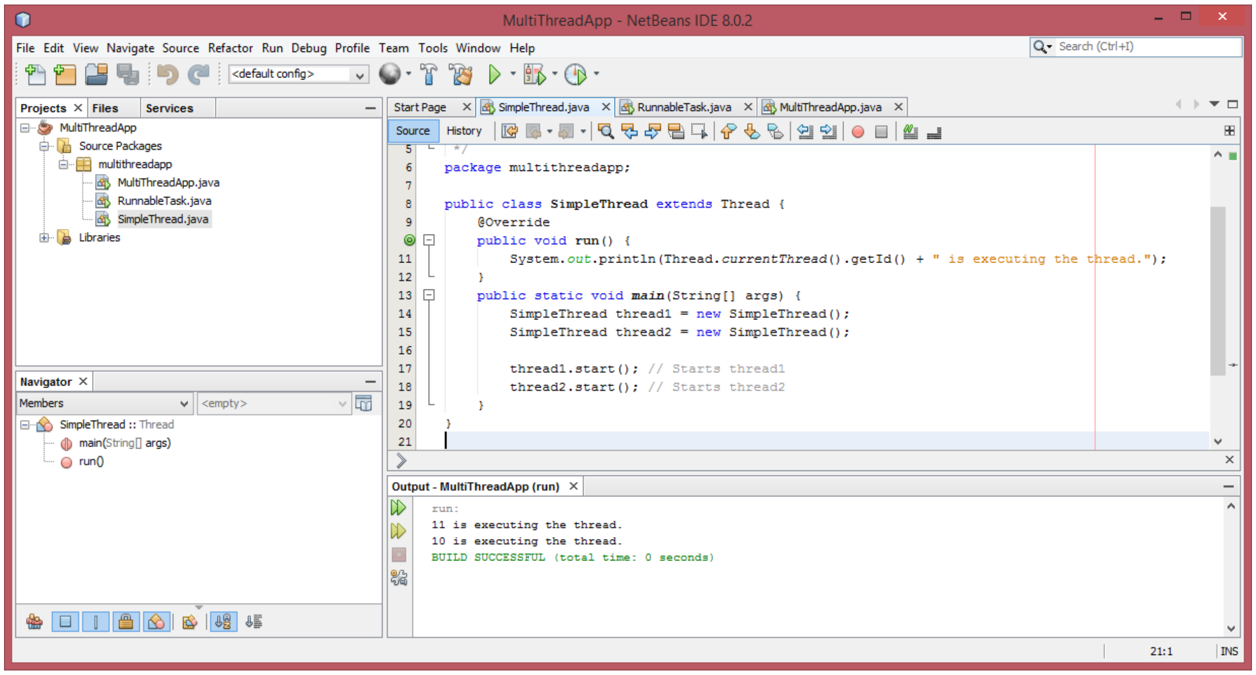
Lab Sheet 01

Task 01 - Create a Simple Thread Class

`SimpleThread.java`

```
package multithreadapp;
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
    }
}
```



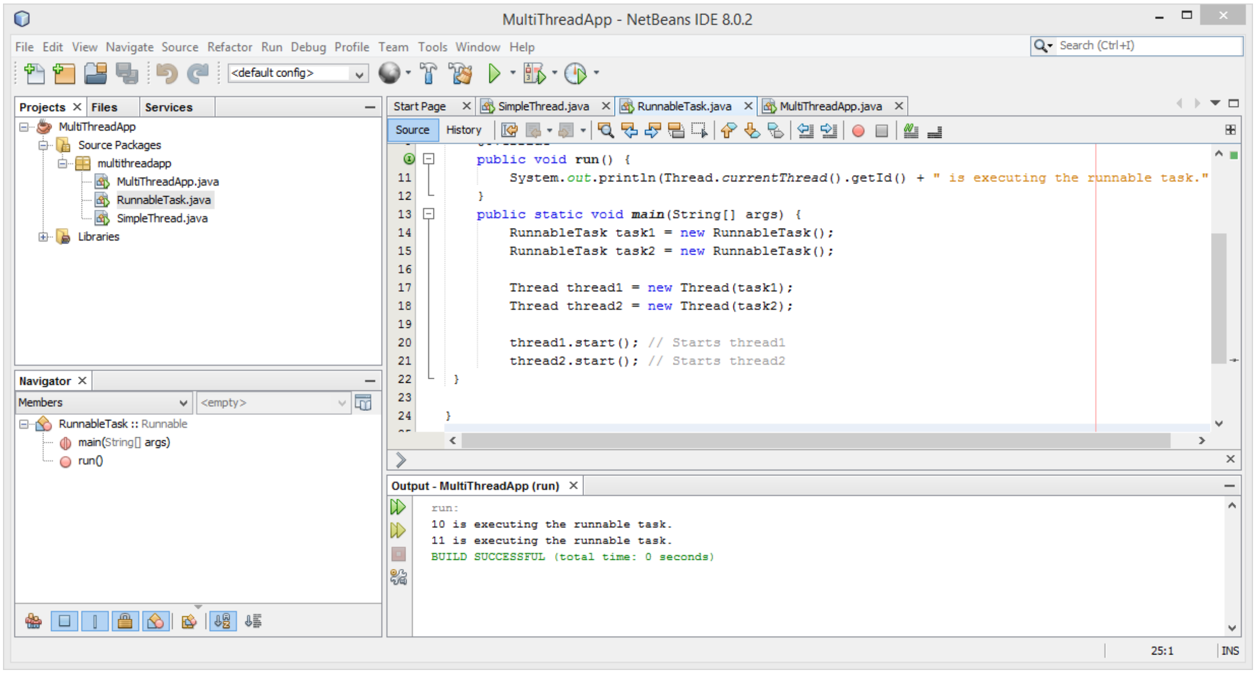
```
 */
package multithreadapp;

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
    }
}
```

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

Task 02 - Create a Runnable Class

`RunnableTask.java`

```
package multithreadapp;
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
 }
}
```



```
    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
    }

}
```

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

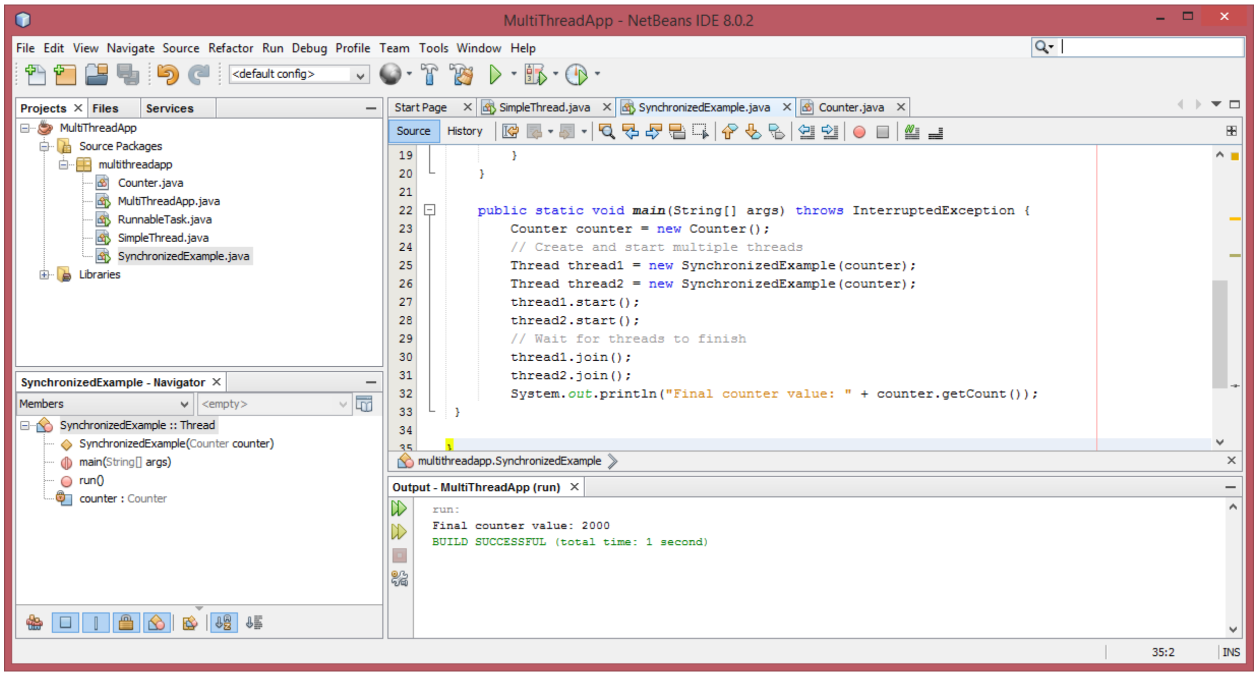
Task 03 - Synchronizing Shared Resources

Counter.java

```
package multithreadapp;
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.java

```
package multithreadapp;
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());
 }
}
```


MultiThreadApp - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects Files Services
MultiThreadApp
Source Packages
multithreadapp
Counter.java
MultiThreadApp.java
RunnableTask.java
SimpleThread.java
SynchronizedExample.java
Libraries
Start Page SimpleThread.java SynchronizedExample.java Counter.java
Source History
SynchronizedExample - Navigator
Members <empty>
SynchronizedExample :: Thread
SynchronizedExample(Counter counter)
main(String[] args)
run()
counter : Counter
multithreadapp.SynchronizedExample
Output - MultiThreadApp (run)
35:2 INS

```
        }
    }

    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());
 }

}
```

```
run:
Final counter value: 2000
BUILD SUCCESSFUL (total time: 1 second)
```

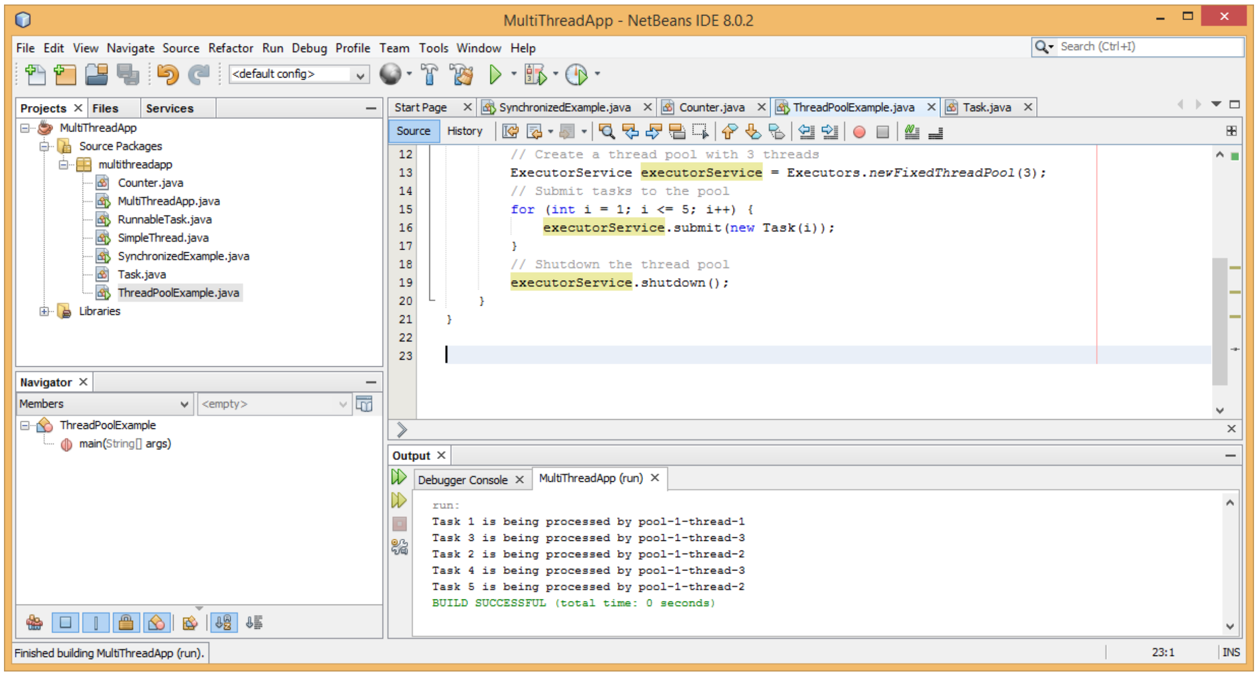
Task 04 - Using ExecutorService for Thread Pooling

Task.java

```
package multithreadapp;
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());
    }
}
```

ThreadPoolExample.java

```
package multithreadapp;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

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();
    }
}
```



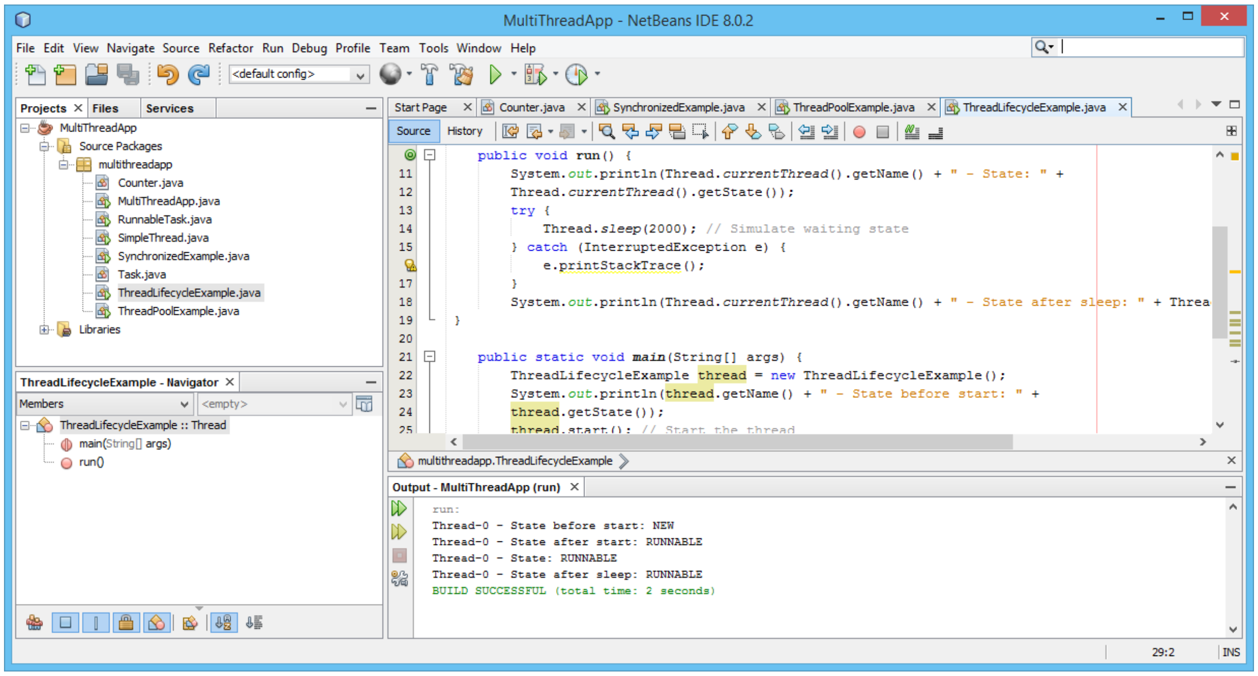
```java
        // Create a thread pool with 3 threads
        ExecutorService executorService = Executors.newFixedThreadPool(3);
        // Submit tasks to the pool
        for (int i = 1; i <= 5; i++) {
            executorService.submit(new Task(i));
        }
        // Shutdown the thread pool
        executorService.shutdown();
    }
}

```

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

Task 05 - Thread Lifecycle Example

ThreadLifecycleExample.java

```
package multithreadapp;
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());
    }
}
```



```
    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: " + Threa
    }

    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
```

```
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)
```