Lab Sheet: Multi-threaded Java Application

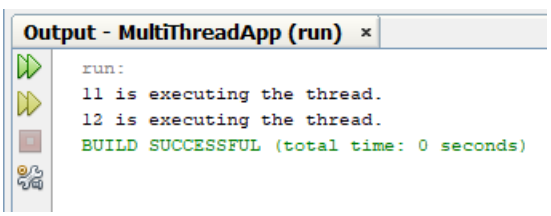
Part 1:

Introduction to Threads in Java

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
}
}
```



Part 2:

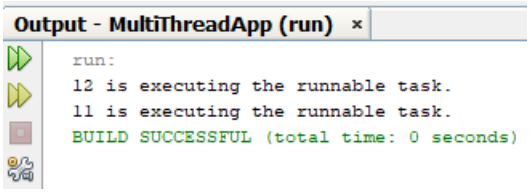
Using Runnable Interface

1. 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
}
}
```



Part 3:

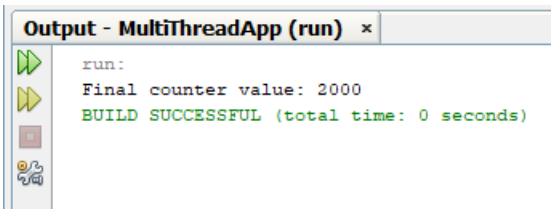
Synchronizing Threads

1. Create a new class called Counter.java to demonstrate synchronization with 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());
}
}
```



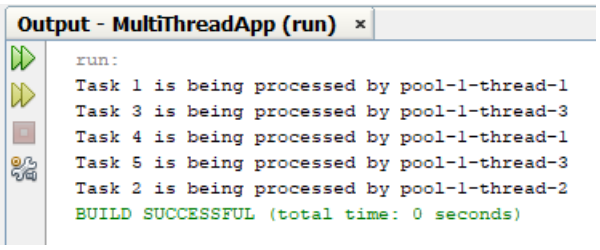
Part 4:

Thread Pooling

1. 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();
}
}
```



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 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());
}
}
```