**UNIT-IV: Multithreading and Files:**

* Introduction,
* Thread Lifecycle,
* Creation of Threads,
* Thread Priorities,
* Thread Synchronization,
* Communication between Threads.
* Reading Data from Files and Writing Data to Files,
* Random Access Files.

**Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

**Advantages of Java Multithreading**

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

## Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

* Process-based Multitasking (Multiprocessing)
* Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

* Each process has an address in memory. In other words, each process allocates a separate memory area.
* A process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another requires some time for saving and loading [registers](https://www.javatpoint.com/register-memory), memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

* Threads share the same address space.
* A thread is lightweight.
* Cost of communication between the thread is low.

# Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

## Explanation of Different Thread States

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

**Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

* **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.  
  A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.
* **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

**Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.



# Java Threads | How to create a thread in Java

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### Thread class:

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

### Commonly used Constructors of Thread class:

* Thread()
* Thread(String name)
* Thread(Runnable r)
* Thread(Runnable r,String name)

### Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

### Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

### Starting a thread:

The **start() method** of Thread class is used to start a newly created thread. It performs the following tasks:

* A new thread starts(with new callstack).
* The thread moves from New state to the Runnable state.
* When the thread gets a chance to execute, its target run() method will run.

### 1) Java Thread Example by extending Thread class

**FileName:** Multi.java

**class** Multi **extends** Thread

{

**public** **void** run()

{

System.out.println("thread is running...");

}

**public** **static** **void** main(String args[])

{

Multi t1=**new** Multi();

t1.start();

  }

}

### 2) Java Thread Example by implementing Runnable interface

**FileName:** Multi3.java

**class** Multi3 **implements** Runnable

{

**public** **void** run()

{

System.out.println("thread is running...");

}

**public** **static** **void** main(String args[])

{

Multi3 m1=**new** Multi3();

Thread t1 =**new** Thread(m1);   // Using the constructor Thread(Runnable r)

t1.start();

 }

}

If you are not extending the Thread class, your class object would not be treated as a thread object. So you need to explicitly create the Thread class object. We are passing the object of your class that implements Runnable so that your class run() method may execute.

### 3) Using the Thread Class: Thread(String Name)

We can directly use the Thread class to spawn new threads using the constructors defined above.

**FileName:** MyThread1.java

**class** MyThread1

{

**public** **static** **void** main(String argvs[])

{

Thread t= **new** Thread("My first thread");

t.start();

String str = t.getName();

System.out.println(str);

}

}

### 4) Using the Thread Class: Thread(Runnable r, String name)

**FileName:** MyThread2.java

**class** MyThread2 **implements** Runnable

{

**public** **void** run()

{

System.out.println("Now the thread is running ...");

}

**public** **static** **void** main(String argvs[])

{

Runnable r1 = **new** MyThread2();

Thread th1 = **new** Thread(r1, "My new thread");

th1.start();

String str = th1.getName();

System.out.println(str);

}

}

# Thread.sleep() in Java with Examples

The Java Thread class provides the two variant of the sleep() method. First one accepts only an arguments, whereas the other variant accepts two arguments. The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is known as the sleeping time of the thread. After the sleeping time is over, the thread starts its execution from where it has left.

### The sleep() Method Syntax:

Following are the syntax of the sleep() method.

**public** **static** **void** sleep(**long** mls) **throws** InterruptedException

**public** **static** **void** sleep(**long** mls, **int** n) **throws** InterruptedException

**Important Points to Remember About the Sleep() Method**

Whenever the Thread.sleep() methods execute, it always halts the execution of the current thread.

Whenever another thread does interruption while the current thread is already in the sleep mode, then the InterruptedException is thrown.

If the system that is executing the threads is busy, then the actual sleeping time of the thread is generally more as compared to the time passed in arguments. However, if the system executing the sleep() method has less load, then the actual sleeping time of the thread is almost equal to the time passed in the argument.

Example of the sleep() method in Java : on the custom thread

The following example shows how one can use the sleep() method on the custom thread.

**class** TestSleepMethod1 **extends** Thread{

**public** **void** run(){

**for**(**int** i=1;i<5;i++){

  // the thread will sleep for the 500 milli seconds

**try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}

    System.out.println(i);

  }

 }

**public** **static** **void** main(String args[]){

  TestSleepMethod1 t1=**new** TestSleepMethod1();

  TestSleepMethod1 t2=**new** TestSleepMethod1();

  t1.start();

  t2.start();

 }

}

Example of the sleep() Method in Java : on the main thread

**FileName:** TestSleepMethod2.java

// important import statements

**import** java.lang.Thread;

**import** java.io.\*;

**public** **class** TestSleepMethod2

{

    // main method

**public** **static** **void** main(String argvs[])

{

**try** {

**for** (**int** j = 0; j < 5; j++)

{

// The main thread sleeps for the 1000 milliseconds, which is 1 sec

// whenever the loop runs

Thread.sleep(1000);

// displaying the value of the variable

System.out.println(j);

}

}

**catch** (Exception expn)

{

// catching the exception

System.out.println(expn);

}

}

}

# Java join() method

The join() method in Java is provided by the java.lang.Thread class that permits one thread to wait until the other thread to finish its execution. Suppose th be the object the class Thread whose thread is doing its execution currently, then the th.join(); statement ensures that th is finished before the program does the execution of the next statement. When there are more than one thread invoking the join() method, then it leads to overloading on the join() method that permits the developer or programmer to mention the waiting period. However, similar to the sleep() method in Java, the join() method is also dependent on the operating system for the timing, so we should not assume that the join() method waits equal to the time we mention in the parameters. The following are the three overloaded join() methods.

**Syntax:** public final void join() throws InterruptedException

## Example of join() Method in Java

// A Java program for understanding the joining of threads

**import** java.io.\*;

**class** ThreadJoin **extends** Thread

{

**public** **void** run()

{

**for** (**int** j = 0; j < 2; j++)

{

**try**

{

Thread.sleep(300);

System.out.println("The current thread name is: " + Thread.currentThread().getName());

}

**catch**(Exception e)

{

System.out.println("The exception has been caught: " + e);

}

System.out.println( j );

}

}

}

**public** **class** ThreadJoinExample

{

**public** **static** **void** main (String argvs[])

{

ThreadJoin th1 = **new** ThreadJoin();

ThreadJoin th2 = **new** ThreadJoin();

ThreadJoin th3 = **new** ThreadJoin();

th1.start();

**try**

{

System.out.println("The current thread name is: "+ Thread.currentThread().getName());

th1.join();

}

**catch**(Exception e)

{

System.out.println("The exception has been caught " + e);

}

th2.start();

**try**

{

System.out.println("The current thread name is: " + Thread.currentThread().getName());

th2.join();

}

**catch**(Exception e)

{

System.out.println("The exception has been caught " + e);

}

th3.start();

}

}

join(long miliseconds) Method Example

**class** TestJoinMethod2 **extends** Thread{

**public** **void** run(){

**for**(**int** i=1;i<=5;i++){

**try**{

    Thread.sleep(500);

   }**catch**(Exception e){System.out.println(e);}

  System.out.println(i);

  }

 }

**public** **static** **void** main(String args[]){

 TestJoinMethod2 t1=**new** TestJoinMethod2();

 TestJoinMethod2 t2=**new** TestJoinMethod2();

 TestJoinMethod2 t3=**new** TestJoinMethod2();

 t1.start();

**try**{

  t1.join(1500);

 }**catch**(Exception e){System.out.println(e);}

 t2.start();

 t3.start();

 }

}

# Naming Thread and Current Thread

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name, i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using the setName() method. The syntax of setName() and getName() methods are given below:

public String getName(): is used to return the name of a thread.

public void setName(String name): is used to change the name of a thread.

Example of naming a thread : Using setName() Method

**class** TestMultiNaming1 **extends** Thread{

**public** **void** run(){

   System.out.println("running...");

  }

**public** **static** **void** main(String args[]){

  TestMultiNaming1 t1=**new** TestMultiNaming1();

  TestMultiNaming1 t2=**new** TestMultiNaming1();

  System.out.println("Name of t1:"+t1.getName());

  System.out.println("Name of t2:"+t2.getName());

  t1.start();

  t2.start();

  t1.setName("gdfgdf");

  System.out.println("After changing name of t1:"+t1.getName());

 }

}

**Example of naming a thread : Without Using setName() Method**

**import** java.io.\*;

**class** ThreadName **extends** Thread

{

ThreadName(String threadName)

{

**super**(threadName);

}

**public** **void** run()

{

System.out.println(" The thread is executing....");

}

}

**public** **class** ThreadNamingExample

{

**public** **static** **void** main (String argvs[])

{

ThreadName th1 = **new** ThreadName("1");

ThreadName th2 = **new** ThreadName("2");

System.out.println("Thread - 1: " + th1.getName());

System.out.println("Thread - 2: " + th2.getName());

th1.start();

th2.start();  }  }

## Current Thread

The currentThread() method returns a reference of the currently executing thread.

**public** **static** Thread currentThread()

### Example of currentThread() method

**FileName:** TestMultiNaming2.java

**class** TestMultiNaming2 **extends** Thread{

**public** **void** run(){

  System.out.println(Thread.currentThread().getName());

 }

**public** **static** **void** main(String args[]){

  TestMultiNaming2 t1=**new** TestMultiNaming2();

  TestMultiNaming2 t2=**new** TestMultiNaming2();

  t1.start();

  t2.start();

 }

}

# Synchronization in Java

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

### Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

### Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. Static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Understanding the problem without Synchronization

**TestSynchronization1.java**

**class** Table{

**void** printTable(**int** n){//method not synchronized

**for**(**int** i=1;i<=5;i++){

     System.out.println(n\*i);

**try**{

      Thread.sleep(400);

     }**catch**(Exception e){System.out.println(e);}

   }

 }

}

**class** MyThread1 **extends** Thread{

Table t;

MyThread1(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(5);

}

}

**class** MyThread2 **extends** Thread{

Table t;

MyThread2(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(100);

}

}

**class** TestSynchronization1{

**public** **static** **void** main(String args[]){

Table obj = **new** Table();//only one object

MyThread1 t1=**new** MyThread1(obj);

MyThread2 t2=**new** MyThread2(obj);

t1.start();

t2.start();

}

}

**Java Synchronized Method**

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

**TestSynchronization2.java**

//example of java synchronized method

**class** Table{

**synchronized** **void** printTable(**int** n){//synchronized method

**for**(**int** i=1;i<=5;i++){

     System.out.println(n\*i);

**try**{

      Thread.sleep(400);

     }**catch**(Exception e){System.out.println(e);}

   }

 }

}

**class** MyThread1 **extends** Thread{

Table t;

MyThread1(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(5);

}

}

**class** MyThread2 **extends** Thread{

Table t;

MyThread2(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(100);

}

}

**public** **class** TestSynchronization2{

**public** **static** **void** main(String args[]){

Table obj = **new** Table();//only one object

MyThread1 t1=**new** MyThread1(obj);

MyThread2 t2=**new** MyThread2(obj);

t1.start();

t2.start();

}

}

**Example of synchronized method by using annonymous class**

In this program, we have created the two threads by using the anonymous class, so less coding is required.

**TestSynchronization3.java**

//Program of synchronized method by using annonymous class

**class** Table{

**synchronized** **void** printTable(**int** n){//synchronized method

**for**(**int** i=1;i<=5;i++){

     System.out.println(n\*i);

**try**{

      Thread.sleep(400);

     }**catch**(Exception e){System.out.println(e);}

   }

 }

}

**public** **class** TestSynchronization3{

**public** **static** **void** main(String args[]){

**final** Table obj = **new** Table();//only one object

Thread t1=**new** Thread(){

**public** **void** run(){

obj.printTable(5);

}

};

Thread t2=**new** Thread(){

**public** **void** run(){

obj.printTable(100);

}

};

t1.start();

t2.start();

}

}

# Synchronized Block in Java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use synchronized block.

If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.

### Points to Remember

* Synchronized block is used to lock an object for any shared resource.
* Scope of synchronized block is smaller than the method.
* A Java synchronized block doesn't allow more than one JVM, to provide access control to a shared resource.
* The system performance may degrade because of the slower working of synchronized keyword.
* Java synchronized block is more efficient than Java synchronized method.

### Syntax

**synchronized** (object reference expression) {

  //code block

}

Example of Synchronized Block

**TestSynchronizedBlock1.java**

**class** Table

{

**void** printTable(**int** n){

**synchronized**(**this**){//synchronized block

**for**(**int** i=1;i<=5;i++){

      System.out.println(n\*i);

**try**{

       Thread.sleep(400);

      }**catch**(Exception e){System.out.println(e);}

     }

   }

 }//end of the method

}

**class** MyThread1 **extends** Thread{

Table t;

MyThread1(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(5);

}

}

**class** MyThread2 **extends** Thread{

Table t;

MyThread2(Table t){

**this**.t=t;

}

**public** **void** run(){

t.printTable(100);

}

}

**public** **class** TestSynchronizedBlock1{

**public** **static** **void** main(String args[]){

Table obj = **new** Table();//only one object

MyThread1 t1=**new** MyThread1(obj);

MyThread2 t2=**new** MyThread2(obj);

t1.start();

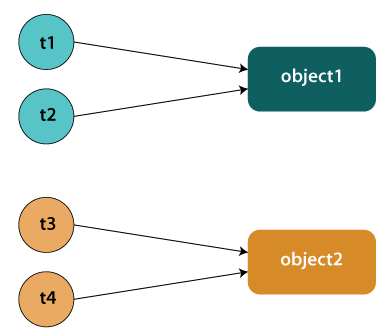
t2.start();

}

}

# Static Synchronization

If you make any static method as synchronized, the lock will be on the class not on object.



### Problem without static synchronization

Suppose there are two objects of a shared class (e.g. Table) named object1 and object2. In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock. But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock. We don't want interference between t1 and t3 or t2 and t4. Static synchronization solves this problem.

### Example of Static Synchronization

In this example we have used **synchronized** keyword on the static method to perform static synchronization.

**TestSynchronization4.java**

class Table

{

 synchronized static void printTable(int n){

   for(int i=1;i<=10;i++){

     System.out.println(n\*i);

     try{

       Thread.sleep(400);

     }catch(Exception e){}

   }

 }

}

class MyThread1 extends Thread{

public void run(){

Table.printTable(1);

}

}

class MyThread2 extends Thread{

public void run(){

Table.printTable(10);

}

}

class MyThread3 extends Thread{

public void run(){

Table.printTable(100);

}

}

class MyThread4 extends Thread{

public void run(){

Table.printTable(1000);

}

}

public class TestSynchronization4{

public static void main(String t[]){

MyThread1 t1=new MyThread1();

MyThread2 t2=new MyThread2();

MyThread3 t3=new MyThread3();

MyThread4 t4=new MyThread4();

t1.start();

t2.start();

t3.start();

t4.start();

}

}

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

Java FileInputStream class declaration

public class FileInputStream extends InputStream

Java FileInputStream class methods

|  |  |  |
| --- | --- | --- |
| **Method** | | **Description** |
| int available() | It is used to return the estimated number of bytes that can be read from the input stream. |
| int read() | It is used to read the byte of data from the input stream. |
| int read(byte[] b) | It is used to read up to **b.length** bytes of data from the input stream. |
| int read(byte[] b, int off, int len) | It is used to read up to **len** bytes of data from the input stream. |
| long skip(long x) | It is used to skip over and discards x bytes of data from the input stream. |
| FileChannel getChannel() | It is used to return the unique FileChannel object associated with the file input stream. |
| FileDescriptor getFD() | It is used to return the [FileDescriptor](https://www.javatpoint.com/java-filedescriptor-class) object. |
| protected void finalize() | It is used to ensure that the close method is call when there is no more reference to the file input stream. |
| void close() | It is used to closes the [stream](https://www.javatpoint.com/java-8-stream). |

Java FileInputStream example 1: read single character

import java.io.FileInputStream;

 class DataStreamExample

{

     public static void main(String args[])

{

          try

{

            FileInputStream fin=new FileInputStream("D:\\testout.txt");

            int i=fin.read();

            System.out.print((char)i);

            fin.close();

          }catch(Exception e){System.out.println(e);}

         }

        }

After executing the above program, you will get a single character from the file.

**Java FileInputStream example 2: read all characters**

import java.io.FileInputStream;

public class DataStreamExample

{

     public static void main(String args[])

{

          try

{

            FileInputStream fin=new FileInputStream("D:\\testout.txt");

            int i=0;

            while((i=fin.read())!=-1){

             System.out.print((char)i);

            }

            fin.close();

          }catch(Exception e){System.out.println(e);}

         }

        }

**Java FileOutputStream Class**

Java FileOutputStream is an output stream used for writing data to a [file](https://www.javatpoint.com/java-file-class).

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use [FileWriter](https://www.javatpoint.com/java-filterwriter-class) than FileOutputStream.

FileOutputStream class declaration

public class FileOutputStream extends OutputStream

FileOutputStream class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| protected void finalize() | It is used to clean up the connection with the file output stream. |
| void write(byte[] ary) | It is used to write **ary.length** bytes from the byte [array](https://www.javatpoint.com/array-in-java) to the file output stream. |
| void write(byte[] ary, int off, int len) | It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream. |
| void write(int b) | It is used to write the specified byte to the file output stream. |
| FileChannel getChannel() | It is used to return the file channel object associated with the file output stream. |
| FileDescriptor getFD() | It is used to return the file descriptor associated with the stream. |
| void close() | It is used to closes the file output stream. |

Java FileOutputStream Example 1: write byte

**import** java.io.FileOutputStream;

**public** **class** FileOutputStreamExample {

**public** **static** **void** main(String args[]){

**try**{

             FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");

             fout.write(65);

             fout.close();

             System.out.println("success...");

            }**catch**(Exception e){System.out.println(e);}

      }

}

Output:

Success...

The content of a text file **testout.txt** is set with the data **A**.

testout.txt

A

Java FileOutputStream example 2: write string

**import** java.io.FileOutputStream;

**public** **class** FileOutputStreamExample {

**public** **static** **void** main(String args[]){

**try**{

             FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");

             String s="Welcome to java.";

**byte** b[]=s.getBytes();//converting string into byte array

             fout.write(b);

             fout.close();

             System.out.println("success...");

            }**catch**(Exception e){System.out.println(e);}

      }

}

Output:

Success...

Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a [file](https://www.javatpoint.com/java-file-class). It is character-oriented class which is used for file handling in [java](https://www.javatpoint.com/java-tutorial).

Unlike FileOutputStream class, you don't need to convert string into byte [array](https://www.javatpoint.com/array-in-java) because it provides method to write string directly.

Java FileWriter class declaration

**public** **class** FileWriter **extends** OutputStreamWriter

Constructors of FileWriter class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileWriter(String file) | Creates a new file. It gets file name in [string](https://www.javatpoint.com/java-string). |
| FileWriter(File file) | Creates a new file. It gets file name in File [object](https://www.javatpoint.com/object-and-class-in-java). |

Methods of FileWriter class

|  |  |
| --- | --- |
| **Method** | **Description** |
| void write(String text) | It is used to write the string into FileWriter. |
| void write(char c) | It is used to write the char into FileWriter. |
| void write(char[] c) | It is used to write char array into FileWriter. |
| void flush() | It is used to flushes the data of FileWriter. |
| void close() | It is used to close the FileWriter. |

**Java FileWriter Example**

In this example, we are writing the data in the file testout.txt using Java FileWriter class.

import java.io.FileWriter;

public class FileWriterExample {

    public static void main(String args[]){

         try{

           FileWriter fw=new FileWriter("D:\\testout.txt");

           fw.write("Welcome to java.");

           fw.close();

          }catch(Exception e){System.out.println(e);}

          System.out.println("Success...");

     }

}

Output:

Success...

**Java FileReader Class**

Java FileReader class is used to read data from the file. It returns data in byte format like [FileInputStream](https://www.javatpoint.com/java-fileinputstream-class) class.It is character-oriented class which is used for [file](https://www.javatpoint.com/java-file-class) handling in [java](https://www.javatpoint.com/java-tutorial).

Java FileReader class declaration

public class FileReader extends InputStreamReader

Constructors of FileReader class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileReader(String file) | It gets filename in [string](https://www.javatpoint.com/java-string). It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |
| FileReader(File file) | It gets filename in [file](https://www.javatpoint.com/java-file-class) instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |

Methods of FileReader class

|  |  |
| --- | --- |
| **Method** | **Description** |
| int read() | It is used to return a character in ASCII form. It returns -1 at the end of file. |
| void close() | It is used to close the FileReader class. |

Java FileReader Example

import java.io.FileReader;

public class FileReaderExample {

    public static void main(String args[])throws Exception{

          FileReader fr=new FileReader("D:\\testout.txt");

          int i;

          while((i=fr.read())!=-1)

          System.out.print((char)i);

          fr.close();

    }

}

# Java - RandomAccessFile

This [class](https://www.javatpoint.com/object-class) is used for reading and writing to random access file. A random access file behaves like a large [array](https://www.javatpoint.com/array-in-java) of bytes. There is a cursor implied to the array called file [pointer](https://www.javatpoint.com/c-pointers), by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is [thrown](https://www.javatpoint.com/throw-keyword). It is a type of IOException.

### Constructor

|  |  |  |  |
| --- | --- | --- | --- |
| [Constructor](https://www.javatpoint.com/java-constructor) | | Description | |
| RandomAccessFile(File file, [String](https://www.javatpoint.com/java-string) mode) | | Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument. | |
| RandomAccessFile(String name, String mode) | | Creates a random access file stream to read from, and optionally to write to, a file with the specified name. | |
| Modifier and Type | Method | | Method |
| void | close() | | It closes this random access file stream and releases any system resources associated with the stream. |
| FileChannel | getChannel() | | It returns the unique [FileChannel](https://www.javatpoint.com/data-transfer-between-channels) object associated with this file. |
| int | readInt() | | It reads a signed 32-bit integer from this file. |
| String | readUTF() | | It reads in a string from this file. |
| void | seek(long pos) | | It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. |
| void | writeDouble(double v) | | It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first. |
| void | writeFloat(float v) | | It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first. |
| void | write(int b) | | It writes the specified byte to this file. |
| int | read() | | It reads a byte of data from this file. |
| long | length() | | It returns the length of this file. |
| void | seek(long pos) | | It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. |

import java.io.IOException;

import java.io.RandomAccessFile;

public class RandomAccessFileExample

{

    static final String FILEPATH ="myFile.TXT";

    public static void main(String[] args) {

        try {

            System.out.println(new String(readFromFile(FILEPATH, 0, 18)));

            writeToFile(FILEPATH, "I love my country and my people", 31);

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private static byte[] readFromFile(String filePath, int position, int size)

            throws IOException {

        RandomAccessFile file = new RandomAccessFile(filePath, "r");

        file.seek(position);

        byte[] bytes = new byte[size];

        file.read(bytes);

        file.close();

        return bytes;

    }

    private static void writeToFile(String filePath, String data, int position)

            throws IOException {

        RandomAccessFile file = new RandomAccessFile(filePath, "rw");

        file.seek(position);

        file.write(data.getBytes());

        file.close();

    }

}

The myFile.TXT contains text "This class is used for reading and writing to random access file."

after running the program it will contains

This class is used for reading I love my country and my peoplele.