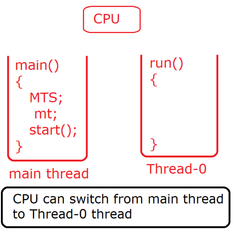
Creating user-defined Thread by using extends Thread class approach :

```
class MyThread extends Thread
{
   @Override
   public void run()
   {
      //Whatever the task we want to perform from the thread, we should write here
   }
}
public class CustomThread
{
  public static void main(String [] args)
  {
     System.out.println("Main thread started!!!");
     MyThread mt = new MyThread();
     mt.start();
     System.out.println("Main thread ended!!!");
   }
}
```



public synchronized void start() :

* It is a predefined non static method of Thread class. It performs the following two tasks :

a) It will make a request to the Operating System to **assign a new thread for concurrent execution.**
b) It will **implicitly call run()** method of the current object.

Note : In this java.lang package, only start() method is used to create a new thread for concurrent execution.

Every time a new thread will be created by using start() method then It would be created in a separate runtime STACK MEMORY. For Every Thread we have separate Stack Memory.

```
package com.ravi.basic;

class MyThread extends Thread
{
    @Override
    public void run()
    {
        System.out.println("Hello user thread, Are you ready to perform task!!");
    }
}

public class CustomThread
{
    public static void main(String[] args)
    {
        System.out.println("Main Thread started!!!");

        MyThread mt = new MyThread();

        mt.start();

        System.out.println("Main Thread ended!!!");
    }

}
```

Note : start() method is synchronized but in advanced version of JDK It is not synchronized.

---

public final boolean isAlive() :

As we know when we call start() method then a new thread will be created in a separate Stack Memory.

Thread class has provided a predefined non static method called isAlive(), return type is boolean. This method is used to verify whether a thread has started or not, that thread is alive or not.(ThreadGroup)

If we use isAlive() method before the start method then it will return false because Thread has not started yet, on the other hand if we use isAlive() method, after start() method then it will return true.

In java, We cannot re-start a thread, If we try to re-start then it will generate a runtime exception i.e java.lang.IllegalThreadStateException.

```
package com.ravi.basic;

class Test extends Thread
{
    @Override
    public void run()
    {
        System.out.println("Child thread is running...");
        System.out.println("It is running with separate stack memory");
    }
}

public class IsAliveDemo
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        System.out.println("Child thread not started yet so, :"+t1.isAlive());
        t1.start();
        System.out.println("Child thread started so, :"+t1.isAlive());

        t1.start();  //java.lang.IllegalThreadStateException
    }
}
```

```
package com.ravi.basic;

class Stuff extends Thread
{
    @Override
    public void run()
    {
        String name = Thread.currentThread().getName();
        System.out.println("Child Thread is Running, name is :"+name);
    }
}
public class ExceptionDemo
{
    public static void main(String[] args)
    {
        String name = Thread.currentThread().getName();
         System.out.println(name+" thread started");

        Stuff s1 = new Stuff();
        Stuff s2 = new Stuff();

        s1.start();
        s2.start();

        System.out.println(10/0);

        System.out.println("Main Thread Ended");
    }

}
```

Note :- Here main thread is interrupted due to AE but still child threads will be executed because child threads are executing with separate Stack.