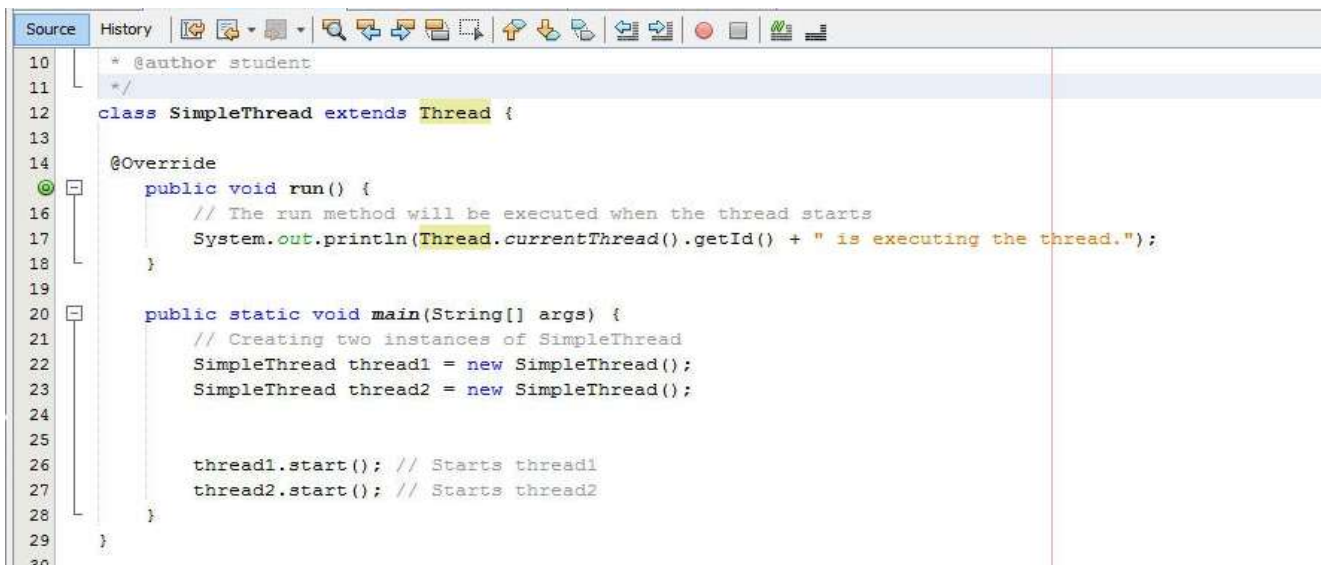


1. Create a Simple Thread Class

```
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();  
        thread2.start();  
    }  
}
```

Code:



```
Source History  
10 * @author student  
11 */  
12 class SimpleThread extends Thread {  
13  
14     @Override  
15     public void run() {  
16         // The run method will be executed when the thread starts  
17         System.out.println(Thread.currentThread().getId() + " is executing the thread.");  
18     }  
19  
20     public static void main(String[] args) {  
21         // Creating two instances of SimpleThread  
22         SimpleThread thread1 = new SimpleThread();  
23         SimpleThread thread2 = new SimpleThread();  
24  
25         thread1.start(); // Starts thread1  
26         thread2.start(); // Starts thread2  
27     }  
28 }  
29  
30
```

Output:



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

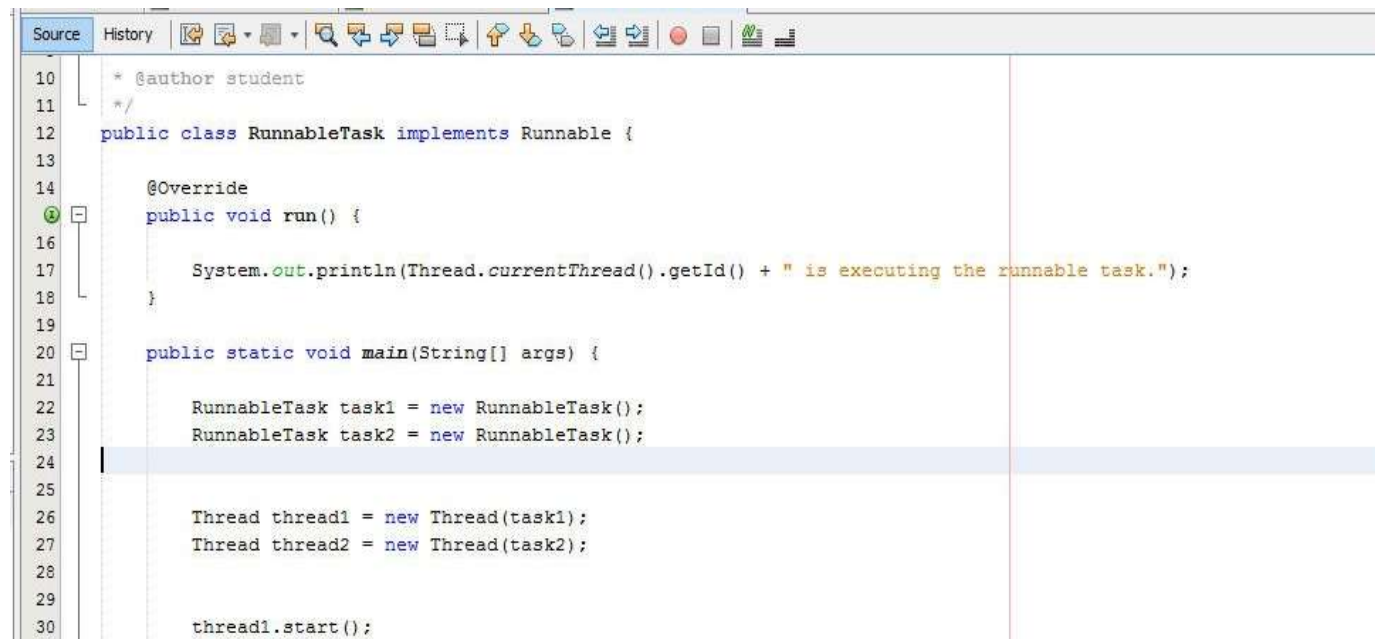
2. Create a Runnable Class

```
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);

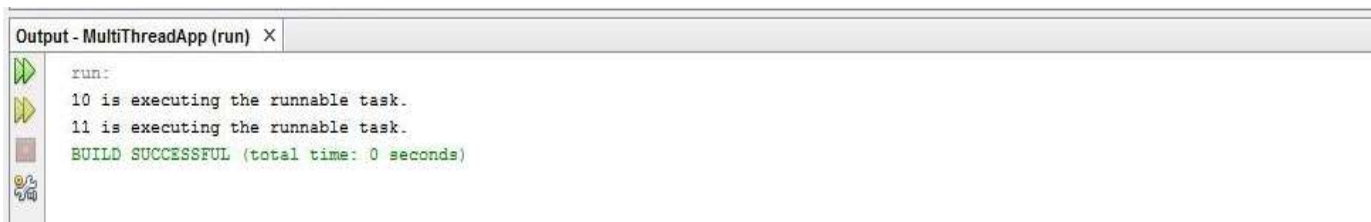
        thread1.start(); // Starts thread1
        thread2.start(); // Starts thread2
    }
}
```

Code:

A screenshot of an IDE window titled 'Source'. The code is for a class named 'RunnableTask' that implements the 'Runnable' interface. It includes an '@Override' annotation for the 'run()' method, which prints the current thread's ID and the message 'is executing the runnable task.'. The 'main' method creates two instances of 'RunnableTask', wraps each in a 'Thread' object, and starts both threads. The code is as follows:

```
10  * @author student
11  */
12  public class RunnableTask implements Runnable {
13
14      @Override
15      public void run() {
16
17          System.out.println(Thread.currentThread().getId() + " is executing the runnable task.");
18      }
19
20      public static void main(String[] args) {
21
22          RunnableTask task1 = new RunnableTask();
23          RunnableTask task2 = new RunnableTask();
24
25
26          Thread thread1 = new Thread(task1);
27          Thread thread2 = new Thread(task2);
28
29
30          thread1.start();
```

Output:

A screenshot of the IDE's output window, titled 'Output - MultiThreadApp (run) X'. It shows the output of the program: 'run:', '10 is executing the runnable task.', '11 is executing the runnable task.', and 'BUILD SUCCESSFUL (total time: 0 seconds)'. The output is as follows:

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

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());
    }
}
```

```
/**
 *
 * @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;
    }
}
```

```

*/
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{

        Counter counter = new Counter();

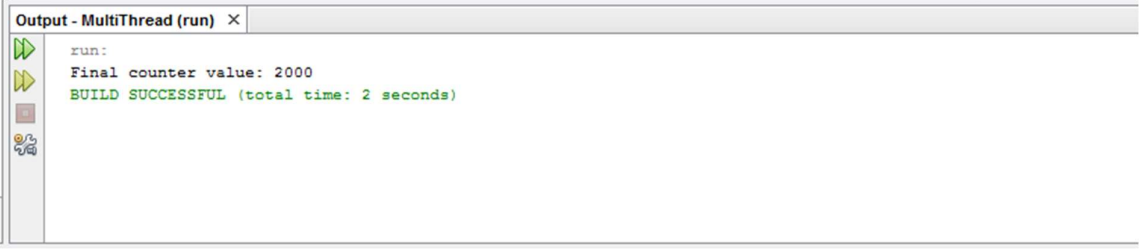
        // Create and start multiple threads
        Thread thread1 = new SynchronizedExample(counter);
        Thread thread2 = new SynchronizedExample(counter);
        thread1.start();
        thread2.start();

        // Wait for threads to finish
        thread1.join();
        thread2.join();
        System.out.println("Final counter value: " + counter.getCount());

    }
}

```

Output:



```
run:
Final counter value: 2000
BUILD SUCCESSFUL (total time: 2 seconds)
```

4. Using ExecutorService for 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;
}

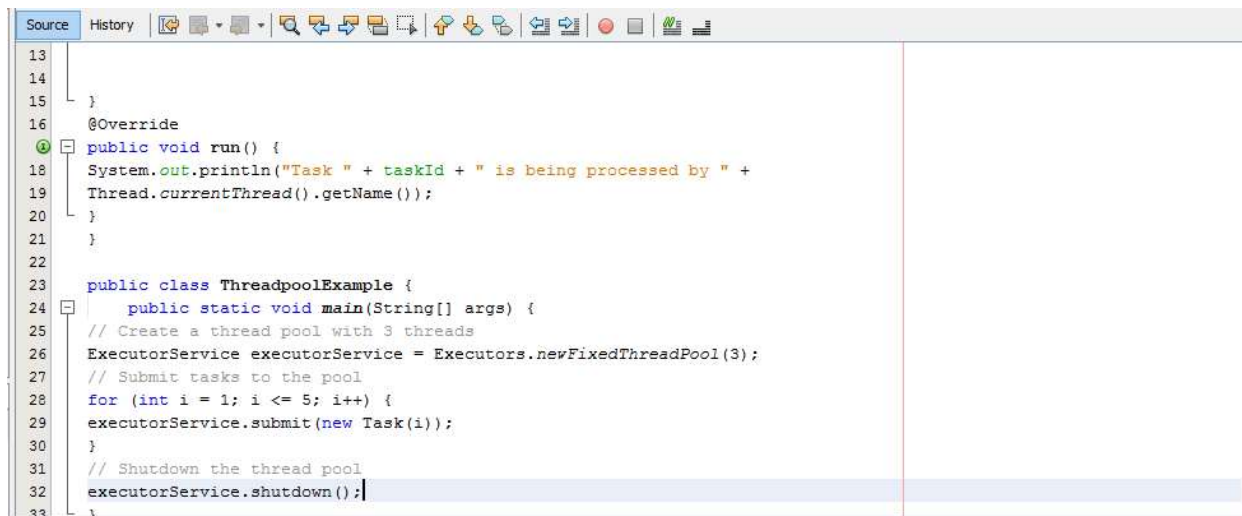
@Override
public void run() {
    System.out.println("Task " + taskId + " is being processed by " +
Thread.currentThread().getName());
}
}

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));
        }

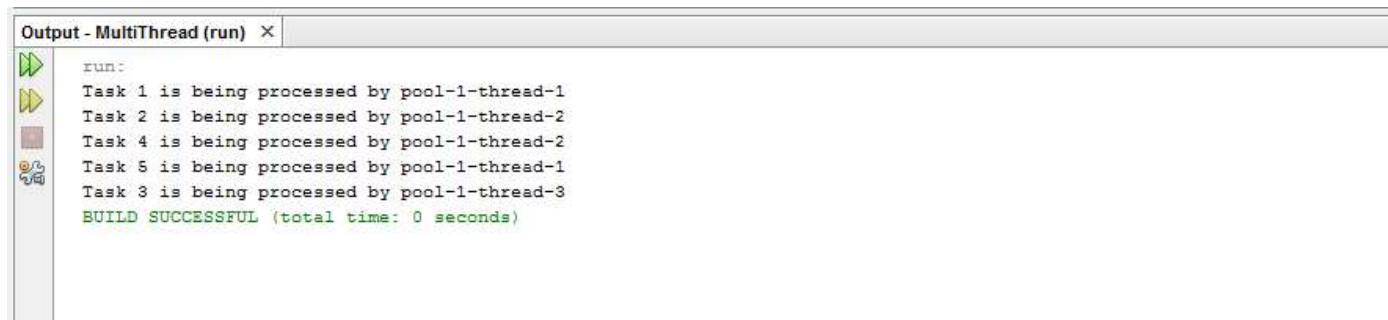
        // Shutdown the thread pool
        executorService.shutdown();
    }
}
```

Code:



```
13  
14  
15 }  
16 @Override  
17 public void run() {  
18     System.out.println("Task " + taskId + " is being processed by " +  
19     Thread.currentThread().getName());  
20 }  
21 }  
22  
23 public class ThreadpoolExample {  
24     public static void main(String[] args) {  
25         // Create a thread pool with 3 threads  
26         ExecutorService executorService = Executors.newFixedThreadPool(3);  
27         // Submit tasks to the pool  
28         for (int i = 1; i <= 5; i++) {  
29             executorService.submit(new Task(i));  
30         }  
31         // Shutdown the thread pool  
32         executorService.shutdown();  
33     }
```

Output:



```
Output - MultiThread (run) X  
run:  
Task 1 is being processed by pool-1-thread-1  
Task 2 is being processed by pool-1-thread-2  
Task 4 is being processed by pool-1-thread-2  
Task 5 is being processed by pool-1-thread-1  
Task 3 is being processed by pool-1-thread-3  
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());

    }
}
```


Code:

```
package threadlifecycleexample;

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());
    }
}
```

Output:



```
Output - MultiThread (run) X
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)
```