GAM/IT/2022/F/0079 – J.S.Senarath

Multi-threaded Java Application

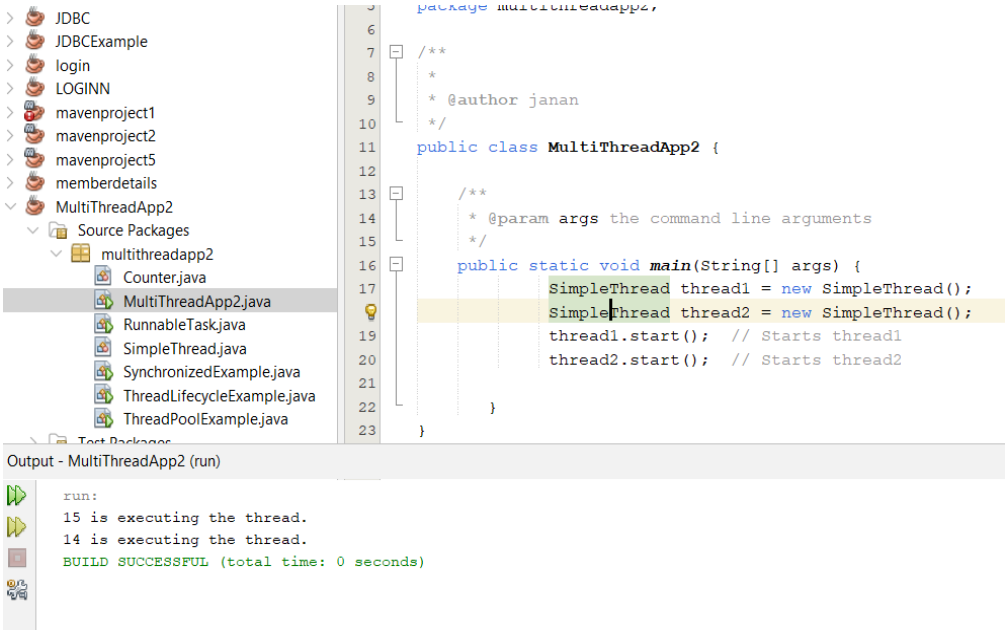
1. **Create a Simple Thread Class**

```
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:



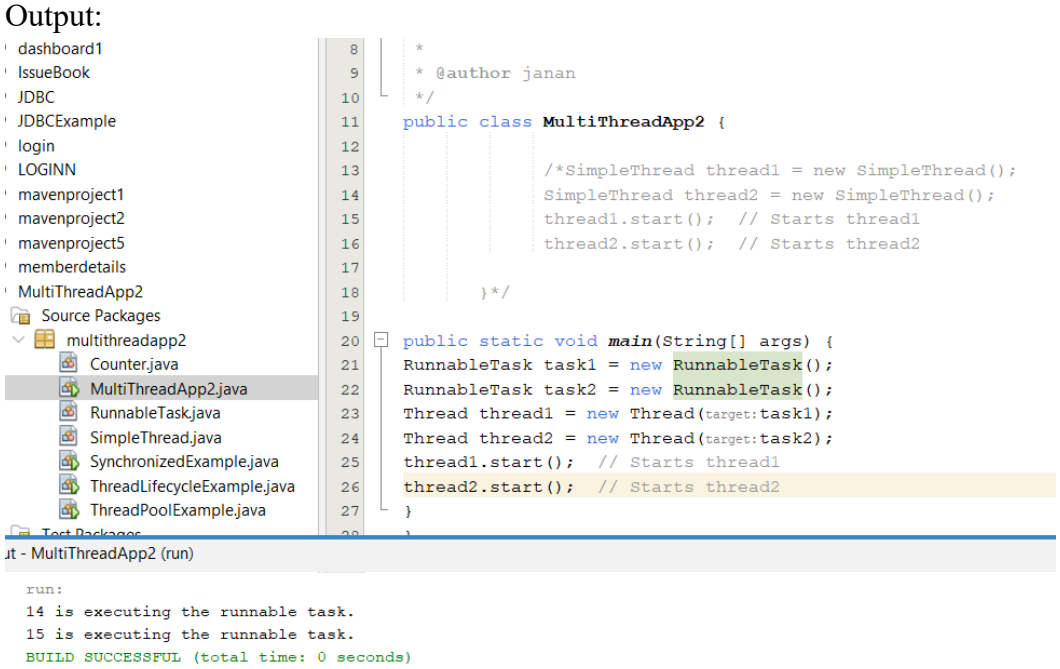
2. **Create a Runnable Class**

```
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:



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:

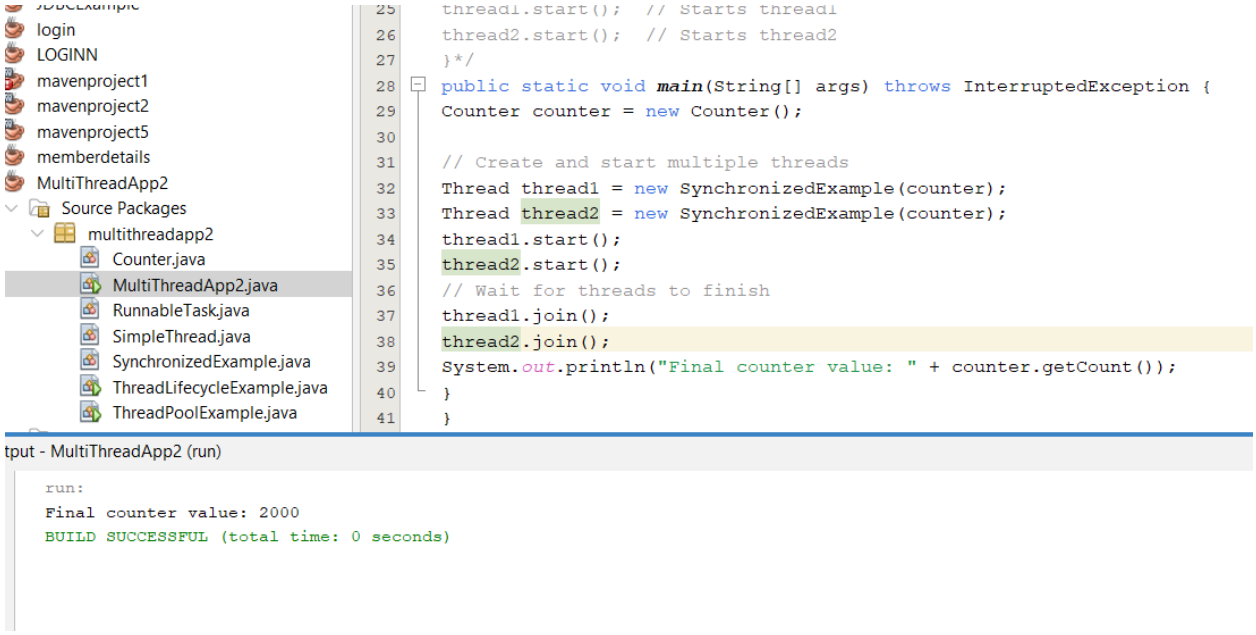
```
public class SynchronizedExample extends Thread {
private Counter counter;
public SynchronizedExample(Counter counter) {
this.counter = counter;
}
@Override
public void run() {
for (int i = 0; i < 1000; i++) {
counter.increment();
}
}
```

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:

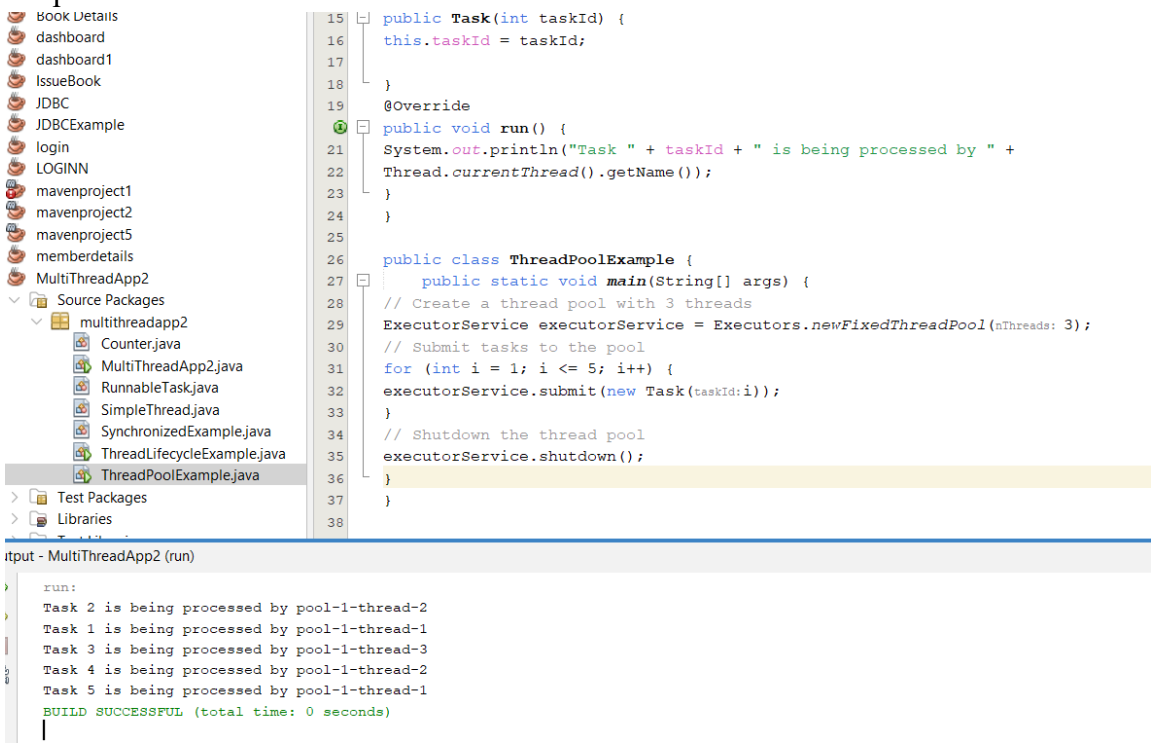


4. **Using ExecutorService for Thread Pooling**

```
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();
}
}
```

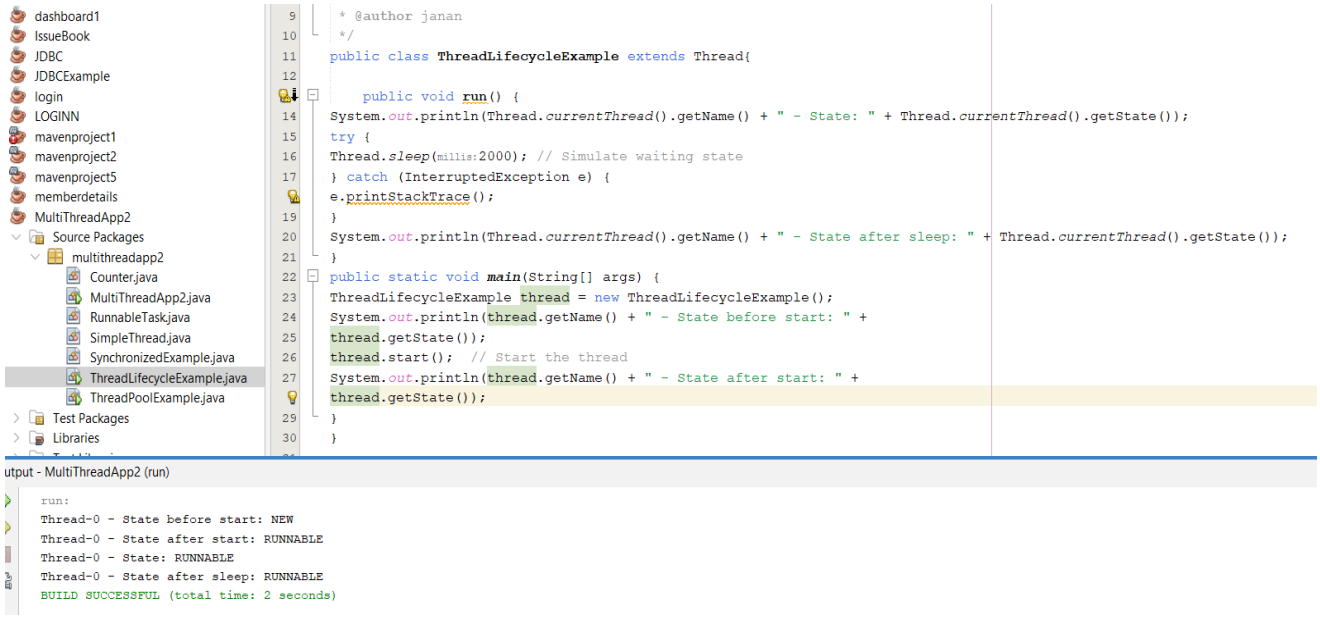
Output:



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:


dashboard1
IssueBook
JDBC
JDBCExample
login
LOGINN
mavenproject1
mavenproject2
mavenproject5
memberdetails
MultiThreadApp2
Source Packages
multithreadapp2
Counter.java
MultiThreadApp2.java
RunnableTask.java
SimpleThread.java
SynchronizedExample.java
ThreadLifecycleExample.java
ThreadPoolExample.java
Test Packages
Libraries
* @author janan
*/
public class ThreadLifecycleExample extends Thread{
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());
}
}
utput - MultiThreadApp2 (run)
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)