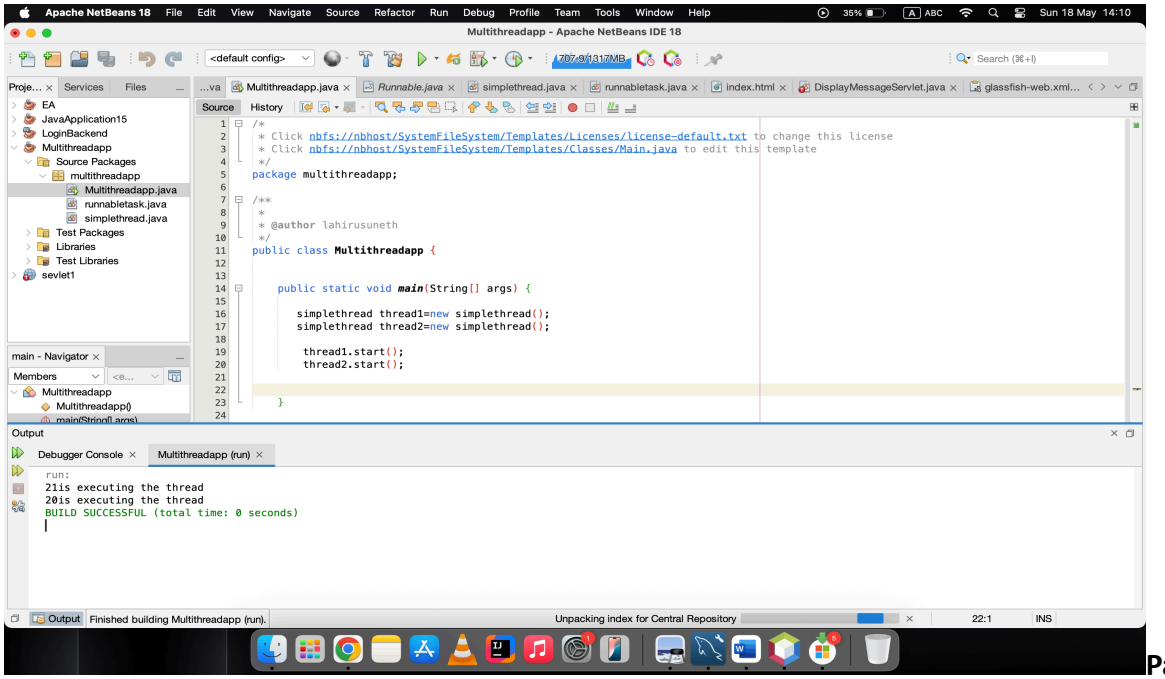
GAM/IT2022/F/0070

Multi Threaded Java Application

Part 01 - Create a simple thread class

```
public  class simplethread extends Thread{

    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();

 }
```

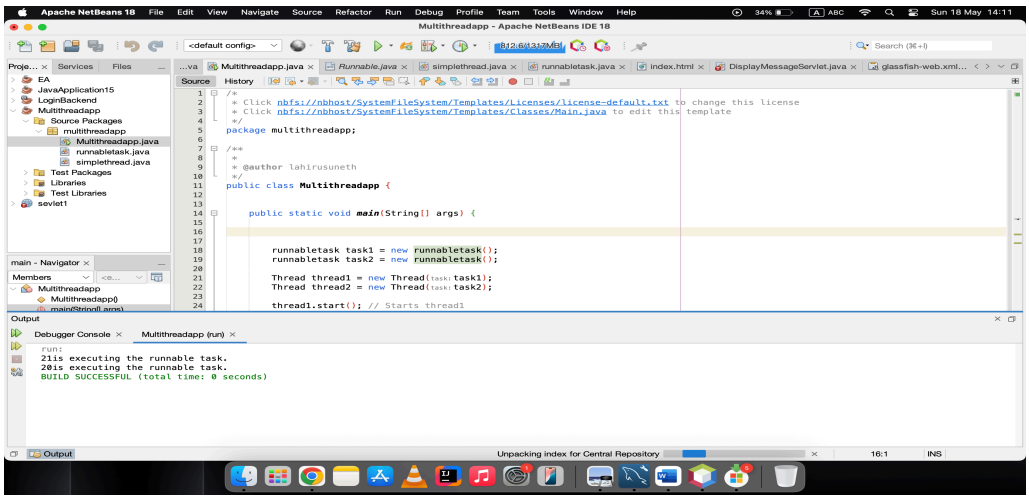


Part

2: Using Runnable Interface

- 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();


  }

 }
}
```

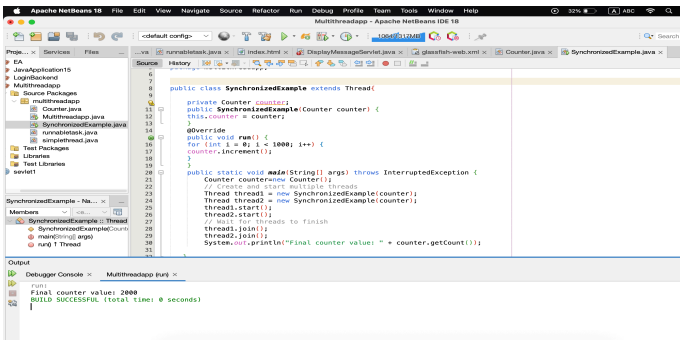


Part 3: Synchronizing Threads

- Synchronizing Shared Resources

```
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;
            }
       }
```

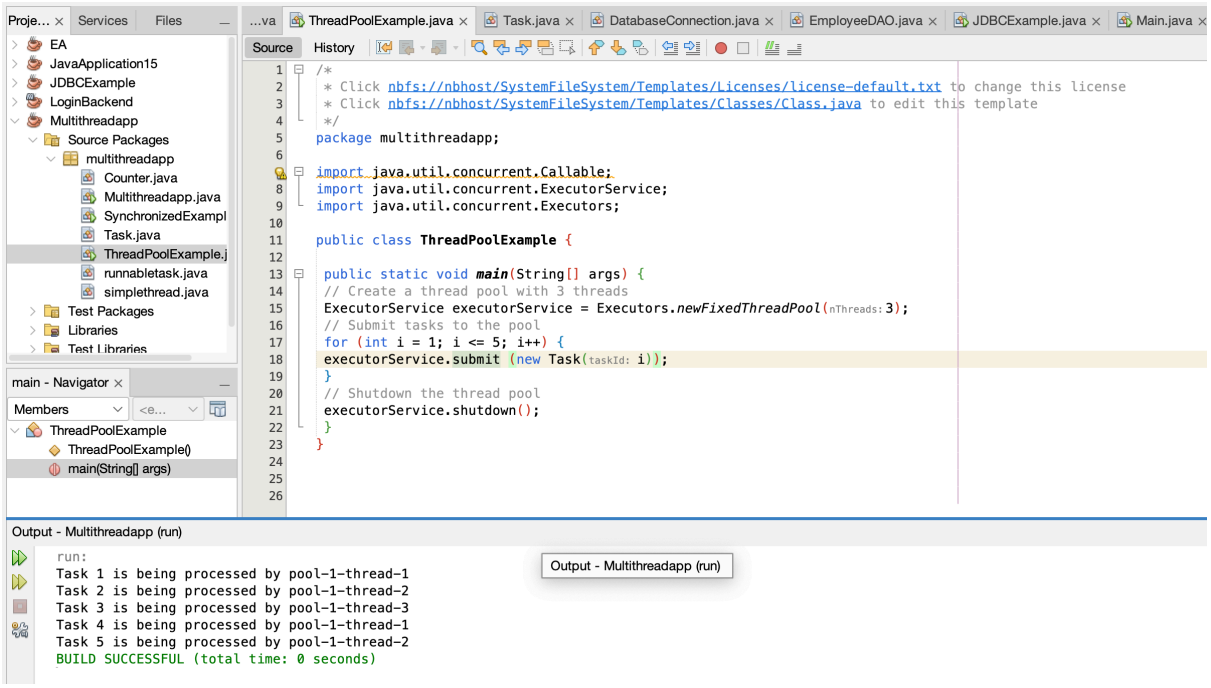
```
public class SynchronizedExample extends Thread{

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



Part 4: Thread Pooling

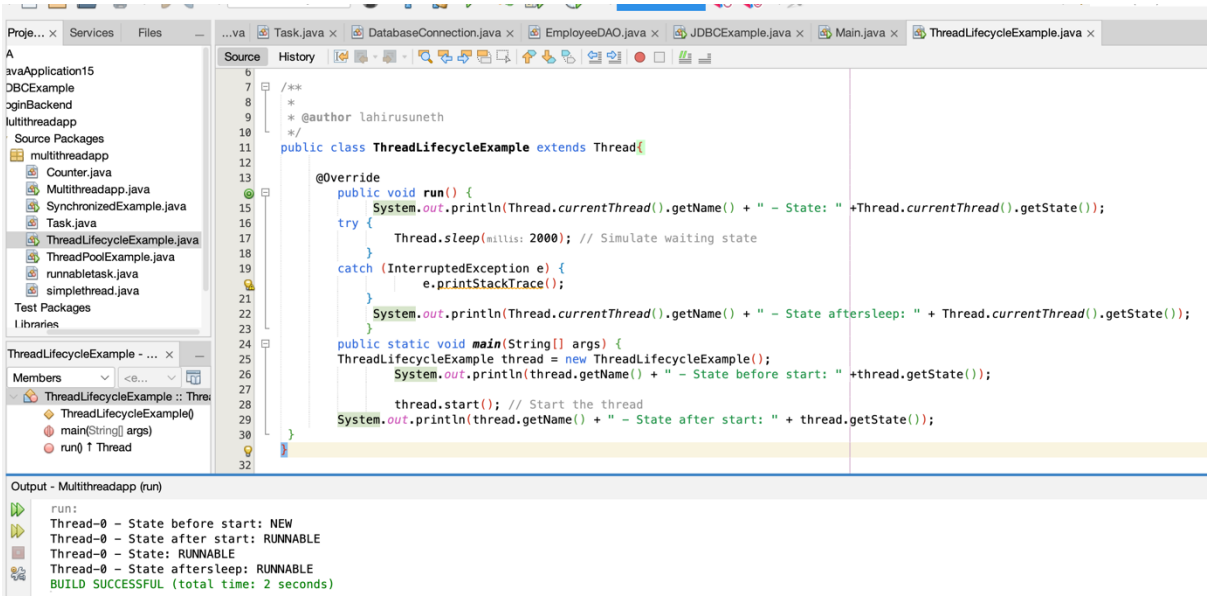
- Using ExecutorService for Thread Pooling

```
import java.io.IOException;
import java.nio.CharBuffer;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Task implements Runnable{

   private final 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 int read(CharBuffer cb) throws IOException {
      throw new UnsupportedOperationException("Not supported yet."); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
   }
}

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();
 }
}
```



Part 5: Thread Lifecycle and States

```
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 aftersleep: " +
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());
 }
}
```


Proje... Services Files
A
avaApplication15
DBCExample
oginBackend
ultithreadapp
Source Packages
multithreadapp
Counter.java
Multithreadapp.java
SynchronizedExample.java
Task.java
ThreadLifecycleExample.java
ThreadPoolExample.java
runnabletask.java
simplethread.java
Test Packages
Libraries
ThreadLifecycleExample - ...
Members
ThreadLifecycleExample :: Thra
ThreadLifecycleExample()
main(String[] args)
run() ↑ Thread
...va Task.java DatabaseConnection.java EmployeeDAO.java JDBCExample.java Main.java ThreadLifecycleExample.java
Source History
/**
*
* @author lahirusuneth
*/
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 aftersleep: " + 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 - Multithreadapp (run)
run:
Thread-0 - State before start: NEW
Thread-0 - State after start: RUNNABLE
Thread-0 - State: RUNNABLE
Thread-0 - State aftersleep: RUNNABLE
BUILD SUCCESSFUL (total time: 2 seconds)