What does start() function do in multithreading in Java?

We have discussed that Java threads are typically created using one of the two methods : (1) Extending thread class. (2) Implementing Runnable

In both the approaches, we override the run() function, but we start a thread by calling the start() function. So why don't we directly call the oveerridden run() function? Why always the start function is called to execute a thread?

What happens when a function is called?
When a function is called the following operations take place:

1. The arguments are evaluated.
2. A new stack frame is pushed into the call stack.
3. Parameters are initialized.
4. Method body is executed.
5. Value is retured and current stack frame is popped from the call stack.

The purpose of start() is to create a separate call stack for the thread. A separate call stack is created by it, and then run() is called by JVM.

Let us see what happens if we don't call start() and rather call run() directly. We have modified the first program discussed here.


```
// Java code to see that all threads are
// pushed on same stack if we use run()
// instead of start().
class ThreadTest extends Thread
{
  public void run()
  {
    try
    {
      // Displaying the thread that is running
      System.out.println ("Thread " +
                Thread.currentThread().getId() +
                " is running");

    }
    catch (Exception e)
    {
      // Throwing an exception
      System.out.println ("Exception is caught");
    }
  }
}

// Main Class
public class Main
{
  public static void main(String[] args)
  {
    int n = 8;
    for (int i=0; i<n; i++)
    {
      ThreadTest object = new ThreadTest();

      // start() is replaced with run() for
      // seeing the purpose of start
      object.run();
    }
  }
}
```

Run on IDE

Output:

```
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
```

We can see from above output that we get same ids for all threads because we have directly called run(). The program that calls start() prints different ids (see this)