<https://www.youtube.com/watch?v=br_TEuE8TbY> –very good tutorial on multithreading

[https://www.youtube.com/watch?v=lotAYC3hLVo&list=PLBB24CFB073F1048E&index=3](https://www.youtube.com/watch?v=lotAYC3hLVo&list=PLBB24CFB073F1048E&index=3good)

[good](https://www.youtube.com/watch?v=lotAYC3hLVo&list=PLBB24CFB073F1048E&index=3good) video for examples

<https://www.youtube.com/watch?v=gx_YUORX5vk&list=PLBB24CFB073F1048E&index=8>

Multithreading in Java – An Introduction

**Multithreading** in Java is one of the Important concepts in [Core Java](http://java9s.com/tutorial/videos/core-java-videos).  Most of the Beginners feel that this concept is more complex as well as confusing. In this article i have tried my best to give examples which can make the concepts simple.

Most of us have mobile phones – sorry, I mean everyone of us. Now, image a situation where we are speaking on phone and still.. we will be able to eat the food and going further, some SMART people can even watch TV while these two activities.



As humans, we are good at multitasking. Our brains are capable of shifting the priorities between these tasks and still be able to perform all of them together with ease.

In the same fashion, computers are designed to perform multiple operations in parallel so that different PROCESSES can execute in the system.  Just as we do, we are able to play the music in our computer and still be able to work on a document.

**Multithreading in Java** gives the ability to execute code by different [threads](http://java9s.com/tutorial/core-java/threads) to perform tasks in parallel or as a separate task without waiting for other to complete.

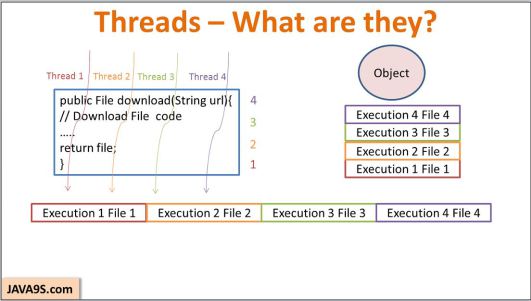


Computer games are the best examples of the **Multithreading** concept. The ability to make different objects act on a platform is enabled with the concept of **Multithreading**.

Anyone who played Road Rash – you can see that there are different bike compete with you which are driven by the computer – They are nothing but threads running in the game application.

Age of empires is also a good example of Multithreading.

We will see a good demonstration of **Multithreading in Java**.



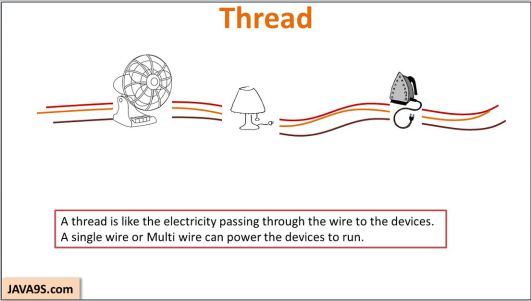
Consider, we have a method which helps in downloading the files when you give the URL.  Now, when you want to download four files, you execute this method for four times  one after the other.  But what if the files are too large and you want all the downloads to happen in parallel?

We should be able to execute this code in parallel for four times right?. That’s what threads are for. Threads are nothing but the execution engines which execute the code independent of each other.

Now when four threads execute the code in parallel, we can download four files in parallel.  – Agreed?

You should understand a point here that there need not be four object for four threads.

Four threads can execute the code in a single object and even threads can create new Objects as well.



So, when it comes to thread it is important that you should be able to imagine how things are getting executing to be able to debug the issues related to multithreading environment. So, let me take a simple real life example to explain how threads can be imagined.

Lets consider, we have three electrical devices, a fan, lamp and an Iron box.

Now think that these devices are connected to ports which are connected with multiple lines. When you power any line, these devices will start working and even power in all the lines will also make them work.

So, you can consider the power through different wires to be threads and electrical devices as objects.

I hope, I was able to use this example to ignite your imagination to make this topic about threads simple.

**Process and Threads:**

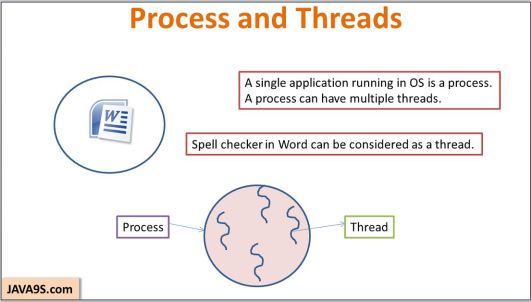
Some of the people argue that process and threads are not different. But I think they are surely different.

You can think the process as an application running in your Operating system.

Consider a word application, It is a single process and when you type in word, a spell checker will keep working in the background to correct your words, this job can be considered as a thread.

So, you should understand that a process can have multiple threads in it.

So, a**Java application** running in a system is a process and can have multiple threads inside it.



Few more points to take you little deep into the threads concept and now,, we need to keep some points on top of mind.

First, as per definition – a thread is nothing but a thread of execution which executes the code. Then..

Every application has threads running in the background. Without threads, there will be NO application that will run.

When you use java class – your main method  starts getting executed – It means that the JVM is creating a thread to start executing the code in main method and we call this method as Main thread.

It is the Operating System which schedules the threads to be processed by the Processor. So the scheduling behavior is dependent on the OS. Again there is a Thread Scheduler inside JVM which coordinates with OS scheduler.

Remember that there is nothing that is guaranteed when it comes to the thread execution.  Do not bet that your thread will be executed at a time or so.. There is no guarantee..

**Creating Threads:**

**Threads** can be created using java.lang.Thread class.

So, when I say**Thread t = new Thread()**, this statement will create a Thread Object – Right, But remember it is only an object just like any other object you would create in Java. It cannot do anything. For now it is just a sitting duck.

**Thread** is the one which executes the code and puts things into action in the application.

Only when you say to start, It starts to execute the code. So, we need to invoke the start method on the thread object.

So, where does it start from? There should be a starting point right? Yes, it is the **run() method**..

So, Every thread object you create will have run() method.

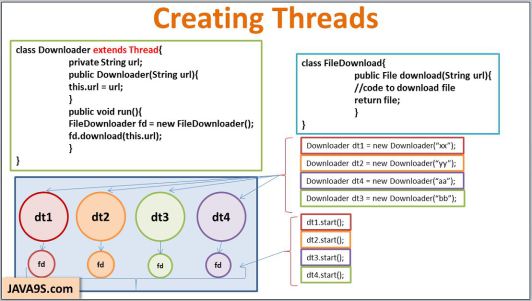
But remember the run method in the**Thread object** has nothing in it which connects to our own code.

Because this is created by the Java  api developers and they really do not what [super](http://java9s.com/topic/super) code we are going to write.

So, we need to override this run method to make it to invoke our objects, method etc,

For this, You should create your own thread type and extend it to the Thread class, then override the run method to put your own code to tell the thread to start the code execution from this point.

 So, let us see a simple Downloader example about creating threads.



Here, I have created a class called Downloader which extends thread.

This class has a constructor which gets the url from which the file has to be downloaded.

I have overridden the run method. The real download code is in the download method of FileDownloader class.

So, I have created the instance of it and I am passing the URL to this method.

Now to start downloading four files, I need four threads. So, I have created four Downloader objects.

To start downloading, I should invoke the start method. So, when the thread starts, each thread will have its corresponding FileDownloader object on which the download method will be executed.

All the four threads here run independent of each other. Any objects or values initialized by a thread will not be visible or available to other threads.

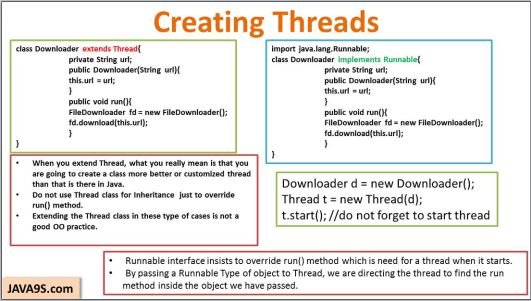
**Using Runnable Interface instead of Thread:**

If you look at the earlier code, you have seen that I have extended the [**Thread**](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.html)class. But this is inheritance here which means that Downloader is more advanced class than Thread class.

But in reality, the Downloader is no substitute in providing advanced methods. So, it is always recommended not inherit Thread class unless you are sure that you are going to write more sophisticated class than that is there in the Java.

So, the whole purpose of extending was to override the run method. For that Java api provides a Runnable interface which enforces to override run method.

Now , we need to create a Thread and tell where the target run is available for it to start executing. This we call as runnable target.



Thread class constructor accepts the [**Runnable**](http://docs.oracle.com/javase/6/docs/api/java/lang/Runnable.html)type of object which understand that the run method is present in that object.

This way it helps to keep code simple and save the inheritance ladder.

After creating the Thread object and passing the runnable target, we should not forget to invoke the start method on it. If not, nothing is going to start.

**Threads and its constructors:**

There are three components which work with threads –

1. [Runnable](http://docs.oracle.com/javase/6/docs/api/java/lang/Runnable.html)**target object** in which the run method exists.
2. **Name of the thread** which you can use the thread to identify in middle of the execution.
3. And the last one is the [**Thread group**](http://docs.oracle.com/javase/7/docs/api/java/lang/ThreadGroup.html). You can group your threads based on the activities they do.

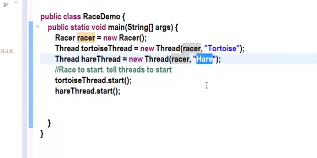
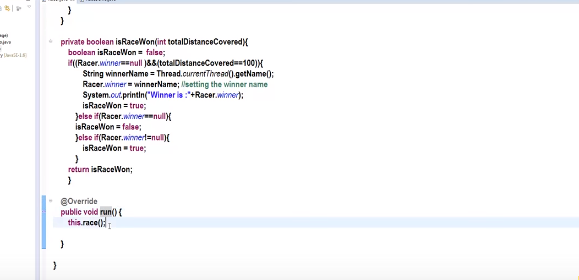
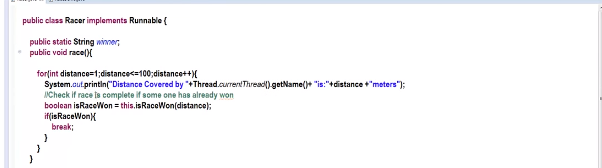
Based on these three, The Thread class has constructors which accept different combinations of these parameters.

Threads Example:

Let us implement an example to create Threads. Consider a Race where two threads race to complete a 100 meters distance. Given below is the implementation of Race example:

A player is considered as Racer and the class implements [Runnable Interface](http://docs.oracle.com/javase/6/docs/api/java/lang/Runnable.html) which indicates to Threads that its run method is implemented in Racer class.

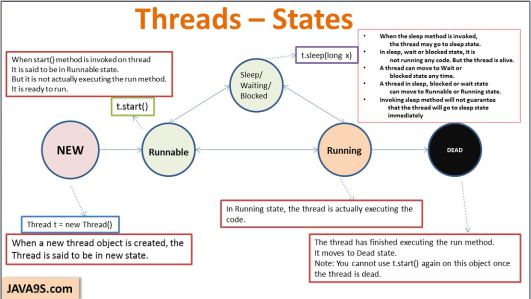
This class makes the thread to iterate for 100 times and for each iteration, it checks if some one has won the race and if already a Thread won the race, then the other quits.

Thread States

From creation of Thread Objects to their Termination, [Threads](http://java9s.com/tutorial/core-java/threads) transit through different states in their life time.

When we create a new thread object, the Thread is said to be in NEW state. In this state it is just an object like any other object that you create in Java. This thread object is not a real execution engine which can execute code.  
To make a thread object to execute code, we invoke start method on it. When start is invoked, The thread is said to be in Runnable state. Remember that In this state the thread is not executing any code but it is ready to execute the code in the run method.

Thread States- New, Runnable, Running, [Sleep](http://java9s.com/topic/sleep)/wait/blocked and Dead

When the thread starts executing the code in the run method, it is said to be in RUNNING state.   
Other than these two states, a thread can also move to Sleep/Wait/ Blocked state. When we invoke sleep method on the current thread, we are telling it to pause for specified amount of time. Apart from sleep, a thread can also move to Wait of blocked state.  
When the thread is in any of these three states, it is said to be active or alive .   
Finally when all the code that is in the run method is complete, the thread is Terminated and we call it a Dead Thread.  
Once a Thread is dead, we cannot invoke the start method to start it again. It is like a Black hole.

Sleep Method in Thread:

There will be some scenarios where in you want a thread to stop executing the code for a period of time and then start again.

* Sleep method in Thread tells the currently executing thread to sleep for specified amount of time.
* Sleep method accepts the time in MilliSeconds.
* It can throw interruptedException . So it should be embedded in the try catch block
* Remember that there is no guarantee that the thread will go to Sleep state the moment is it executed and no guarantee that the thread will sleep for specified amount of time. The thread scheduler can wake it up any time.
* Once the thread completes or out of its sleep state, it can move to Running or Runnable state.

Thread Sleep – An example

Racer.java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | public class Racer implements Runnable {    public static String winner;  public void race() {  for(int distance=1;distance&lt;=100;distance++){  System.out.println("Distance Covered by "+Thread.currentThread().getName()+ "is:"+distance +"meters");  //Check if race is complete if some one has already won  if(Thread.currentThread().getName().equals("Hare")&amp;&amp;distance==30){  try {  Thread.sleep(1000\*10);//Will sleep for Ten seconds  } catch (InterruptedException e) {    e.printStackTrace();  }  }  boolean isRaceWon = this.isRaceWon(distance);  if(isRaceWon){  break;  }  }  }    private boolean isRaceWon(int totalDistanceCovered){  boolean isRaceWon =  false;  if((Racer.winner==null )&amp;&amp;(totalDistanceCovered==100)){  String winnerName = Thread.currentThread().getName();  Racer.winner = winnerName; //setting the winner name  System.out.println("Winner is :"+Racer.winner);  isRaceWon = true;  }else if(Racer.winner==null){  isRaceWon = false;  }else if(Racer.winner!=null){  isRaceWon = true;  }  return isRaceWon;  }    @Override  public void run() {  this.race();    }    }    RaceDemo.java  public class RaceDemo {  public static void main(String[] args) {  Racer racer = new Racer();  Thread tortoiseThread = new Thread(racer, "Tortoise");  Thread hareThread = new Thread(racer, "Hare");  //Race to start. tell threads to start  tortoiseThread.start();  hareThread.start();    }  } |

**Thread Priorities:**

Understanding the Thread priorities is important to know how [yield](http://java9s.com/topic/yield) method works in Multithreading.

* Remember that all the threads carry normal priority when a priority is not specified.
* Priorities can be specified from 1 to 10. 10 being the highest, 1 being the lowest priority and 5 being the normal priority.
* Remember that the thread with highest priority will be given preference in execution. But there is no guarantee that it will be in running state the moment it starts.
* Always the currently executing thread might have the higher priority when compared to the threads in the pool who are waiting for their chance.
* It is the thread scheduler which decides what thread should be executed.
* t.setPriority() can be used to set the priorities for the threads.
* Remember that the priorities should be set before the threads start method is invoked.
* You can use the constants, MIN\_PRIORITY,MAX\_PRIORITY and NORM\_PRIORITY for setting priorities.

**Yield Method in Thread:**

* Yield is a Static method .
* Yield tells the currently executing thread to give a chance to the threads that have equal priority in the Thread Pool.
* There is no guarantee that Yield will make the currently executing thread to runnable state immediately.
* Remember an important point that yield method does not make the thread to go to Wait or Blocked state. It can only make a thread from Running State to Runnable State.

**Yield Example:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86 | YieldDemo.java    //This class prints the Stats and also the threads priorities while executing the yield method  public class YieldDemo {  public static void main(String[] args) {    YeildRunnable yr = new YeildRunnable();  ThreadGroup allThreads= new ThreadGroup("All Threads");  Thread t1 = new Thread(allThreads,yr,"t1");  Thread t2 = new Thread(allThreads,yr,"t2");  Thread t3 = new Thread(allThreads,yr,"t3");  Thread t4 = new Thread(allThreads,yr,"t4");  Thread t5 = new Thread(allThreads,yr,"t5");  Thread t6 = new Thread(allThreads,yr,"t6");  Thread t7 = new Thread(allThreads,yr,"t7");  Thread t8 = new Thread(allThreads,yr,"t8");  Thread t9 = new Thread(allThreads,yr,"t9");  t1.setPriority(Thread.MAX\_PRIORITY);  t2.setPriority(Thread.MAX\_PRIORITY);  t3.setPriority(Thread.MAX\_PRIORITY);    t4.setPriority(Thread.NORM\_PRIORITY);  t5.setPriority(Thread.NORM\_PRIORITY);  t6.setPriority(Thread.NORM\_PRIORITY);    t7.setPriority(Thread.MIN\_PRIORITY);  t8.setPriority(Thread.MIN\_PRIORITY);  t9.setPriority(Thread.MIN\_PRIORITY);    // starting all the threads  t1.start();  t2.start();  t3.start();  t4.start();  t5.start();  t6.start();  t7.start();  t8.start();  t9.start();    //Getting the status of each thread using the Main thread  /\*while(true){  State t1State = t1.getState();  State t2State = t2.getState();  State t3State = t3.getState();  State t4State = t4.getState();  State t5State = t5.getState();  State t6State = t6.getState();  State t7State = t7.getState();  State t8State = t8.getState();  State t9State = t9.getState();    System.out.println(System.nanoTime()+" t1 Status:"+t1State);  System.out.println(System.nanoTime()+" t2 Status:"+t2State);  System.out.println(System.nanoTime()+" t3 Status:"+t3State);  System.out.println(System.nanoTime()+" t4 Status:"+t4State);  System.out.println(System.nanoTime()+" t5 Status:"+t5State);  System.out.println(System.nanoTime()+" t6 Status:"+t6State);  System.out.println(System.nanoTime()+" t7 Status:"+t7State);  System.out.println(System.nanoTime()+" t8 Status:"+t8State);  System.out.println(System.nanoTime()+" t9 Status:"+t9State);    //Checking if all the threads are terminated  int activeThreads = allThreads.activeCount();  if(activeThreads==0){  break;  }  }\*/    }    }    YieldRunnable.java  public class YeildRunnable implements Runnable {    @Override  public void run() {    System.out.println("Current Thread:"+Thread.currentThread().getName()+" Priority "+Thread.currentThread().getPriority());  Thread.yield();  System.out.println("Current Thread:"+Thread.currentThread().getName()+" Priority "+Thread.currentThread().getPriority()+" -End");    }    } |

**Join Method in Thread:**

* Join method belongs to Thread object and not a static member. When join is invoked an a thread instance, this thread will tell currently executing thread to wait till the Joining thread completes.
* Join is used in the situations when a task which should be completed before the current task is going to finish.
* Just to give another example of how Join works, say we have a slow train on the track moving and another fast train should take the same track, so, we make way to the fast train and tell the slow train to join once after the fast one is left.

**Join method Example:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | JoinDemo.java    public class JoinDemo {  public static Thread firstThread;  public static Thread secondThread;  public static void main(String[] args) {  JoinThreads jt = new JoinThreads();  firstThread = new Thread(jt,"First Thread");  secondThread = new Thread(jt, "Second Thread");  firstThread.start();    while(true){  State firstThreadStatus = firstThread.getState();  State secondThreadStatus = secondThread.getState();  System.out.println(System.nanoTime()+" First Thread status:"+firstThreadStatus);  System.out.println(System.nanoTime()+" Second Thread Status :"+secondThreadStatus);  if(firstThreadStatus.equals(State.TERMINATED)&amp;&amp;secondThreadStatus.equals(State.TERMINATED)){  break;  }  }  // secondThread.start();    }    }    public class JoinThreads implements Runnable {    @Override  public void run() {  //for(int i =0 ; i&lt;100;i++){  System.out.println(System.nanoTime()+" Current Thread:"+Thread.currentThread().getName()+" is at the beginning");    //}  if(Thread.currentThread().getName().equals("First Thread")){  try {  System.out.println(System.nanoTime()+" Inside the join condition");  JoinDemo.secondThread.start();  JoinDemo.secondThread.join();  //Getting the first threads state    } catch (InterruptedException e) {  e.printStackTrace();  }    }    System.out.println(System.nanoTime()+" Current Thread:"+Thread.currentThread().getName()+" is at the end");  }    } |

<https://www.youtube.com/watch?v=hg790546fEE>

->watch all 4 videos