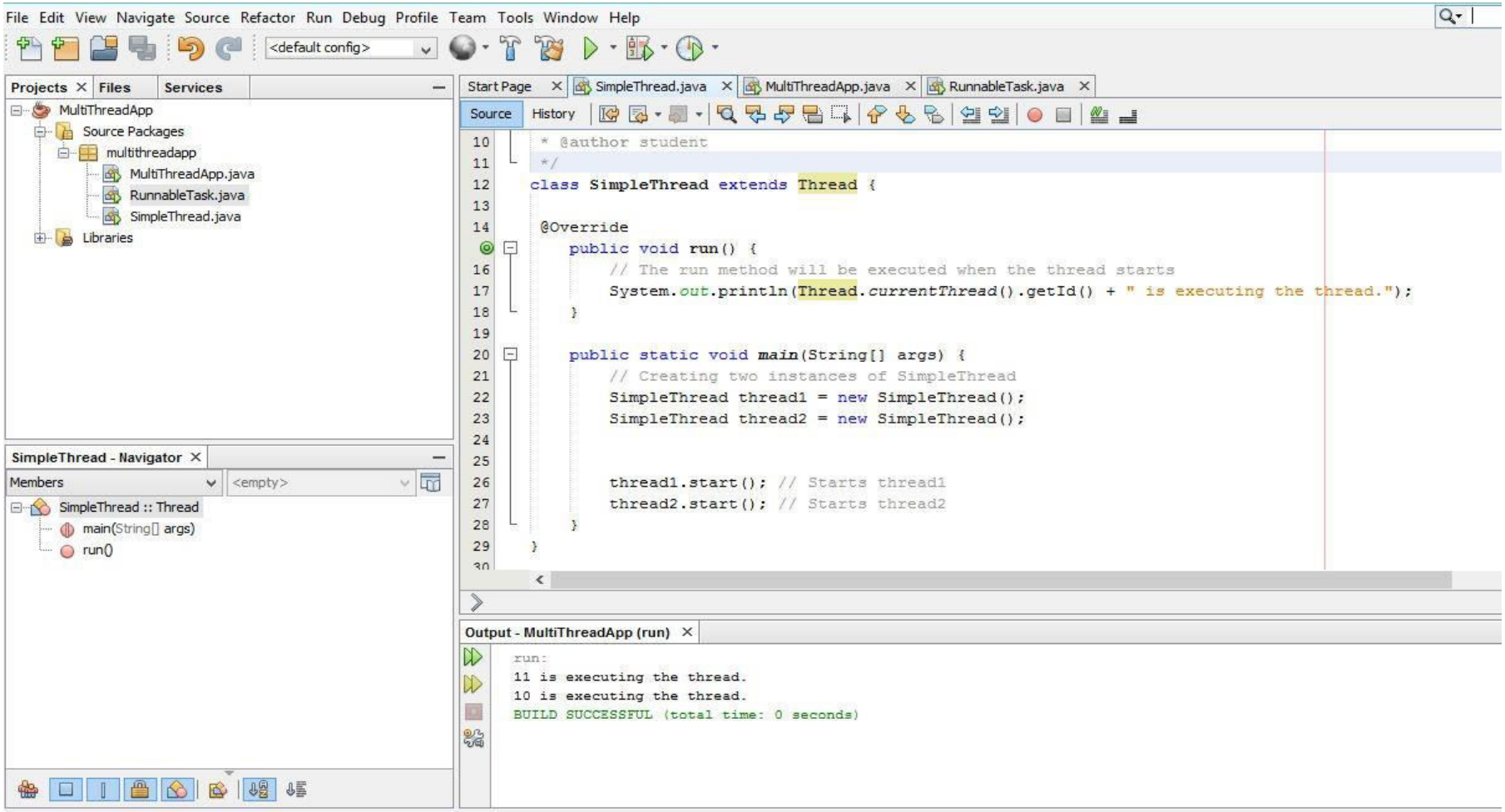
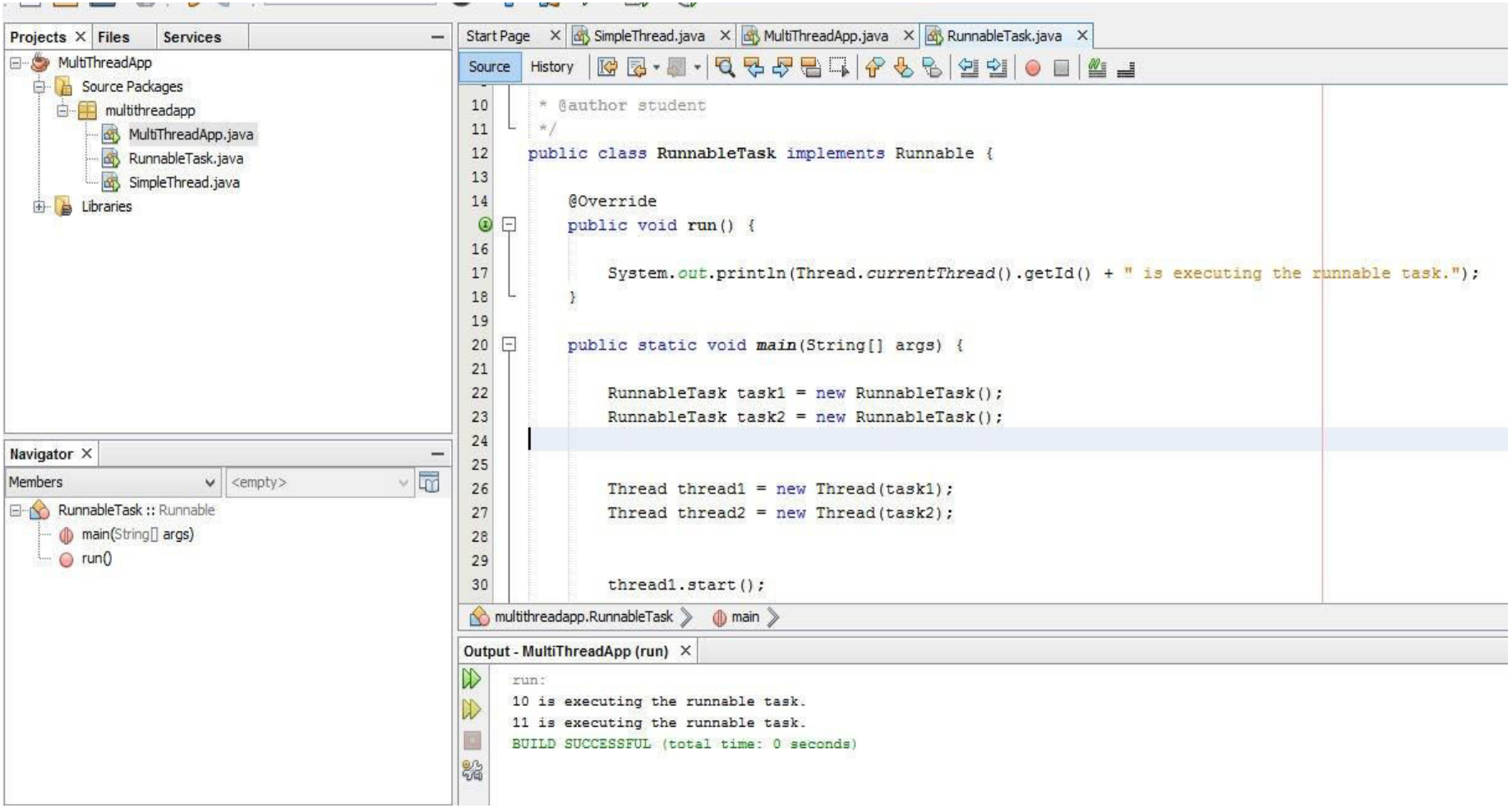
1. Create a Simple Thread Class

```
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
} }
```


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects
Files
Services
MultiThreadApp
Source Packages
multithreadapp
MultiThreadApp.java
RunnableTask.java
SimpleThread.java
Libraries
SimpleThread - Navigator
Members
<empty>
SimpleThread :: Thread
main(String[] args)
run()
Start Page
SimpleThread.java
MultiThreadApp.java
RunnableTask.java
Source
History
10 * @author student
11 */
12 class SimpleThread extends Thread {
13
14 @Override
public void run() {
16 // The run method will be executed when the thread starts
17 System.out.println(Thread.currentThread().getId() + " is executing the thread.");
18 }
19
20 public static void main(String[] args) {
21 // Creating two instances of SimpleThread
22 SimpleThread thread1 = new SimpleThread();
23 SimpleThread thread2 = new SimpleThread();
24
25
26 thread1.start(); // Starts thread1
27 thread2.start(); // Starts thread2
28 }
29 }
Output - MultiThreadApp (run)
run:
11 is executing the thread.
10 is executing the thread.
BUILD SUCCESSFUL (total time: 0 seconds)

2. 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
} }
```


Projects
Files
Services
MultiThreadApp
Source Packages
multithreadapp
MultiThreadApp.java
RunnableTask.java
SimpleThread.java
Libraries
Navigator
Members
<empty>
RunnableTask :: Runnable
main(String[] args)
run()
Start Page
SimpleThread.java
MultiThreadApp.java
RunnableTask.java
Source
History
10 * @author student
11 */
12 public class RunnableTask implements Runnable {
13
14 @Override
15 public void run() {
16
17 System.out.println(Thread.currentThread().getId() + " is executing the runnable task.");
18 }
19
20 public static void main(String[] args) {
21
22 RunnableTask task1 = new RunnableTask();
23 RunnableTask task2 = new RunnableTask();
24
25
26 Thread thread1 = new Thread(task1);
27 Thread thread2 = new Thread(task2);
28
29
30 thread1.start();
multithreadapp.RunnableTask
main
Output - MultiThreadApp (run)
run:
10 is executing the runnable task.
11 is executing the runnable task.
BUILD SUCCESSFUL (total time: 0 seconds)

3. Synchronizing 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());
} }
```

```java
/**
 *
 * @author student
 */
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;
    }
}
```

```java
 */
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();  // Increment counter
        }
    }
```

```java
    public static void main(String[] args) throws InterruptedException {
        // Create a shared Counter object
        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 execution
        thread1.join();
        thread2.join();

        // Output the final value of the counter
        System.out.println("Final counter value: " + counter.getCount());
    }
}
```

synchronizedexample.SynchronizedExample

Output - SynchronizedExample (run)

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

4.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();
} }
```

```java
*/
package threadpoolexample;

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( nThreads: 3);
// Submit tasks to the pool
        for (int i = 1; i <= 5; i++) {
            executorService.submit(new Task( taskId: i));
        }
// Shutdown the thread pool
        executorService.shutdown();
    }
}
```

Output

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

5. Thread Lifecycle Example

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

```java
package threadlifecycleexample;

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 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

```
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)
```