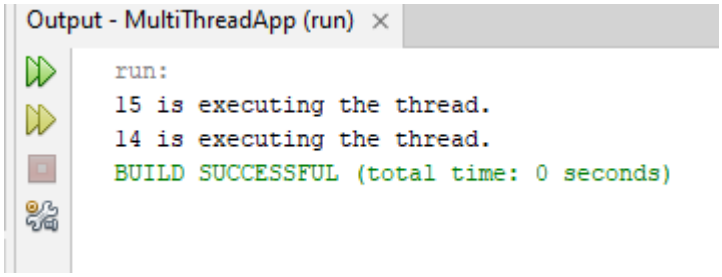
01.Create a Simple Thread Class

```
package multithreadapp;

public class SimpleThread extends Thread {
   public void run () {
System.out.println(Thread.currentThread(). getId () + " is executing the thread.");
}}

public class MultiThreadApp {

    public static void main(String[] args) {

       SimpleThread thread1 = new SimpleThread();
       SimpleThread thread2 = new SimpleThread();
       thread1.start(); // Starts thread1
       thread2.start(); // Starts thread2
   }
  }
```

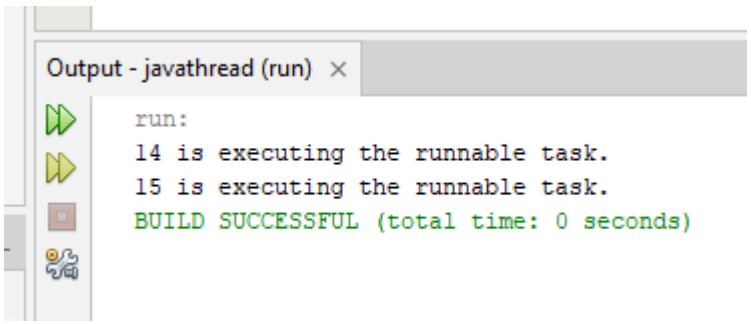
Output:



02. creating runnable class

```
public class RunnableTask implements Runnable {
   @Override
   public void run () {
System.out.println(Thread.currentThread().getId() + " is executing the runnable task.");
}}

public class Javathread {
public static void main(String[] args) {
     RunnableTask task1 = new RunnableTask();
     RunnableTask task2 = new RunnableTask();
     Thread thread1 = new Thread(task1);
     Thread thread2 = new Thread(task2);
      thread1.start(); // Starts thread1
      thread2.start(); // Starts thread2
   }
  }
```

Output:

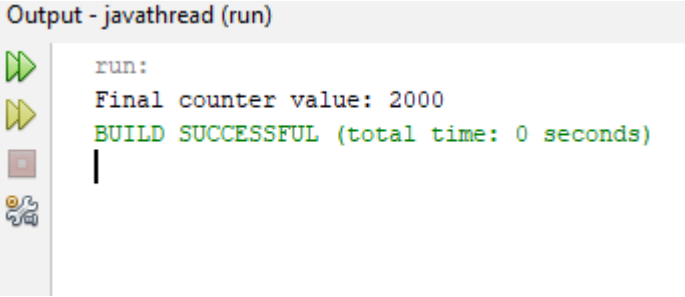


03. synchronizing shared resources

```
public class counter {
    private int count = 0;
// Synchronized method to ensure thread-safe access to the counter
public synchronized void increment() {
count++;}
public int getCount() {
return count;}
}

public class SynchronizedExample extends Thread {
private counter counter;
public SynchronizedExample(counter counter) {
this.counter = counter;}
@Override
public void run() {
for (int i = 0; i < 1000; i++) {
counter.increment();  }}
}

counter counter = new counter();
      // Create and start multiple threads
Thread thread1 = new SynchronizedExample(counter);
Thread thread2 = new SynchronizedExample(counter);
thread1.start();
thread2.start();
// Wait for threads to finish
    try {
        thread1.join();
```

```
    } catch (InterruptedException ex) {
        Logger.getLogger(Javathread.class.getName()).log(Level.SEVERE, null, ex);
    }
thread2.join();
System.out.println("Final counter value: " + counter.getCount());
    }
  }
```

Output:

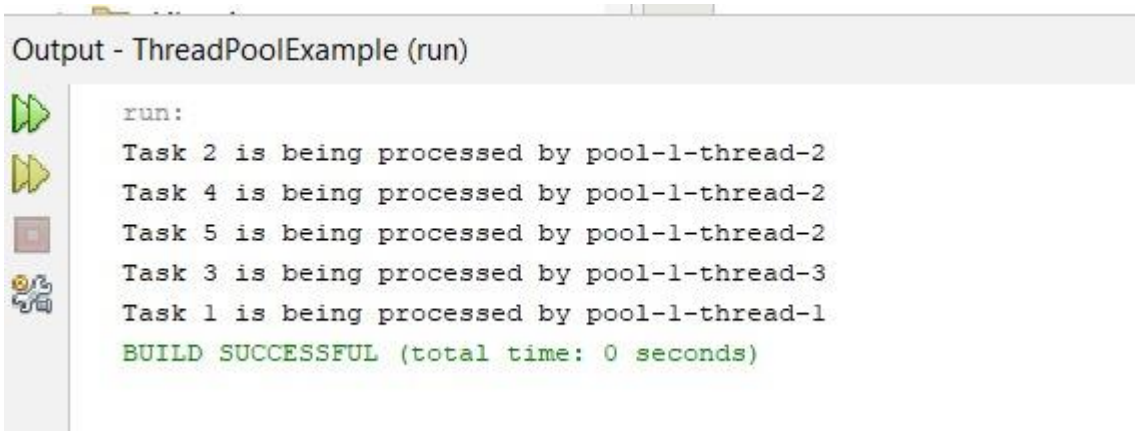


04. Using Executor Service for Thread Pooling

```
package threadpoolexample;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Threadpoolexample {

    public static void main(String[] args) {
        // Create a thread pool with 3 threads
ExecutorService executorService = Executors.newFixedThreadPool(3);
// Submit tasks to the pool
for (int i = 1; i <= 5; i++) {
executorService.submit(new Task(i));
}
// Shutdown the thread pool
executorService.shutdown();

    }

package threadpoolexample;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class task implements Runnnable {
    private int taskId;
public task(int taskId){
    this.taskId=taskId;
}
public void run() {
```

```
System.out.println("Task " + taskId + " is being processed by "
+Thread.currentThread().getName());


}
```

Output:



05. Thread Lifecycle Example

```
public class ThreadLifecycleExample extends Thread {
    @Override
    public void run() {
System.out.println(Thread.currentThread().getName() + " - State:
+Thread.currentThread().getState());
try {
Thread.sleep(2000); // Simulate waiting state
} catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println(Thread.currentThread().getName() + " - State aftersleep: " +
Thread.currentThread().getState());
}
}

public class Javathread {
public static void main(String[] args) {
ThreadLifecycleExample thread = new ThreadLifecycleExample();
System.out.println(thread.getName() + " - State before start: " +thread.getState());
thread.start(); // Start the thread
System.out.println(thread.getName() + " - State after start: " +thread.getState());
}}
```

Output:

```
Output - javathread (run)
run:
Thread-0 - State before start: NEW
Thread-0 - State after start: RUNNABLE
Thread-0 - State: RUNNABLE
Thread-0 - State aftersleep: RUNNABLE
BUILD SUCCESSFUL (total time: 2 seconds)
```