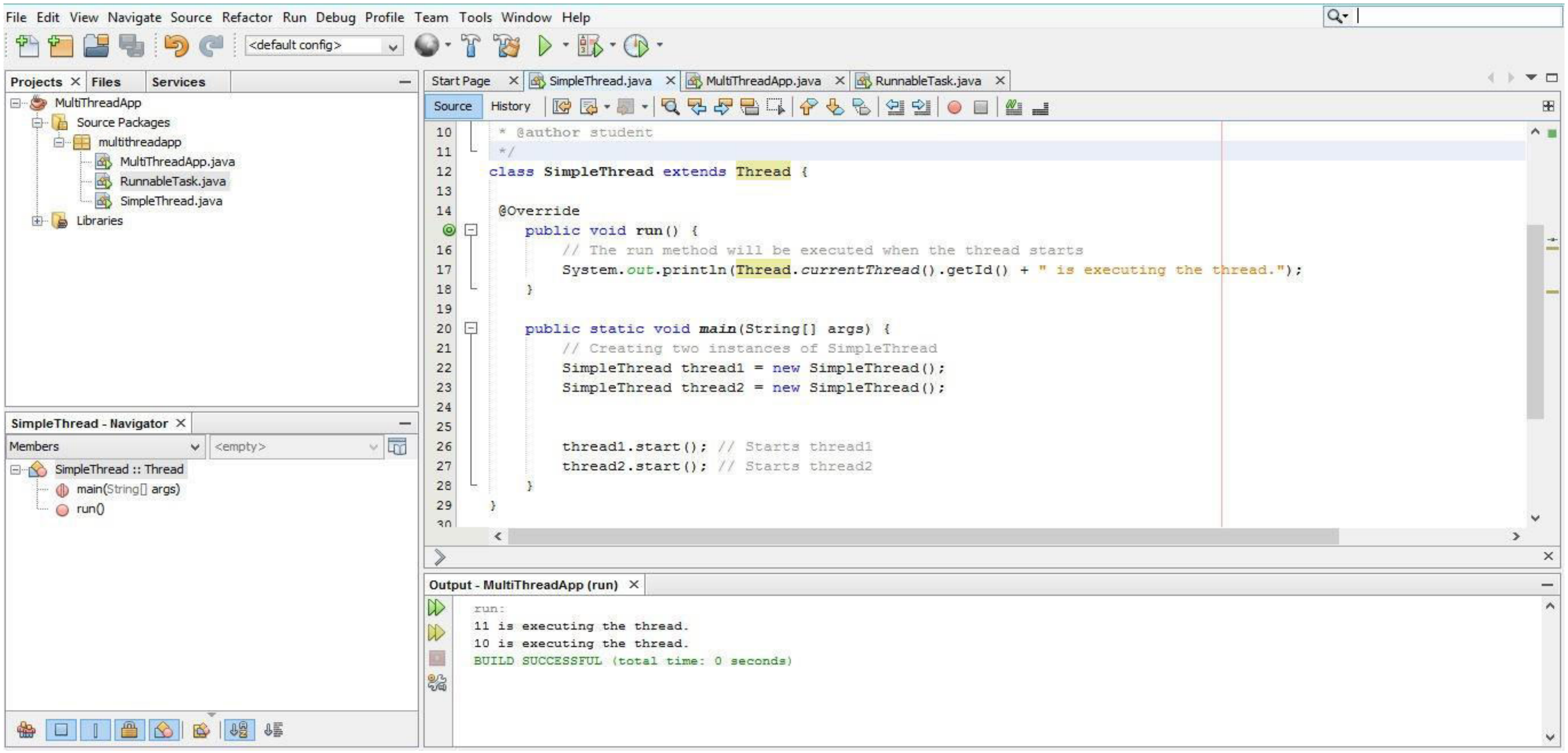
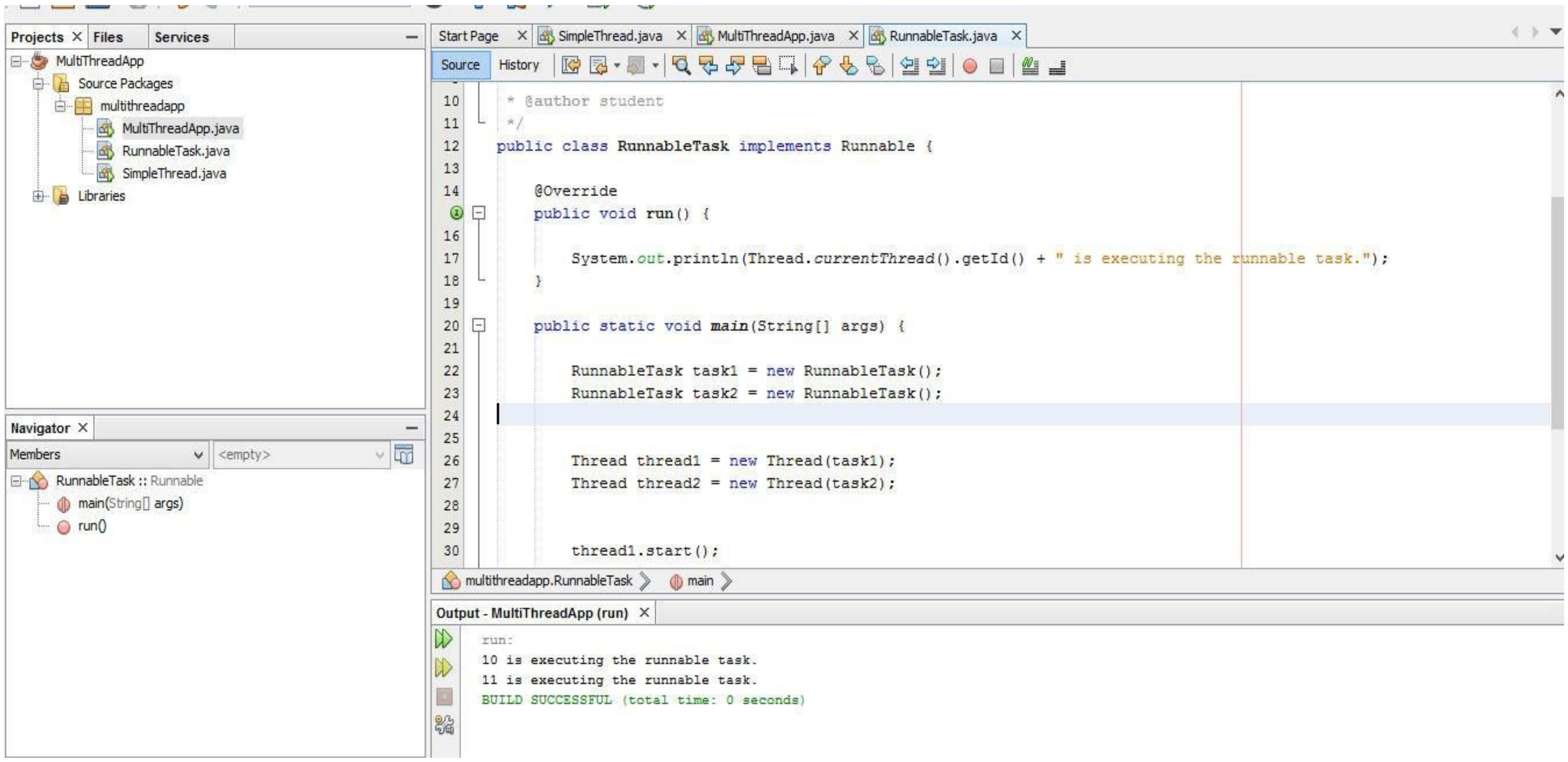
1. Create a Simple Thread Class



```java
 * @author student
 */
class SimpleThread extends Thread {

 @Override
    public void run() {
        // The run method will be executed when the thread starts
        System.out.println(Thread.currentThread().getId() + " is executing the thread.");
    }

    public static void main(String[] args) {
        // Creating two instances of SimpleThread
        SimpleThread thread1 = new SimpleThread();
        SimpleThread thread2 = new SimpleThread();


        thread1.start(); // Starts thread1
        thread2.start(); // Starts thread2
    }
}
```

```
run:
11 is executing the thread.
10 is executing the thread.
BUILD SUCCESSFUL (total time: 0 seconds)
```

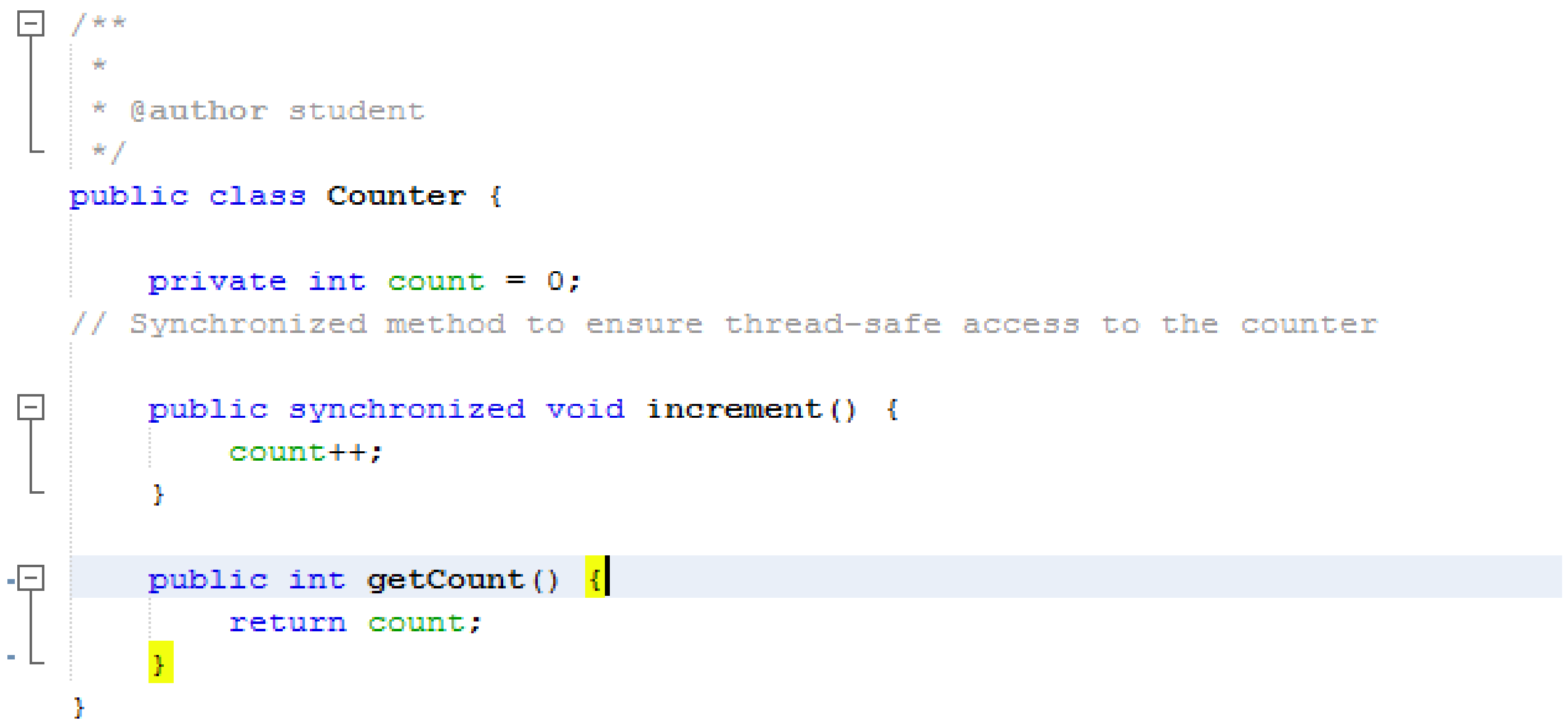
2. Create a Runnable Class



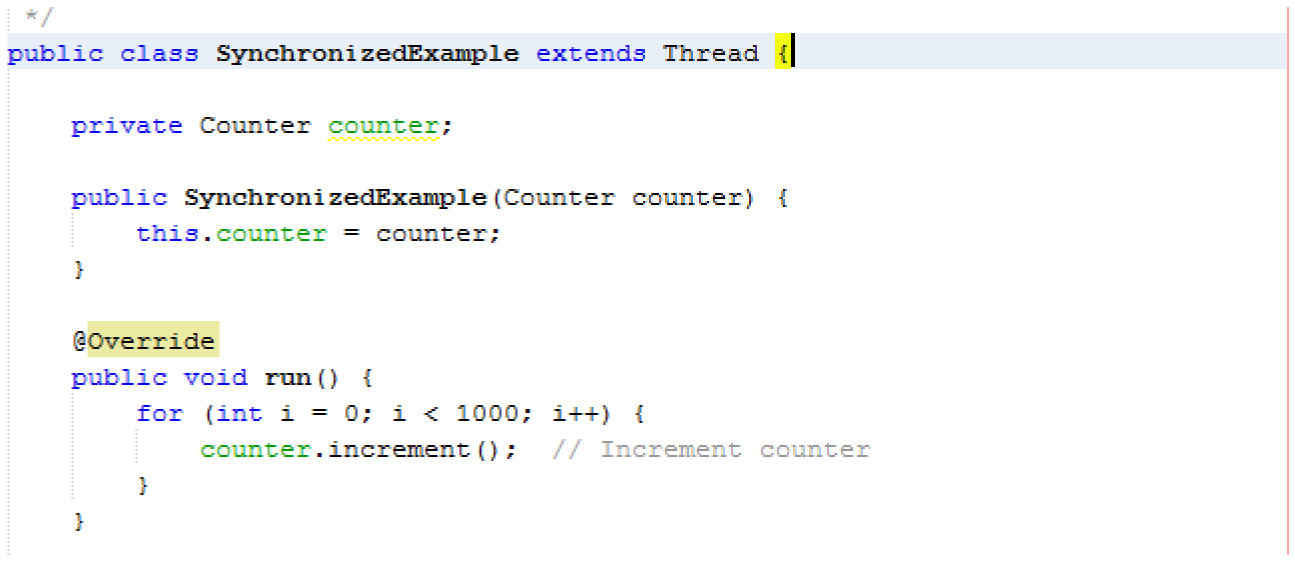
```
 * @author student
 */
public class RunnableTask implements Runnable {

    @Override
    public void run() {

        System.out.println(Thread.currentThread().getId() + " is executing the runnable task.");
    }

    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();
```

```
run:
10 is executing the runnable task.
11 is executing the runnable task.
BUILD SUCCESSFUL (total time: 0 seconds)
```

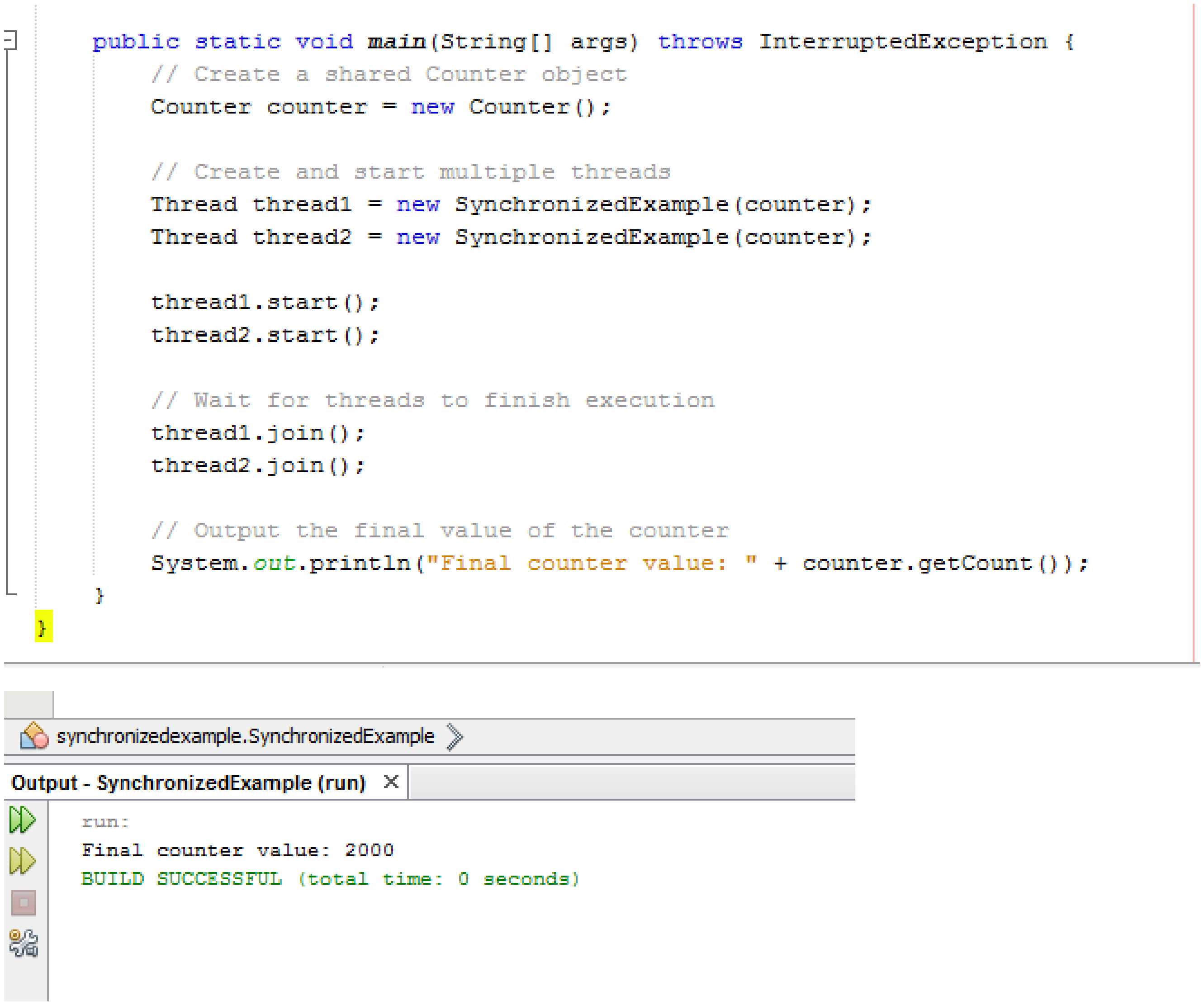
3. Synchronizing Shared Resources



```java
/**
 *
 * @author student
 */
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;
    }
}
```



```java
 */
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();  // Increment counter
        }
    }
```

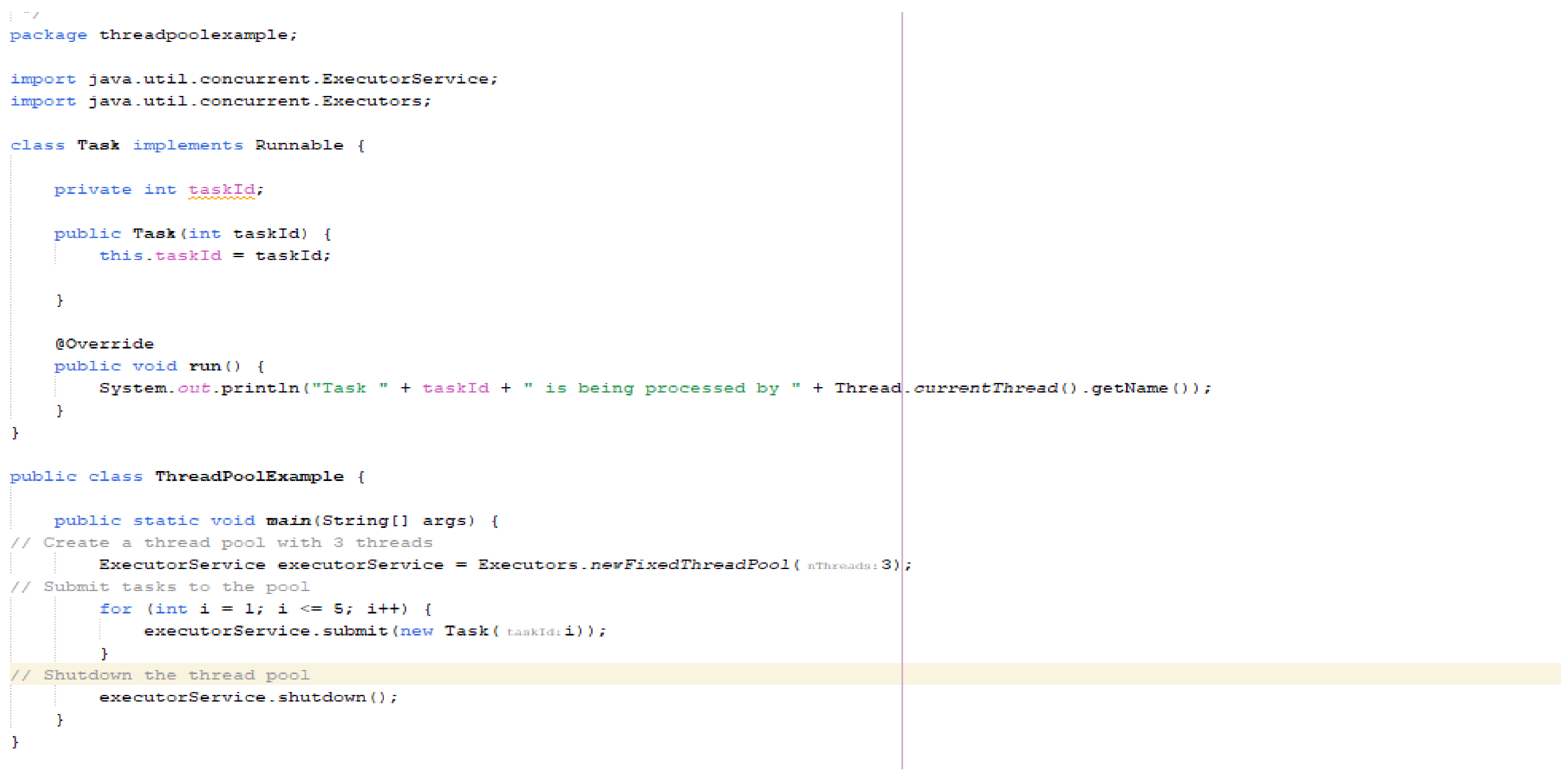


```
    public static void main(String[] args) throws InterruptedException {
        // Create a shared Counter object
        Counter counter = new Counter();

        // Create and start multiple threads
        Thread thread1 = new SynchronizedExample(counter);
        Thread thread2 = new SynchronizedExample(counter);

        thread1.start();
        thread2.start();

        // Wait for threads to finish execution
        thread1.join();
        thread2.join();

        // Output the final value of the counter
        System.out.println("Final counter value: " + counter.getCount());
    }
}
```

synchronizedexample.SynchronizedExample
Output - SynchronizedExample (run)

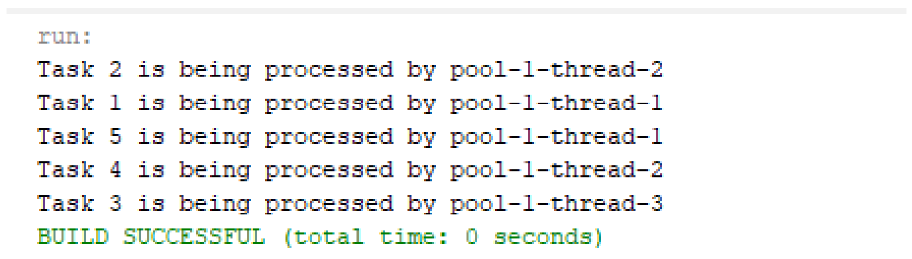
```
run:
Final counter value: 2000
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Using ExecutorService for Thread Pooling



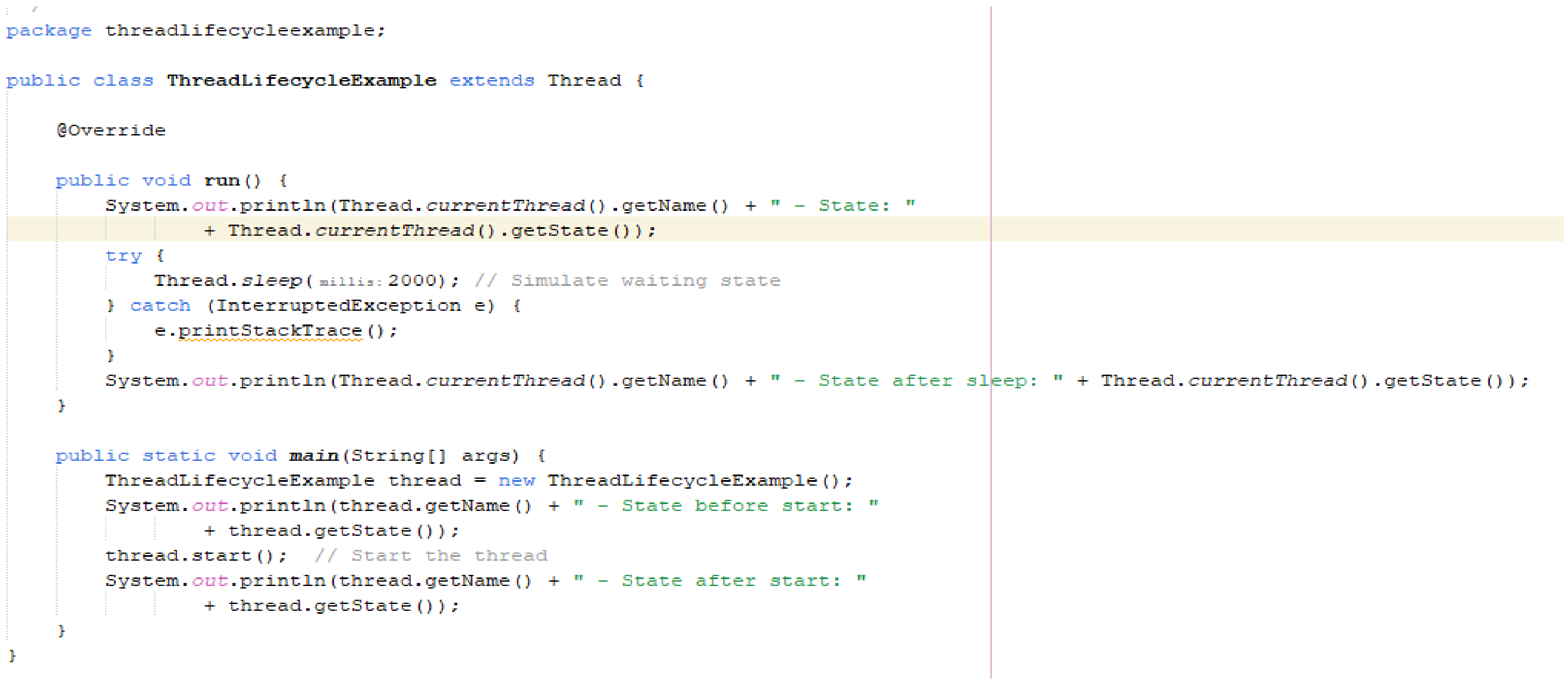
```java
 */
package threadpoolexample;

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

class Task implements Runnable {

    private int taskId;

    public Task(int taskId) {
        this.taskId = taskId;

    }

    @Override
    public void run() {
        System.out.println("Task " + taskId + " is being processed by " + Thread.currentThread().getName());
    }
}

public class ThreadPoolExample {

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

Output



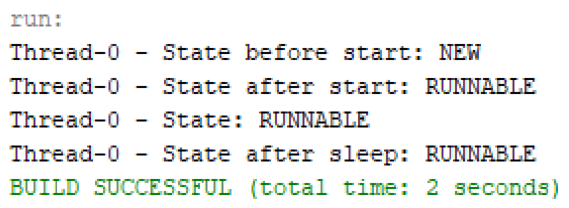
```
run:
Task 2 is being processed by pool-1-thread-2
Task 1 is being processed by pool-1-thread-1
Task 5 is being processed by pool-1-thread-1
Task 4 is being processed by pool-1-thread-2
Task 3 is being processed by pool-1-thread-3
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Thread Lifecycle Example



```java
package threadlifecycleexample;

public class ThreadLifecycleExample extends Thread {

    @Override

    public void run() {
        System.out.println(Thread.currentThread().getName() + " - State: "
                + Thread.currentThread().getState());
        try {
            Thread.sleep( millis: 2000); // Simulate waiting state
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(Thread.currentThread().getName() + " - State after sleep: " + Thread.currentThread().getState());
    }

    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



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