1) Java Thread Example by extending Thread class

```
class SingleThread extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
SingleThread t1=new SingleThread();
t1.start();
 }
}
```

2) Java Thread Example by implementing Runnable interface

```
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(Run
nable r)
t1.start();
 }
}
```

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

```
public class MyThread1
{
// Main method
public static void main(String argvs[])
{
// creating an object of the Thread class using the constructor Thre
ad(String name)
Thread t= new Thread("My first thread");

// the start() method moves the thread to the active state
t.start();
// getting the thread name by invoking the getName() method
String str = t.getName();
System.out.println(str);
}
}
```

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

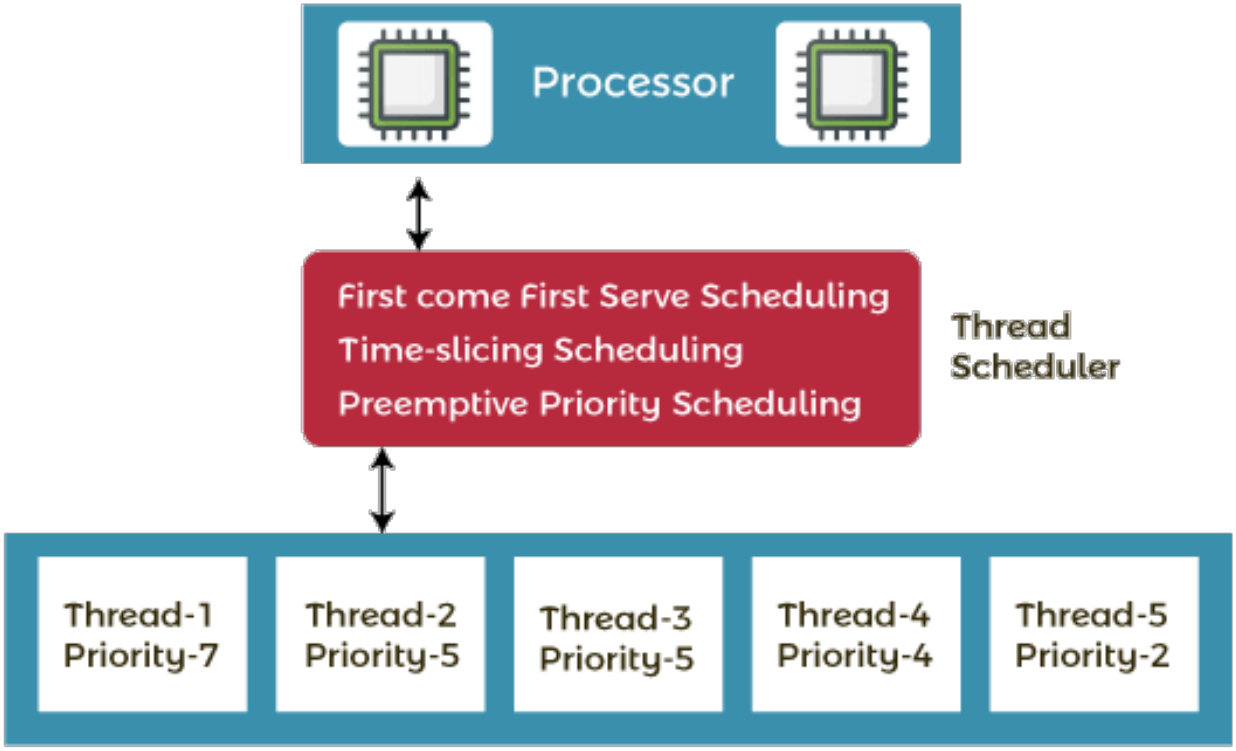
```
public class MyThread2 implements Runnable
{
public void run()
{
System.out.println("Now the thread is running ...");
}
// main method
public static void main(String argvs[])
{
// creating an object of the class MyThread2
Runnable r1 = new MyThread2();

// creating an object of the class Thread using Thread(Runnable r, S
tring name)
Thread th1 = new Thread(r1, "My new thread");
```

```
// the start() method moves the thread to the active state
th1.start();

// getting the thread name by invoking the getName() method
String str = th1.getName();
System.out.println(str);
}
}
```

Working of the Java Thread Scheduler



Working of Thread Scheduler

Example of the sleep() method in Java : 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

```
import java.lang.Thread;
import java.io.*;
public class TestSleepMethod2
{
public static void main(String argvs[])
{
 try {
for (int j = 0; j < 5; j++)
{
Thread.sleep(1000);
 System.out.println(j);
}
}
catch (Exception expn)
{  System.out.println(expn);
}
}  }
```

Example of the sleep() Method in Java: When the sleeping time is -ive

```
import java.lang.Thread;
import java.io.*;

public class TestSleepMethod3
{
public static void main(String args[])
{
try
{
for (int j = 0; j < 5; j++)
{
  Thread.sleep(-100);
System.out.println(j);
}
}
catch (Exception expn)
{
System.out.println(expn);
}
}
}
```

Output:

```
java.lang.IllegalArgumentException: timeout value is negative
```

Can we start a thread twice

```
public class TestThreadTwice1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
```

```
  TestThreadTwice1 t1=new TestThreadTwice1();
  t1.start();
  t1.start();
 }
}
```

Output:

```
running
Exception in thread "main"
java.lang.IllegalThreadStateException
```

What if we call Java run() method directly instead start() method?

```
class TestCallRun1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestCallRun1 t1=new TestCallRun1();
  t1.run();//fine, but does not start a separate call stack
 }
}
```

Output:
running

Priority of a Thread (Thread Priority)

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

```
import java.lang.*;

public class ThreadPriorityExample extends Thread
{
public void run()
{
System.out.println("Inside the run() method");
}
public static void main(String argvs[])
{
ThreadPriorityExample th1 = new ThreadPriorityExample();
ThreadPriorityExample th2 = new ThreadPriorityExample();
ThreadPriorityExample th3 = new ThreadPriorityExample();
// 1st Thread
System.out.println("Priority of the thread th1 is : " + th1.getPriority()
);
// 2nd Thread
System.out.println("Priority of the thread th2 is : " + th2.getPriority()
);
// 3rd Thread
System.out.println("Priority of the thread th2 is : " + th2.getPriority()
);
th1.setPriority(6);
th2.setPriority(3);
th3.setPriority(9);
System.out.println("Priority of the thread th1 is : " + th1.getPriority()
);
System.out.println("Priority of the thread th2 is : " + th2.getPriority()
);
System.out.println("Priority of the thread th3 is : " + th3.getPriority()
);

// Displaying name of the currently executing thread
System.out.println("Currently Executing The Thread : " + Thread.cu
rrentThread().getName());

System.out.println("Priority of the main thread is : " + Thread.curren
tThread().getPriority());

// Priority of the main thread is 10 now
```

```
Thread.currentThread().setPriority(10);

System.out.println("Priority of the main thread is : " + Thread.curren
tThread().getPriority());
}
}
```

Output:

```
Priority of the thread th1 is : 5
Priority of the thread th2 is : 5
Priority of the thread th2 is : 5
Priority of the thread th1 is : 6
Priority of the thread th2 is : 3
Priority of the thread th3 is : 9
Currently Executing The Thread : main
Priority of the main thread is : 5
Priority of the main thread is : 10
```

```
import java.lang.*;

public class ThreadPriorityExample1 extends Thread
{
 public void run()
{
// the print statement
System.out.println("Inside the run() method");
}

public static void main(String argvs[])
{

// Now, priority of the main thread is set to 7
Thread.currentThread().setPriority(7);

System.out.println("Priority of the main thread is : " + Thread.curren
tThread().getPriority());

ThreadPriorityExample1 th1 = new ThreadPriorityExample1();
```

```
System.out.println("Priority of the thread th1 is : " + th1.getPriority()
);
}
}
```

Output:

```
Priority of the main thread is : 7
Priority of the thread th1 is : 7
```

Daemon Thread in Java

Daemon thread in Java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

Points to remember for Daemon Thread in Java

- It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- Its life depends on user threads.
- It is a low priority thread.

Simple example of Daemon thread in java

```
public class TestDaemonThread1 extends Thread{
 public void run(){

if(Thread.currentThread().isDaemon()){
//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();
  t1.setDaemon(true);                  //now t1 is daemon thread
  t1.start();                          //starting threads
  t2.start();
  t3.start();
 }  }
```

Output:

```
daemon thread work
user thread work
user thread work
```

```
class TestDaemonThread2 extends Thread{
 public void run(){
  System.out.println("Name: "+Thread.currentThread().getName());
   System.out.println("Daemon: "+Thread.currentThread().isDaemon(
));
 }
  public static void main(String[] args){
  TestDaemonThread2 t1=new TestDaemonThread2();
  TestDaemonThread2 t2=new TestDaemonThread2();
```

```
  t1.start();
  t1.setDaemon(true);//will throw exception here
  t2.start();
 }
}
```

Output:

```
exception in thread main:
java.lang.IllegalThreadStateException
```

join(): When the join() method is invoked, the current thread stops its execution and the thread goes into the wait state.

Java Thread Pool

Java Thread pool represents a group of worker threads that are waiting for the job and reused many times.

In the case of a thread pool, a group of fixed-size threads is created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, the thread is contained in the thread pool again.

Thread Pool Methods

newFixedThreadPool(int s): The method creates a thread pool of the fixed size s.

newCachedThreadPool(): The method creates a new thread pool that creates the new threads when needed but will still use the previously created thread whenever they are available to use.

Advantage of Java Thread Pool

Better performance It saves time because there is no need to create a new thread.

Real time usage

It is used in Servlet and JSP where the container creates a thread pool to process the request.

File: WorkerThread.java

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class WorkerThread implements Runnable {
    private String message;
    public WorkerThread(String s){
        this.message=s;
    }
     public void run() {
                    System.out.println(Thread.currentThread().getName()
+" (Start) message = "+message);
                                                processmessage();//
call processmessage method that sleeps the thread for 2 seconds
                    System.out.println(Thread.currentThread().getName()
+" (End)");//prints thread name
    }
    private void processmessage() {
        try {  Thread.sleep(2000);  } catch (InterruptedException e) { e.
printStackTrace(); }
    }
}
```

File: TestThreadPool.java

```
public class TestThreadPool {
     public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(5)
;//creating a pool of 5 threads
        for (int i = 0; i < 10; i++) {
            Runnable worker = new WorkerThread("" + i);
                                            executor.execute(worker);//
calling execute method of ExecutorService
```

```
        }
        executor.shutdown();
        while (!executor.isTerminated()) {   }

        System.out.println("Finished all threads");
    }
}
```

Output:

```
pool-1-thread-1 (Start) message = 0
pool-1-thread-2 (Start) message = 1
pool-1-thread-3 (Start) message = 2
pool-1-thread-5 (Start) message = 4
pool-1-thread-4 (Start) message = 3
pool-1-thread-2 (End)
pool-1-thread-2 (Start) message = 5
pool-1-thread-1 (End)
pool-1-thread-1 (Start) message = 6
pool-1-thread-3 (End)
pool-1-thread-3 (Start) message = 7
pool-1-thread-4 (End)
pool-1-thread-4 (Start) message = 8
pool-1-thread-5 (End)
pool-1-thread-5 (Start) message = 9
pool-1-thread-2 (End)
pool-1-thread-1 (End)
pool-1-thread-4 (End)
pool-1-thread-3 (End)
pool-1-thread-5 (End)
Finished all threads
```

Java Thread Example - implementing Runnable interface

To make a class runnable, we can implement java.lang.Runnable interface and provide implementation in `public void run()`

method. To use this class as Thread, we need to create a Thread object by passing object of this runnable class and then call `start()` method to execute the `run()` method in a separate thread. Here is a java thread example by implementing Runnable interface.

```
public class HeavyWorkRunnable implements Runnable {

    @Override
    public void run() {
        System.out.println("Doing heavy processing - 
START "+Thread.currentThread().getName());
        try {
            Thread.sleep(1000);
            //Get database connection, delete unused data 
from DB
            doDBProcessing();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Doing heavy processing - END 
"+Thread.currentThread().getName());
    }

    private void doDBProcessing() throws 
InterruptedException {
        Thread.sleep(5000);
    }

}
```

Java Thread Example - extending Thread class

We can extend **java.lang.Thread** class to create our own java thread class and override `run()` method. Then we can create it's object and call `start()` method to execute our custom java thread class run method. Here is a simple java thread example showing how to extend Thread class.

```
public class MyThread extends Thread {

    public MyThread(String name) {
        super(name);
    }
```

```
    @Override
    public void run() {
        System.out.println("MyThread - START
"+Thread.currentThread().getName());
        try {
            Thread.sleep(1000);
            //Get database connection, delete unused data
from DB
            doDBProcessing();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("MyThread - END
"+Thread.currentThread().getName());
    }

    private void doDBProcessing() throws
InterruptedException {
        Thread.sleep(5000);
    }

}
```

Here is a test program showing how to create a java thread and execute it.

```
public class ThreadRunExample {

    public static void main(String[] args){
        Thread t1 = new Thread(new HeavyWorkRunnable(),
"t1");
        Thread t2 = new Thread(new HeavyWorkRunnable(),
"t2");
        System.out.println("Starting Runnable threads");
        t1.start();
        t2.start();
        System.out.println("Runnable Threads has been
started");
        Thread t3 = new MyThread("t3");
        Thread t4 = new MyThread("t4");
        System.out.println("Starting MyThreads");
        t3.start();
        t4.start();
        System.out.println("MyThreads has been started");
```

```
    }
}
```

Output of the above java thread example program is:

```
Starting Runnable threads
Runnable Threads has been started
Doing heavy processing - START t1
Doing heavy processing - START t2
Starting MyThreads
MyThread - START Thread-0
MyThreads has been started
MyThread - START Thread-1
Doing heavy processing - END t2
MyThread - END Thread-1
MyThread - END Thread-0
Doing heavy processing - END t1
```