

Multi-threaded Java Application

1. Create a Simple Thread Class

```
package multithreadapp2;
```

```
public class SimpleThread extends Thread{
```

```
    @Override
```

```
    public void run() {
```

```
        System.out.println(Thread.currentThread().getId() + " is executing the thread.");
```

```
    }
```

Main Method:

```
public static void main(String[] args) {
```

```
    SimpleThread thread1 = new SimpleThread();
```

```
    SimpleThread thread2 = new SimpleThread();
```

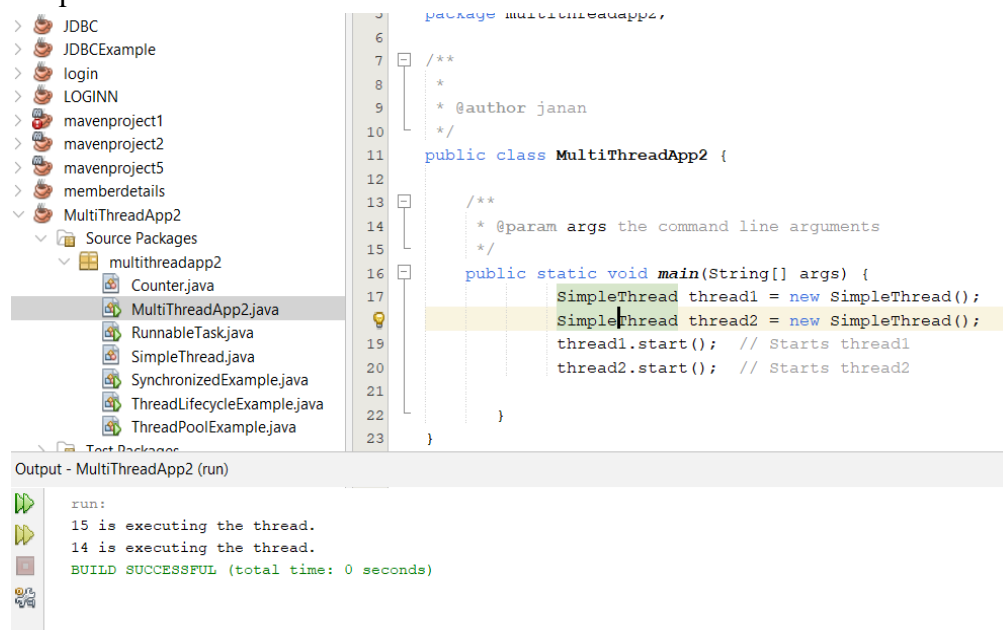
```
    thread1.start(); // Starts thread1
```

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

```
}
```

```
}
```

Output:



2. Create a Runnable Class

```
package multithreadapp2;
```

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

Main Method:

```
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:



```
8      *  
9      * @author janan  
10     */  
11     public class MultiThreadApp2 {  
12  
13         /*SimpleThread thread1 = new SimpleThread();  
14         SimpleThread thread2 = new SimpleThread();  
15         thread1.start(); // Starts thread1  
16         thread2.start(); // Starts thread2  
17  
18         */  
19  
20     public static void main(String[] args) {  
21         RunnableTask task1 = new RunnableTask();  
22         RunnableTask task2 = new RunnableTask();  
23         Thread thread1 = new Thread(target:task1);  
24         Thread thread2 = new Thread(target:task2);  
25         thread1.start(); // Starts thread1  
26         thread2.start(); // Starts thread2  
27     }  
28 }
```

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

3. Synchronizing Shared Resources

Counter class:

```
package multithreadapp2;
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;
    }
}
```

SynchronizedExample class:

```
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();
        }
    }
}
```

Main Method:

```
public static void main(String[] args) throws InterruptedException {
    Counter counter = new Counter();

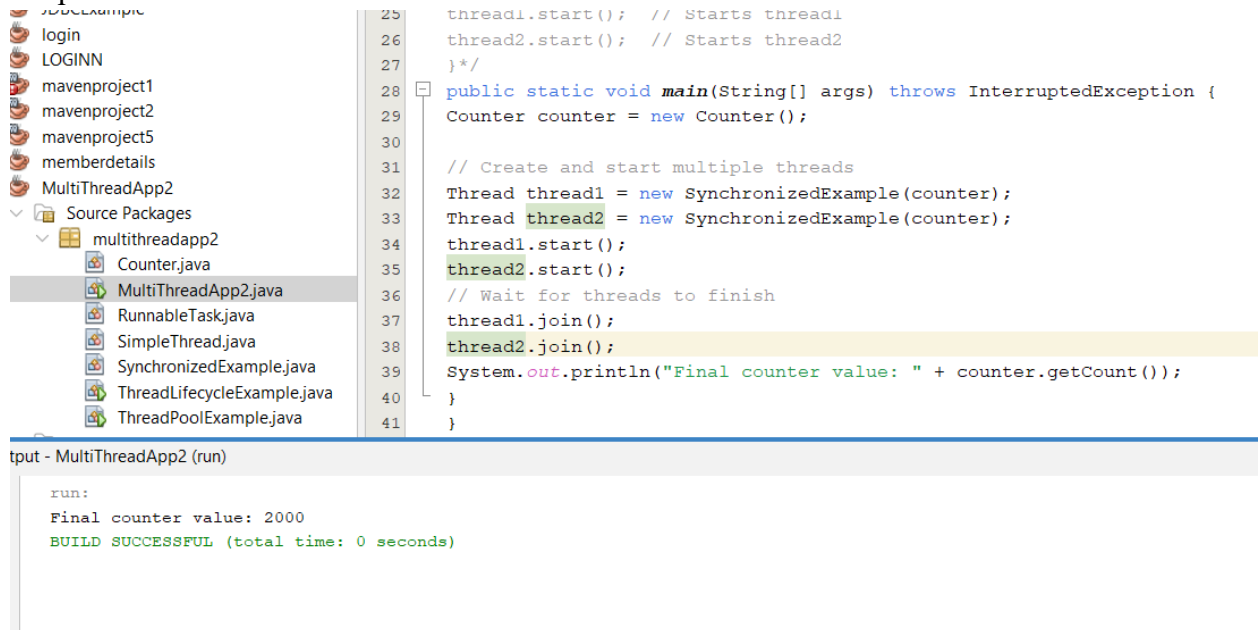
    // Create and start multiple threads
    Thread thread1 = new SynchronizedExample(counter);
    Thread thread2 = new SynchronizedExample(counter);
    thread1.start();
```

```

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

```

Output:



The screenshot shows an IDE with a project structure on the left, source code in the center, and a terminal output at the bottom.

Project Structure (Left):

- login
- LOGINN
- mavenproject1
- mavenproject2
- mavenproject5
- memberdetails
- MultiThreadApp2
 - Source Packages
 - multithreadapp2
 - Counter.java
 - MultiThreadApp2.java
 - RunnableTask.java
 - SimpleThread.java
 - SynchronizedExample.java
 - ThreadLifecycleExample.java
 - ThreadPoolExample.java

Source Code (Center):

```

25 thread1.start(); // Starts thread1
26 thread2.start(); // Starts thread2
27 */
28 public static void main(String[] args) throws InterruptedException {
29     Counter counter = new Counter();
30
31     // Create and start multiple threads
32     Thread thread1 = new SynchronizedExample(counter);
33     Thread thread2 = new SynchronizedExample(counter);
34     thread1.start();
35     thread2.start();
36     // Wait for threads to finish
37     thread1.join();
38     thread2.join();
39     System.out.println("Final counter value: " + counter.getCount());
40 }
41 }

```

Terminal Output (Bottom):

```

tput - MultiThreadApp2 (run)

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

```

4. Using ExecutorService for Thread Pooling

```
package multithreadapp2;
```

```
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());
}
}

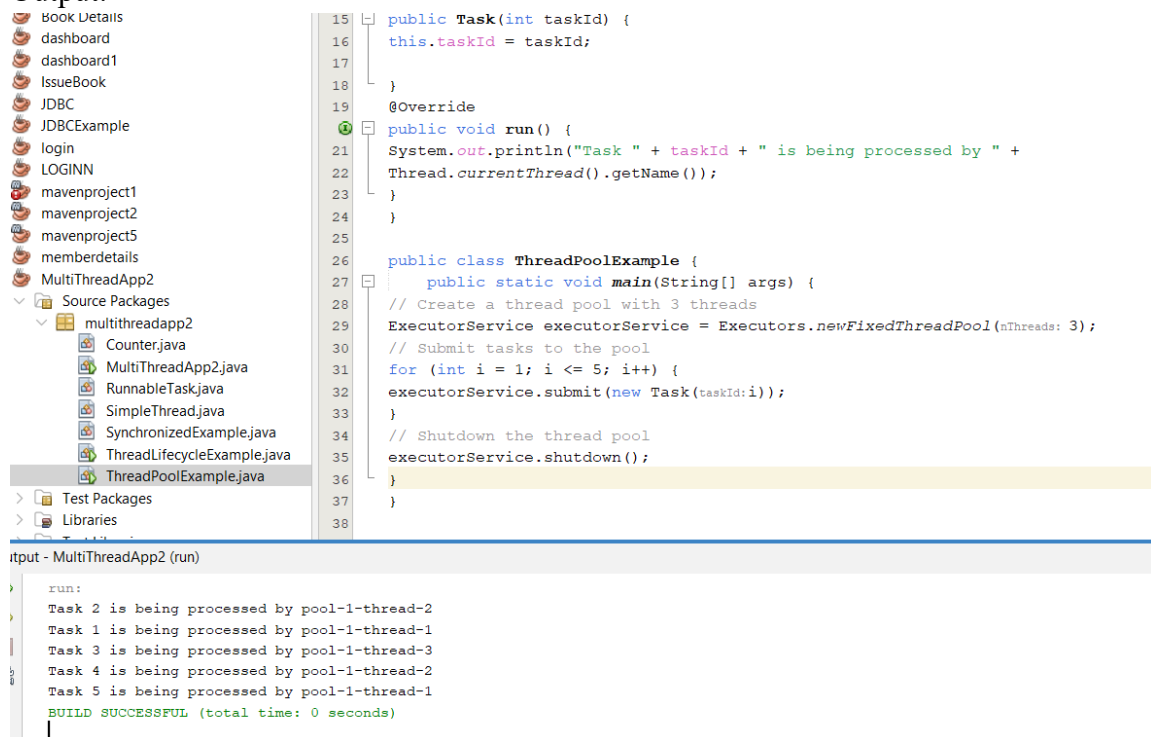
```

```

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();
    }
}

```

Output:



The screenshot shows an IDE with a project named 'MultiThreadApp2'. The 'Source Packages' view on the left shows a package 'multithreadapp2' containing several Java files, including 'ThreadPoolExample.java'. The main editor displays the code for 'Task' and 'ThreadPoolExample' classes. The 'Task' class has a 'run()' method that prints the task ID and the thread name. The 'ThreadPoolExample' class has a 'main()' method that creates a thread pool with 3 threads, submits 5 tasks, and then shuts down the pool. The output window at the bottom shows the results of running the application, displaying the task IDs and the thread names that processed them. The output is as follows:

```

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

```

5. Thread Lifecycle Example

```
package multithreadapp2;

public class ThreadLifecycleExample extends Thread{

    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:

The screenshot shows an IDE with a project explorer on the left, a code editor in the center, and an output console at the bottom.

Project Explorer: The left sidebar shows a project structure with folders like 'dashboard1', 'IssueBook', 'JDBC', 'JDBCExample', 'login', 'LOGINN', 'mavenproject1', 'mavenproject2', 'mavenproject5', 'memberdetails', 'MultiThreadApp2', 'Source Packages', 'Test Packages', and 'Libraries'. Under 'MultiThreadApp2', there is a sub-folder 'multithreadapp2' containing several Java files, including 'ThreadLifecycleExample.java' which is currently selected.

Code Editor: The center pane displays the code for 'ThreadLifecycleExample.java'. The code defines a class that extends 'Thread' and implements a 'run()' method. The 'run()' method prints the thread's state, sleeps for 2000 milliseconds, and prints the state again. A 'main' method creates an instance of 'ThreadLifecycleExample', starts it, and prints its state before and after starting.

```
9      * @author janan
10     */
11     public class ThreadLifecycleExample extends Thread{
12
13         public void run() {
14             System.out.println(Thread.currentThread().getName() + " - State: " + Thread.currentThread().getState());
15             try {
16                 Thread.sleep(2000); // Simulate waiting state
17             } catch (InterruptedException e) {
18                 e.printStackTrace();
19             }
20             System.out.println(Thread.currentThread().getName() + " - State after sleep: " + Thread.currentThread().getState());
21         }
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     }
```

Output Console: The bottom pane, titled 'Output - MultiThreadApp2 (run)', shows the execution output. It displays the thread's state at various points: before start (NEW), after start (RUNNABLE), and after sleep (RUNNABLE). The output concludes with 'BUILD SUCCESSFUL (total time: 2 seconds)'.

```
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)
```