reusable code, so should not handle exception.
        public static void copyFile(String srcFile, String destFile)
                                throws FileNotFoundException, IOException
        {
                FileInputStream     fis = new FileInputStream(srcFile);
                FileOutputStream    fos =  new FileOutputStream(destFile);

                int i;
                while( ( i = fis.read() ) != -1)
                {
                        fos.write(i);
                }

                System.out.println("Data has written to "+destFile);
        }
}

//It is user application, so it must handle exceptions to print user understandable exception messages
//Test.java
class Test{
        public static void main(String[] args) {
                try{
                        FileCopy.copyFile(args[0], args[1]);
                }
                catch(ArrayIndexOutOfBoundsException aeiob){
                        System.out.println("Error: Please pass source and destination file names");
                        System.out.println("Usage: java Test abc.txt bbc.txt");
                }
                catch(FileNotFoundException fnfe){
                        System.out.println("Error: The given files "+ args[0] +" , " + args[1] + "
                                are not found, make sure they are avilable in the current path");
                }
                catch(IOException ioe)
                {
                        System.out.println("Error: Reading or writting failed");
                        e.printStackTrace();
                }
        }
}
```

Limitation of FIS and FOS

FIS and FOS allow us to read and write data only in the format of bytes. It is not possible to read or write data in the format of Primitive data or objects.

Solution

We should use *filter input stream* and *filter output stream* classes in connection with *FileInputStream* and *FileOutputStream* classes.

FilterInputStream and FilterOutputStream

Filter classes are used to improve the performance or used to add more functionality to underlying InputStream and OutputStream.

Note: From above diagram we can conclude

- The filter connected to InputStream is called FilterInputStream.
- The filter connected to OutputStream is called FilterOutputStream.
- Filters cannot connect to source or destination directly, instead they can only be connected to another *InputStream* or *OutputStream*
- So, all *FilterInputStream* and *FilterOutputStream* classes contain constructor to take other *InputStream* and *OutputStream* object as argument which it uses as its basic source of data for reading and writing.