Before we talk about multithreading, let’s discuss threads. A thread is a light-weight smallest part of a process that can run concurrently with the other parts(other threads) of the same process. Threads are independent because they all have separate path of execution that’s the reason if an exception occurs in one thread, it doesn’t affect the execution of other threads. All threads of a process share the common memory. The process of executing multiple threads simultaneously is known as multithreading.

What is Single Thread?

A single thread is basically a lightweight and the smallest unit of processing. Java uses threads by using a "Thread Class".

There are two types of thread – user thread and daemon thread (daemon threads are used when we want to clean the application and are used in the background).

When an application first begins, user thread is created. Post that, we can create many user threads and daemon threads.

Single Thread Example:

```
package demotest;

public class GuruThread

{

	public static void main(String[] args) {

		System.out.println("Single Thread");

	}

}
```

Advantages of single thread:

Reduces overhead in the application as single thread execute in the system

Also, it reduces the maintenance cost of the application.

What is Multithreading?

Multithreading in java is a process of executing two or more threads simultaneously to maximum utilization of CPU.

Multithreaded applications are where two or more threads run concurrently; hence it is also known as Concurrency in Java. This multitasking is done, when multiple processes share common resources like CPU, memory, etc.

Each thread runs parallel to each other. Threads don't allocate separate memory area; hence it saves memory. Also, context switching between threads takes less time.

Example of Multi thread:

```
package demotest;

public class GuruThread1 implements Runnable

{

      public static void main(String[] args) {

       Thread guruThread1 = new Thread("svcet");

       Thread guruThread2 = new Thread("mca");

       guruThread1.start();

       guruThread2.start();

       System.out.println("Thread names are following:");

       System.out.println(guruThread1.getName());

       System.out.println(guruThread2.getName());

   }

   @Override

   public void run() {

   }


}
```

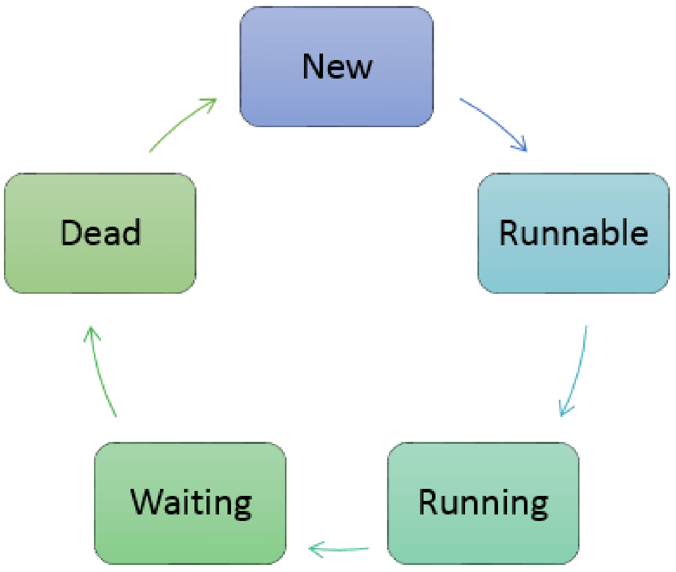
Advantages of multithread:

The users are not blocked because threads are independent, and we can perform multiple operations at times

As such the threads are independent, the other threads won't get affected if one thread meets an exception.

Thread Life Cycle in Java

The Lifecycle of a thread:



There are various stages of life cycle of thread as shown in above diagram:

New

Runnable

Running

Waiting

Dead

New: In this phase, the thread is created using class "Thread class".It remains in this state till the program starts the thread. It is also known as born thread.

Runnable: In this page, the instance of the thread is invoked with a start method. The thread control is given to scheduler to finish the execution. It depends on the scheduler, whether to run the thread.

Running: When the thread starts executing, then the state is changed to "running" state. The scheduler selects one thread from the thread pool, and it starts executing in the application.

Waiting: This is the state when a thread has to wait. As there multiple threads are running in the application, there is a need for synchronization between threads. Hence, one thread has to wait, till the other thread gets executed. Therefore, this state is referred as waiting state.

Dead: This is the state when the thread is terminated. The thread is in running state and as soon as it completed processing it is in "dead state".

Some of the commonly used methods for threads are:

Method	Description
start()	This method starts the execution of the thread and JVM calls the run() method on the thread.
Sleep(int milliseconds)	This method makes the thread sleep hence the thread's execution will pause for milliseconds provided and after that, again the thread starts executing. This help in synchronization of the threads.
getName()	It returns the name of the thread.
setPriority(int newpriority)	It changes the priority of the thread.
yield ()	It causes current thread on halt and other threads to execute.

Multitasking vs Multithreading vs Multiprocessing vs parallel processing

If you are new to java you may get confused among these terms as they are used quite frequently when we discuss multithreading. Let’s talk about them in brief.

Multitasking: Ability to execute more than one task at the same time is known as multitasking.

Multithreading: We already discussed about it. It is a process of executing multiple threads simultaneously. Multithreading is also known as Thread-based Multitasking.

Multiprocessing: It is same as multitasking, however in multiprocessing more than one CPUs are involved. On the other hand one CPU is involved in multitasking.

Parallel Processing: It refers to the utilization of multiple CPUs in a single computer system.

Creating a thread in Java

There are two ways to create a thread in Java:
1) By extending Thread class.
2) By implementing Runnable interface.

Before we begin with the programs(code) of creating threads, let’s have a look at these methods of Thread class. We have used few of these methods in the example below.

getName(): It is used for Obtaining a thread’s name

getPriority(): Obtain a thread’s priority

isAlive(): Determine if a thread is still running

join(): Wait for a thread to terminate

run(): Entry point for the thread

sleep(): suspend a thread for a period of time

start(): start a thread by calling its run() method

Method 1: Thread creation by extending Thread class

Example 1:

```
class MultithreadingDemo extends Thread{

  public void run(){

    System.out.println("My thread is in running state.");

  }

  public static void main(String args[]){

     MultithreadingDemo obj=new MultithreadingDemo();

     obj.start();
  }
}
```

Output:

```
My thread is in running state.
```

Method 2: Thread creation by implementing Runnable Interface

A Simple Example

```
class MultithreadingDemo implements Runnable{
  public void run(){
    System.out.println("My thread is in running state.");
  }
  public static void main(String args[]){
     MultithreadingDemo obj=new MultithreadingDemo();
     Thread tobj =new Thread(obj);
     tobj.start();
 }
}
```

Output:

```
My thread is in running state.
```

Thread priorities

Thread priorities are the integers which decide how one thread should be treated with respect to the others.

Thread priority decides when to switch from one running thread to another, process is called context switching

A thread can voluntarily release control and the highest priority thread that is ready to run is given the CPU.

A thread can be preempted by a higher priority thread no matter what the lower priority thread is doing. Whenever a higher priority thread wants to run it does.

To set the priority of the thread setPriority() method is used which is a method of the class Thread Class.

In place of defining the priority in integers, we can use MIN_PRIORITY, NORM_PRIORITY or MAX_PRIORITY.

Methods: isAlive() and join()

In all the practical situations main thread should finish last else other threads which have spawned from the main thread will also finish.

To know whether the thread has finished we can call isAlive() on the thread which returns true if the thread is not finished.

Another way to achieve this by using join() method, this method when called from the parent thread makes parent thread wait till child thread terminates.

These methods are defined in the Thread class.

We have used isAlive() method in the above examples too.