

Java Threads

- A thread is a process that runs independently.
- A program starts off running using a single thread which spawns other threads.
- Multi-threading is a major part of Java Network Programming.
- To start the execution of a thread, the `start()` method is called.
- In turn, the `start()` method calls the `run()` method.
- You may stop or interrupt a thread by calling the `stop()` or `interrupt()` method, respectively.
- There are two types of threads: a **daemon thread** and a **user thread**.
- A **daemon thread** is a background thread that is subordinate to the thread that created it. It is created by calling the `setDaemon()` method with an argument set to *true* before it starts.
- A **user thread** has a life of its own.
- There are two ways of defining a multi-threaded application: instantiate a thread or implement a Runnable interface.

// A Thread subclass application

```
public class SampleThread extends Thread {
    private int threadNum;

    public SampleThread(int val){
        threadNum = val;
    }//end of constructor

    public void run( ){
        System.out.println("Starting Thread Number: " + threadNum);
        try {
            Thread.sleep((long)(Math.random()*10000));
        }catch (InterruptedException e) { }
        System.out.println("Stopping Thread Number: " + threadNum);
    }//end of run

    public static void main (String [ ] args){
        for (int i=0; i<5; i++){
            Thread thr = new SampleThread(i);
            thr.start();
        }
    }//end of main
} //end of class definition
```

- Sometimes, it is better to create a thread by implementing a Runnable because we may need to extend the functionality of the instance object. Note that, in Java, inheritance is allowed only once.

// A Runnable implementation

```
public class SampleRunnable implements Runnable {
    private int threadNum;

    public SampleRunnable(int val){
        threadNum=val;
    }//end of constructor method

    public void run(){
        System.out.println("Starting Runnable Thread Number: " + threadNum);

        try {
            Thread.sleep((long)(Math.random()*10000));
            System.out.println("Completed sleep..now Waking up Thread Number: " +
                threadNum);
        }catch (InterruptedException e) {
            System.out.println("Interrupted Runnable Thread Number: "+
                threadNum);
        }

    }

    }//end of run

    public static void main (String [] args){
        Thread [ ] thr = new Thread[5];
        for (int i=0; i<5; i++){
            Runnable srun = new SampleRunnable(i); //create a runnable object
            thr[i] = new Thread(srun);           //create a thread using the runnable object
            thr[i].start();
            int flag = (int) (Math.random()*3);

            if (flag==0) thr[i].interrupt(); //interrupt thread
        }//end of for loop

    }//end of main
} //end of class definition
```

- Connecting Threads can be achieved by using the **join()** method. Thus, in the above code snippet, if the code

```
try {
    thr[4].join();
    System.out.println("Main is out of here....");
} catch (InterruptedException e) { }
```

is inserted before the end of the main() method, the main thread will wait for the death of thread thr[4] before proceeding.