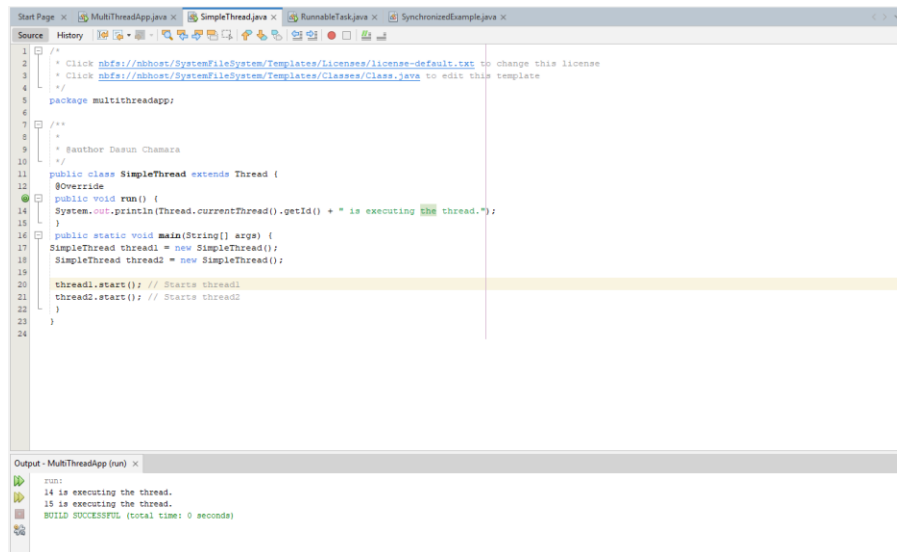


1. Create a Simple Thread Class

```
public class SimpleThread extends Thread {  
    @Override  
    public void run() {  
        System.out.println(Thread.currentThread().getId() + " is executing  
the thread.");  
    }  
    public static void main(String[] args) {  
  
        SimpleThread thread1 = new SimpleThread();  
  
        SimpleThread thread2 = new SimpleThread();  
  
        thread1.start(); // Starts thread1  
        thread2.start(); // Starts thread2  
    }  
}
```



The screenshot shows an IDE with the following components:

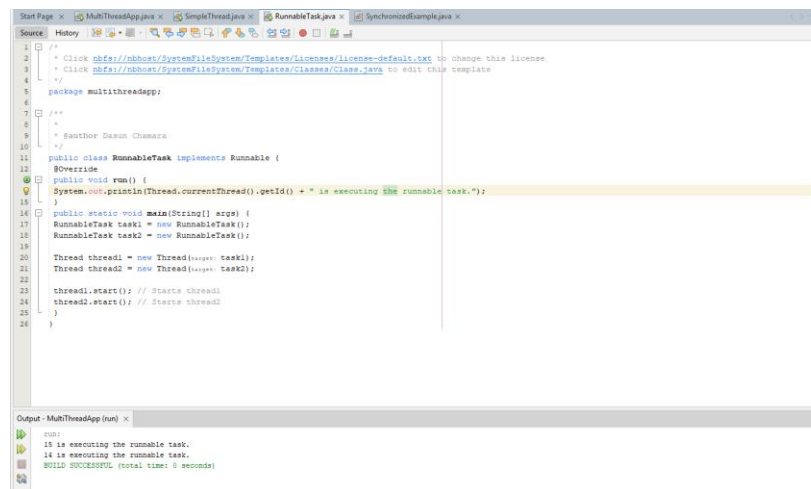
- Source Editor:** Displays the code for `SimpleThread.java`. The code is as follows:

```
1  /**  
2   * Click nbfs://nbhost/SystemFileSystem/Template/License/license-default.txt to change this license  
3   * Click nbfs://nbhost/SystemFileSystem/Template/Classes/Class.java to edit this template  
4   */  
5  package multithreadapp;  
6  
7  /**  
8   *  
9   * @author Dasun Chamara  
10  */  
11 public class SimpleThread extends Thread {  
12     @Override  
13     public void run() {  
14         System.out.println(Thread.currentThread().getId() + " is executing the thread.");  
15     }  
16  
17     public static void main(String[] args) {  
18         SimpleThread thread1 = new SimpleThread();  
19         SimpleThread thread2 = new SimpleThread();  
20  
21         thread1.start(); // Starts thread1  
22         thread2.start(); // Starts thread2  
23     }  
24 }
```
- Output Console:** Shows the output of the program:

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

2. Create a Runnable Class

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

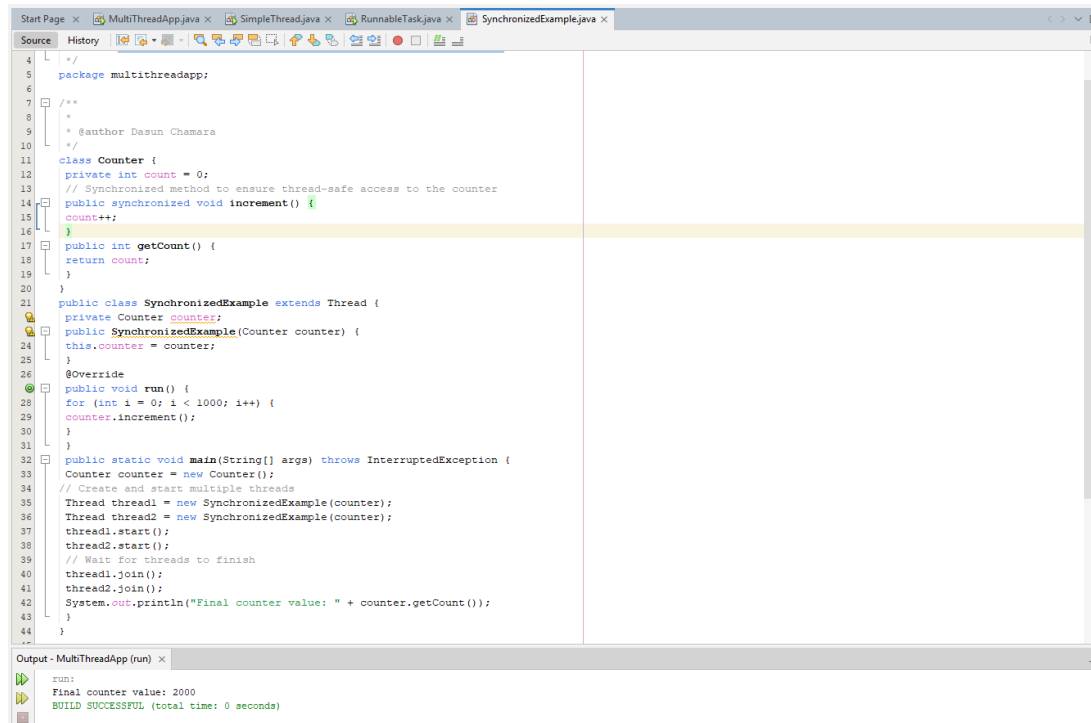


3. Synchronizing Shared Resources

```
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();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        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
        thread1.join();
        thread2.join();
        System.out.println("Final counter value: " + counter.getCount());
    }
}
```



4. Thread Pooling

```
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(3);

        // Submit tasks to the pool

        for (int i = 1; i <= 5; i++) {

            executorService.submit(new Task(i));

        }

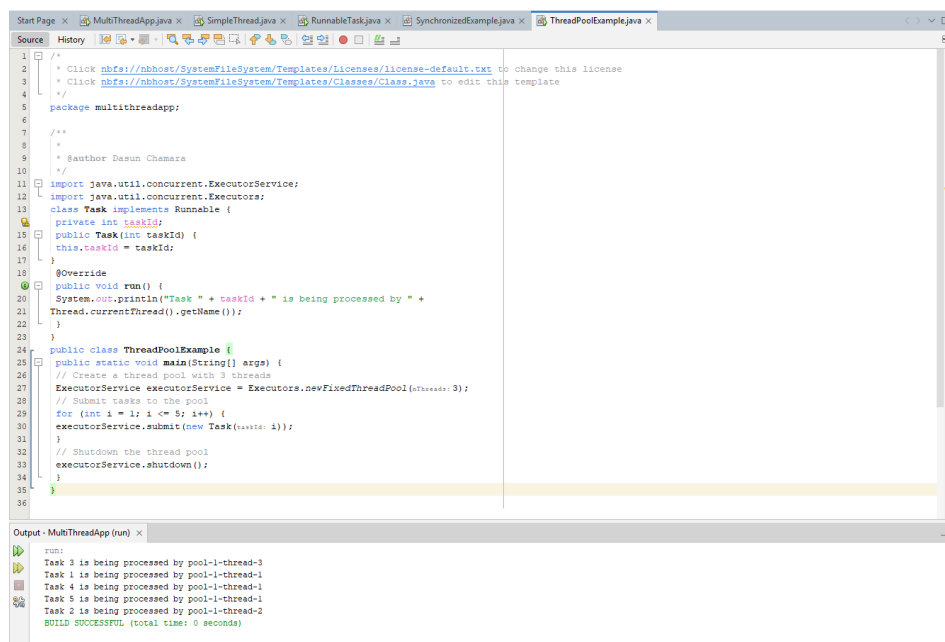
        // Shutdown the thread pool

        executorService.shutdown();

    }

}

```



The screenshot shows an IDE with multiple tabs. The active tab is 'ThreadPoolExample.java'. The code in the editor matches the code block above. The 'Output' window at the bottom shows the following text:

```

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

```

5. 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 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());  
    }  
}
```

}

The screenshot shows an IDE with several tabs open: Start Page, MultiThreadApp.java, SimpleThread.java, RunnableTask.java, SynchronizedExample.java, ThreadPoolExample.java, and ThreadLifecycleExample.java. The active tab is ThreadLifecycleExample.java, which contains the following code:

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package multithreadapp;
6
7  /**
8   *
9   * @author Dasun Chamara
10  */
11  public class ThreadLifecycleExample extends Thread {
12      @Override
13      public void run() {
14          System.out.println(Thread.currentThread().getName() + " - State: " +
15              Thread.currentThread().getState());
16          try {
17              Thread.sleep(2000); // Simulate waiting state
18          } catch (InterruptedException e) {
19              e.printStackTrace();
20          }
21          System.out.println(Thread.currentThread().getName() + " - State after sleep: " + Thread.currentThread().getState());
22      }
23      public static void main(String[] args) {
24          ThreadLifecycleExample thread = new ThreadLifecycleExample();
25          System.out.println(thread.getName() + " - State before start: " +
26              thread.getState());
27          thread.start(); // Start the thread
28          System.out.println(thread.getName() + " - State after start: " +
29              thread.getState());
30      }
31  }
```

The output window at the bottom shows the following output:

```
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)
```

The status bar at the bottom right shows "33:1 INS".