A multi-threaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

When a Java program starts up, one thread begins running immediately. This is usually called the *main* thread of our program because it is the one that is executed when our program begins.

There are certain properties associated with the main thread which are as follows:

* It is the thread from which other “child” threads will be spawned.
* Often, it must be the last thread to finish execution because it performs various shutdown actions

**Thead Name**

**Numeric t1 = new Numeric("MyThread ");**

or

**Numeric t1 = new Numeric();**

**t1.setName("MyThread");**

or

**Thread tx=new Thread(t2,"MyThread")**

**t1.setPriority(10);**

**tx.setPriority(Thread.MAX\_PRIORITY);**

-----

**in run()**

System.***out***.println("Thread Name : "+Thread.*currentThread*().getName());

 "Thread " + Thread.currentThread().getId() + " is running");

-----

**Reading / writing priority of main thread**

 System.out.println(

            "Main thread priority : "

            + Thread.currentThread().getPriority());

        // Main thread priority is set to 10

        Thread.currentThread().setPriority(10);

        System.out.println(

            "Main thread priority : "

            + Thread.currentThread().getPriority());

------

  Thread ct = new Thread() {

            // run() method of a thread

            public void run()

            {

                for (int i = 0; i < 5; i++) {

                    System.out.println("Child thread");

                }

            }

        };

**Thread Lambda demo**

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

Thread t1=**new** Thread(

()->

{

**for**(**int** i=1;i<=26;i++)

System.***out***.println(i);

}

);

t1.start();

}

-----

**Thread States**

Numeric t1 = **new** Numeric("MyThread");

t1.start();

**while** (**true**) {

Thread.State state = t1.getState();

System.***out***.println(state);

**if** (state == Thread.State.***TERMINATED***) {

**break**;

}

}

--------

Soak t1=new Soak();

Wash t2=new Wash();

Dry t3=new Dry();

t1.start();

t1.join();//join(2000)//join(2000,100)

t2.start();

t2.join();

t3.start();

---------

**package** threadsynchronization;

**public** **class** Bank {

**private** **double** balance;

**public** Bank()

{

**this**.balance=3000;

}

**public** **double** getBalance()

{

**return** **this**.balance;

}

**public** **synchronized** **void** withdraw(**double** amount)

{

**if**(**this**.balance<amount)

{

System.***out***.println("Insufficient Funds :" + getBalance());

}

**try**

{

wait();

Thread.*sleep*(2000);

}

**catch**(InterruptedException ie){}

System.***out***.println("Withdrawl Successful" );

**this**.balance-=amount;

System.***out***.println("New Balance " +getBalance());

}

**public** **synchronized** **void** deposit(**double** amount)

{

**this**.balance+=amount;

notify();

}

}

-----

**package** threadsynchronization;

**class** Father **extends** Thread {

Bank btemp;

**public** Father(Bank bx) {

**this**.btemp = bx;

}

**public** **void** run() {

btemp.deposit(7000);

}

}

----

**package** threadsynchronization;

**public** **class** Son **extends** Thread{

Bank btemp;

**public** Son(Bank bx) {

**this**.btemp = bx;

}

**public** **void** run() {

btemp.withdraw(5000);

}

}

-----

**package** threadsynchronization;

**public** **class** ITCDemo {

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

Bank b1=**new** Bank();

Father t1=**new** Father(b1);

Son t2=**new** Son(b1);

t2.start();

t1.start();

}

}

Deadlock

psvm()

{

  System.out.println("Entering into Deadlock");

 Thread.currentThread().join();

 System.out.println("This statement will never execute");

}

// The statement “Thread.currentThread().join()”, will tell Main thread to wait for this thread(i.e. wait for itself) to die. Thus Main thread wait for itself to die, which is nothing but a deadlock