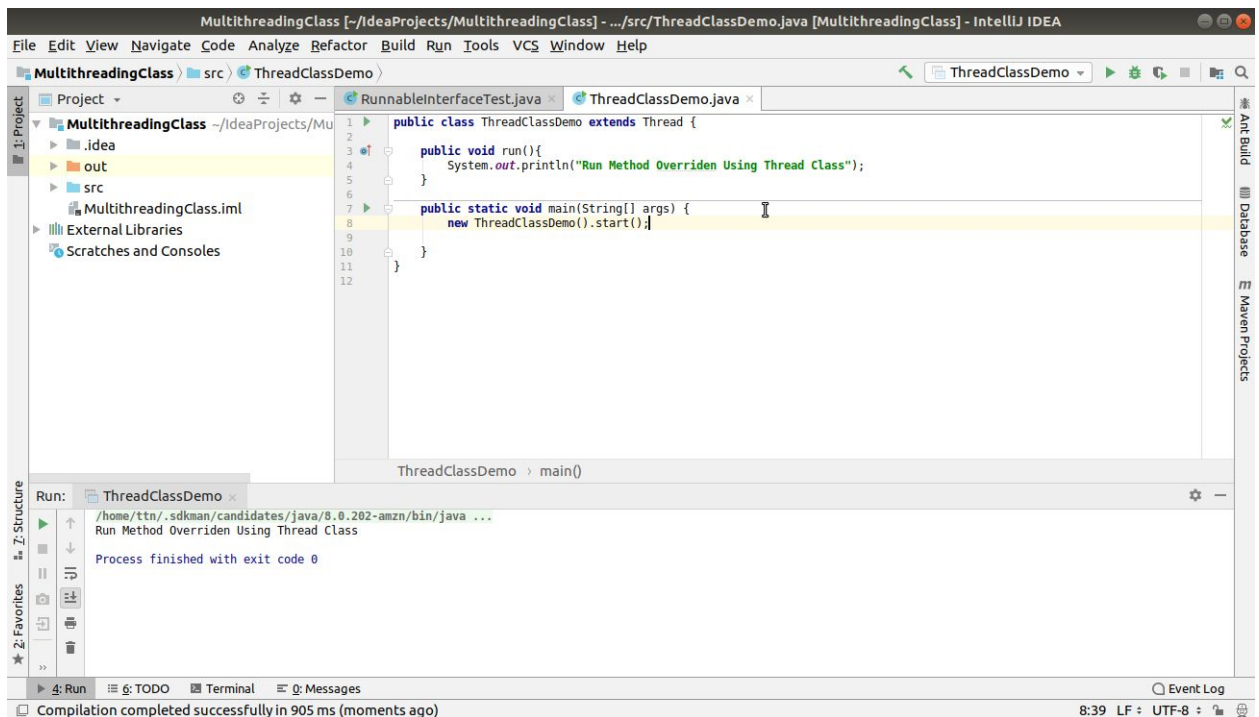
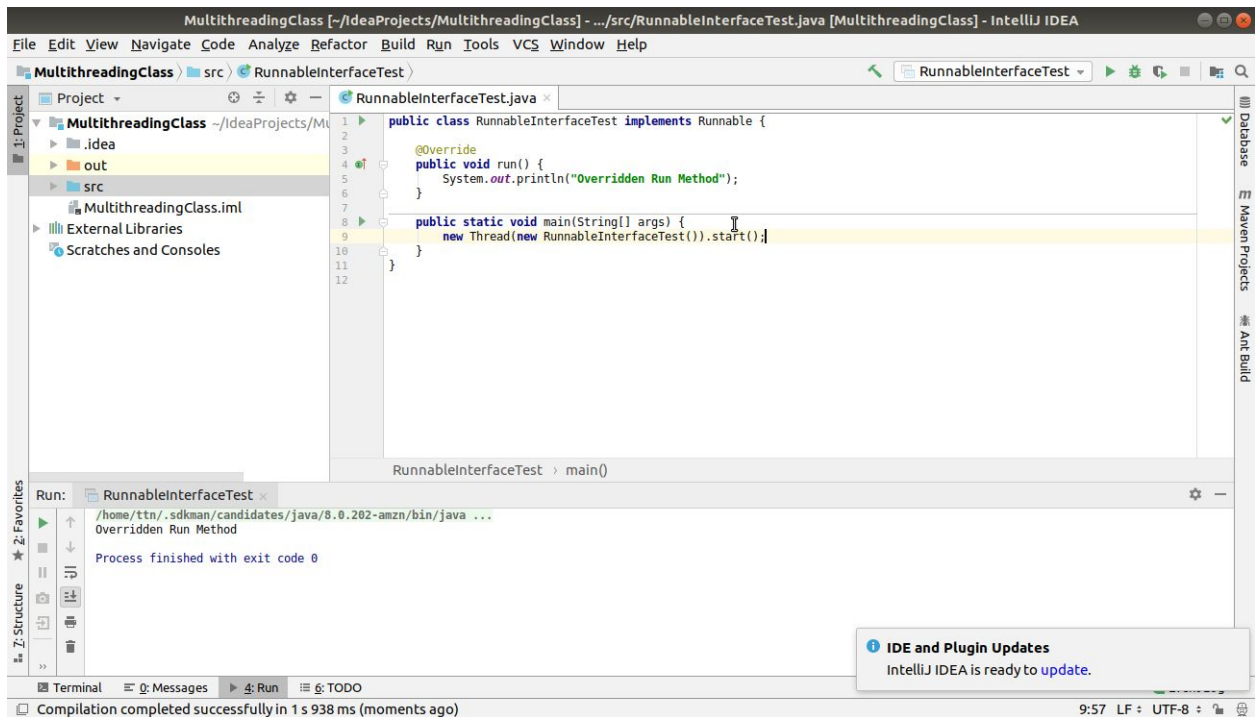
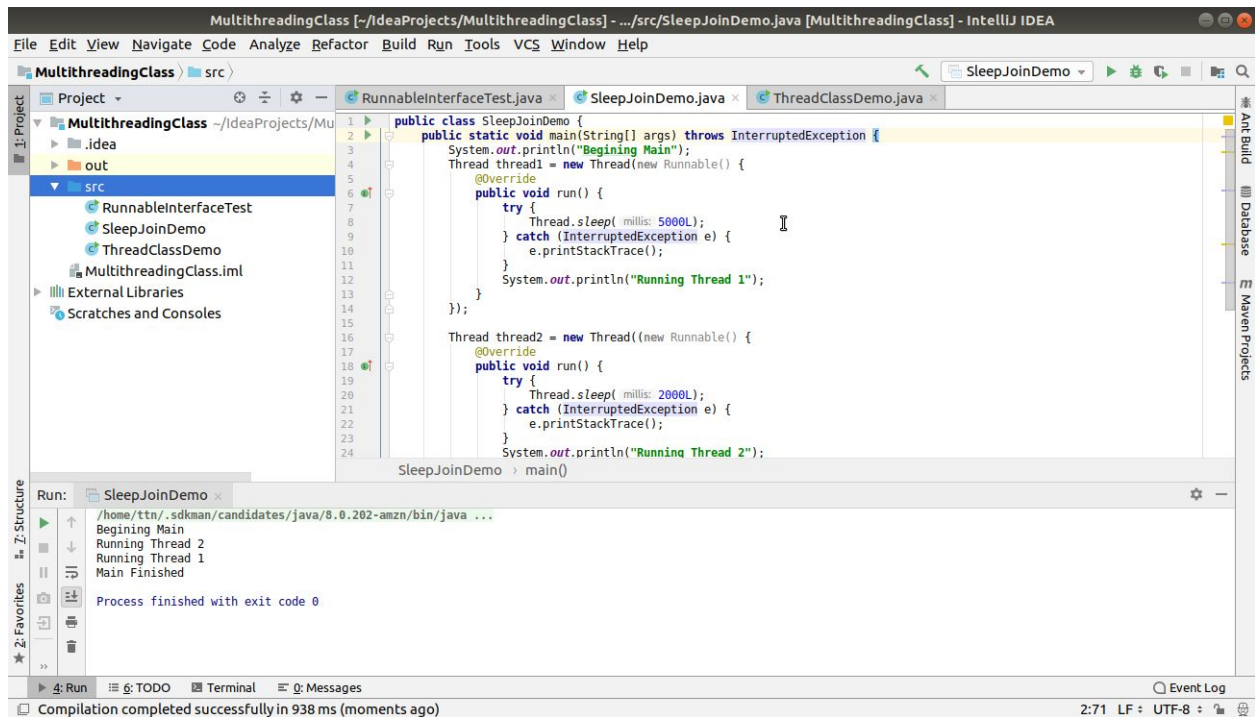


Multithreading Assignment

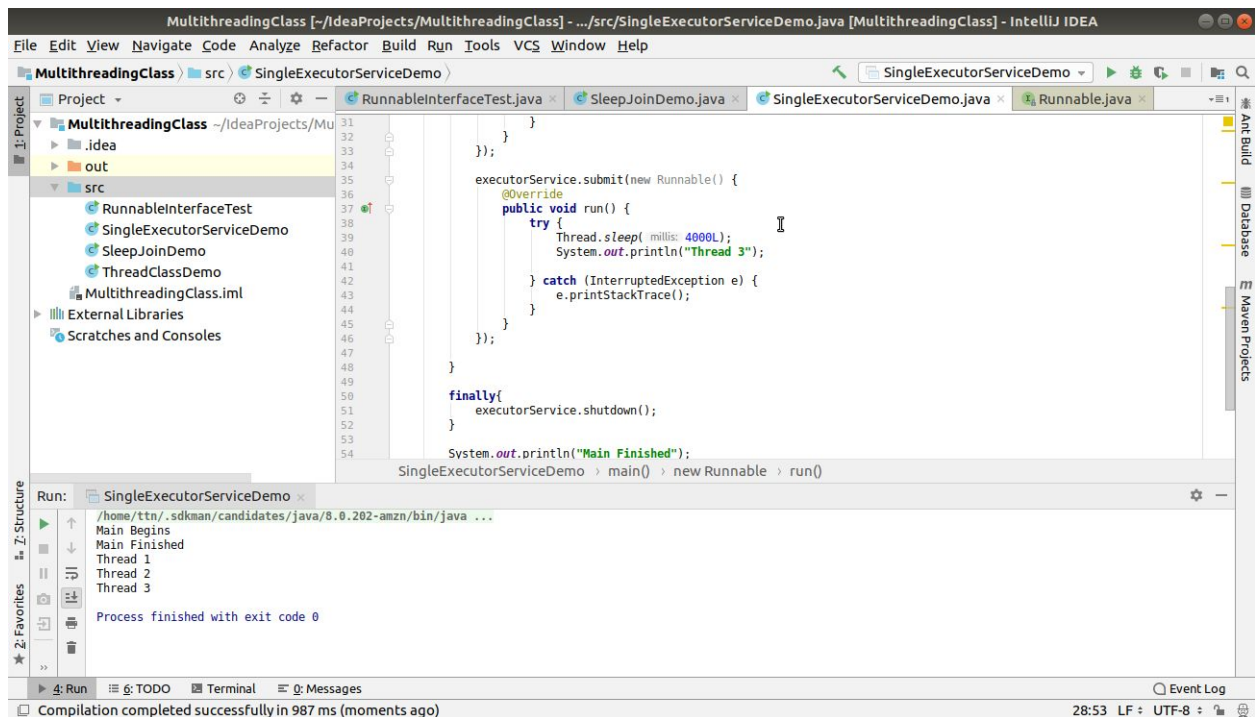
1. Create and Run a Thread using Runnable Interface and Thread class.



2. Use sleep and join methods with thread.

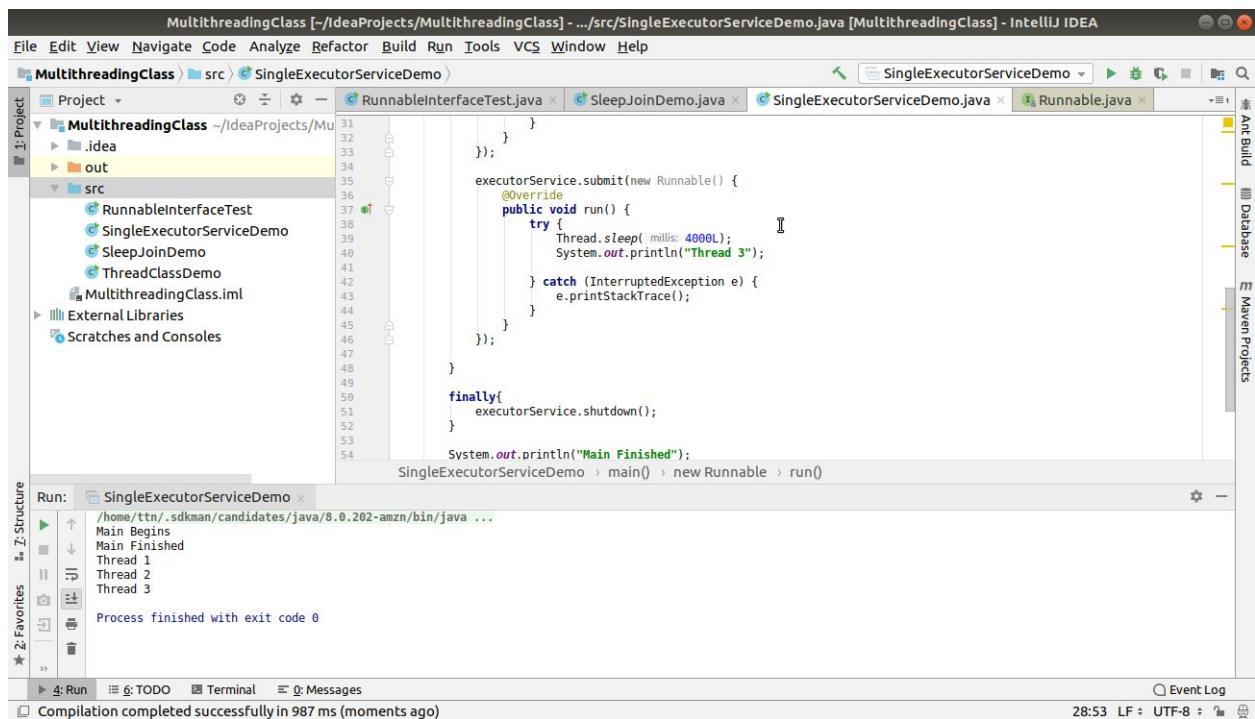


3. Use a singleThreadExecutor to submit multiple threads.

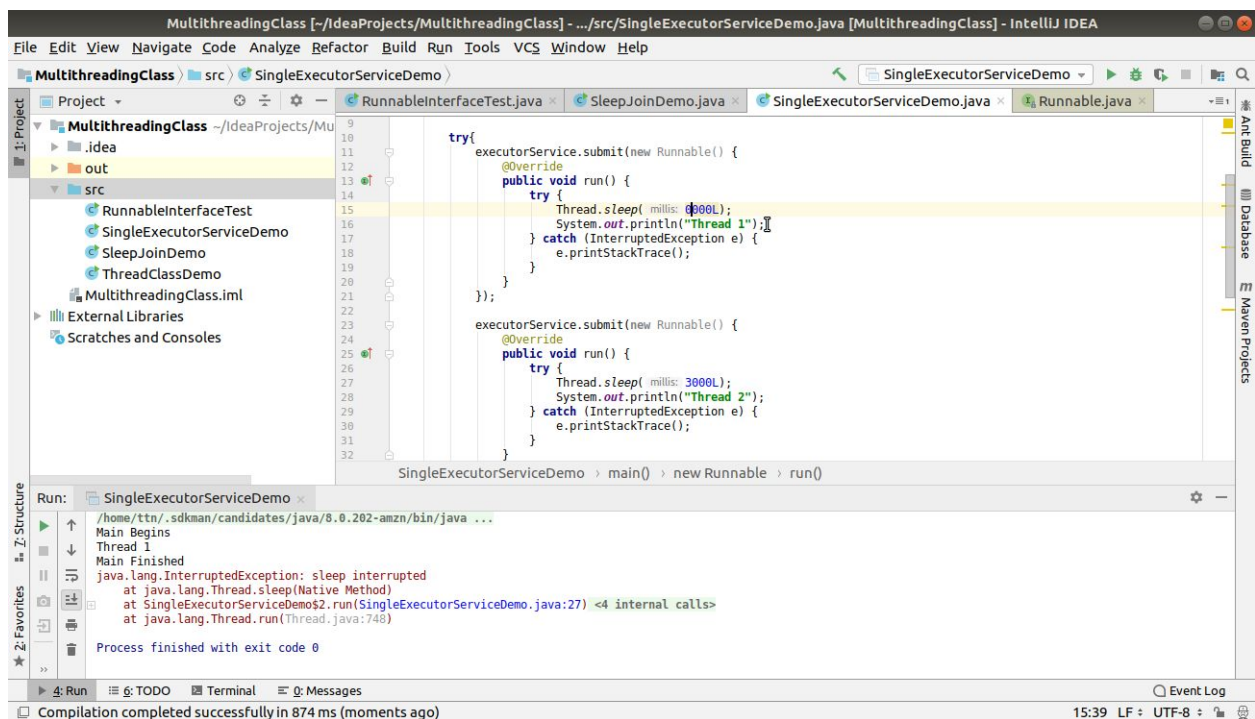


4. Try shutdown() and shutdownNow() and observe the difference.

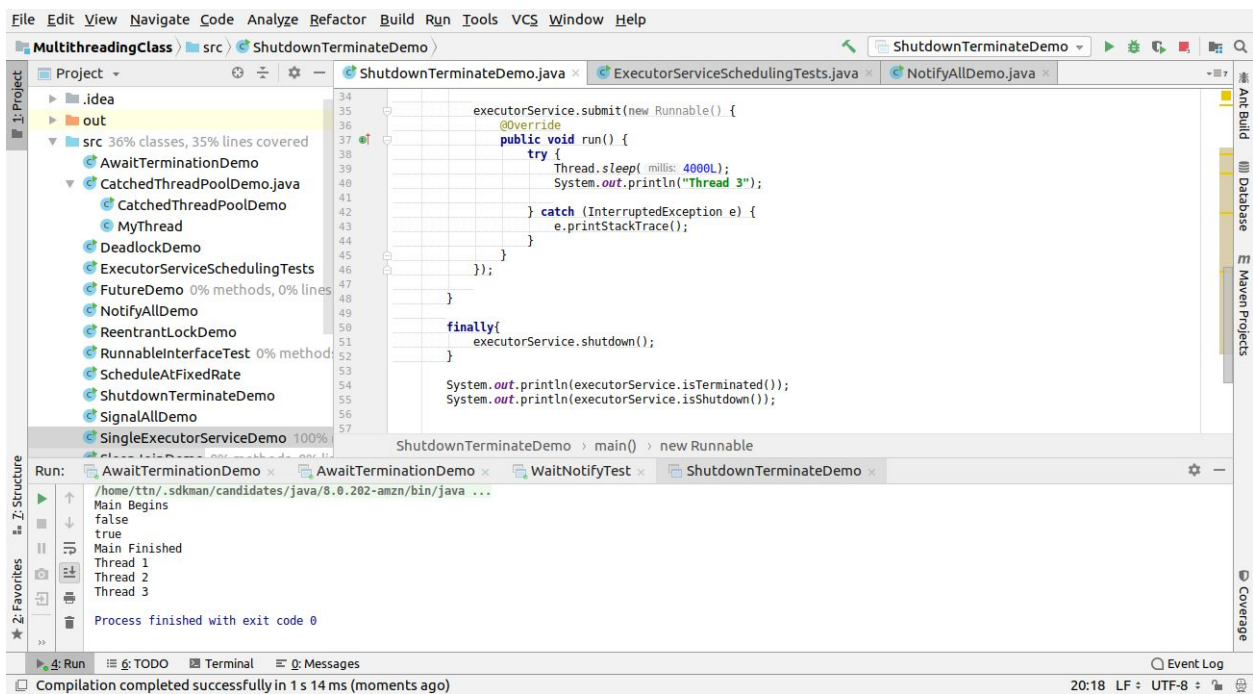
ShutDown



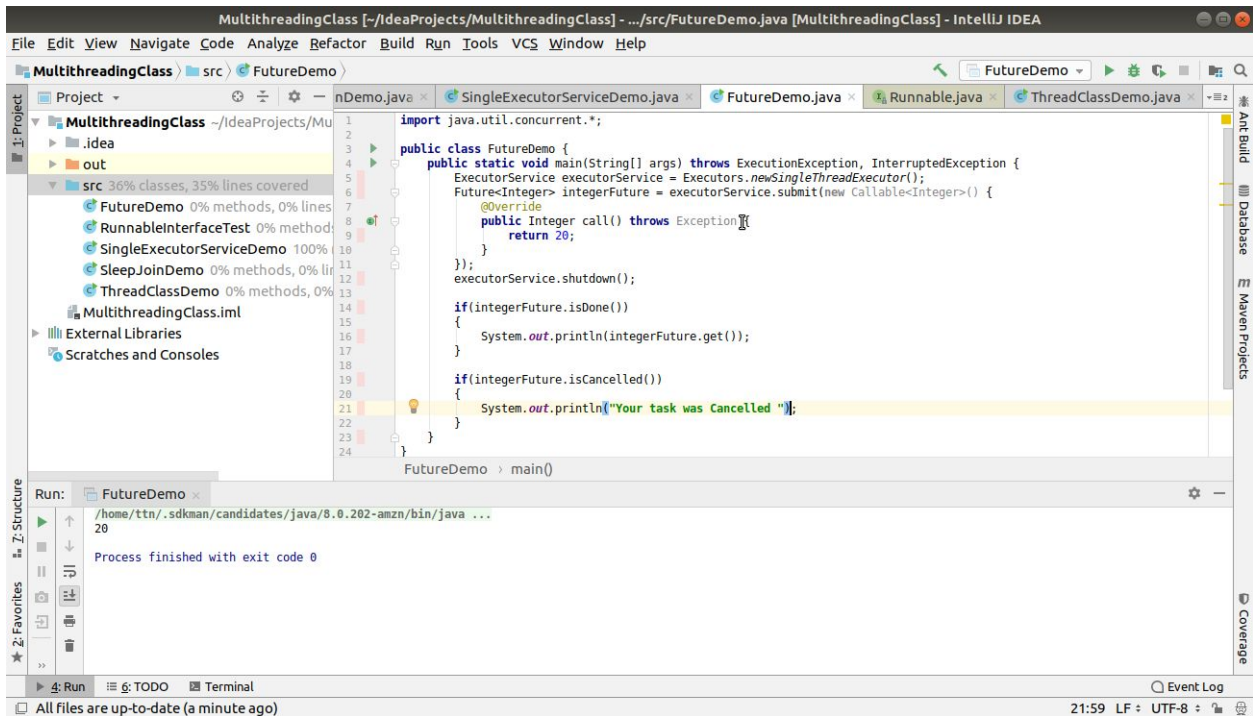
ShutDownNow



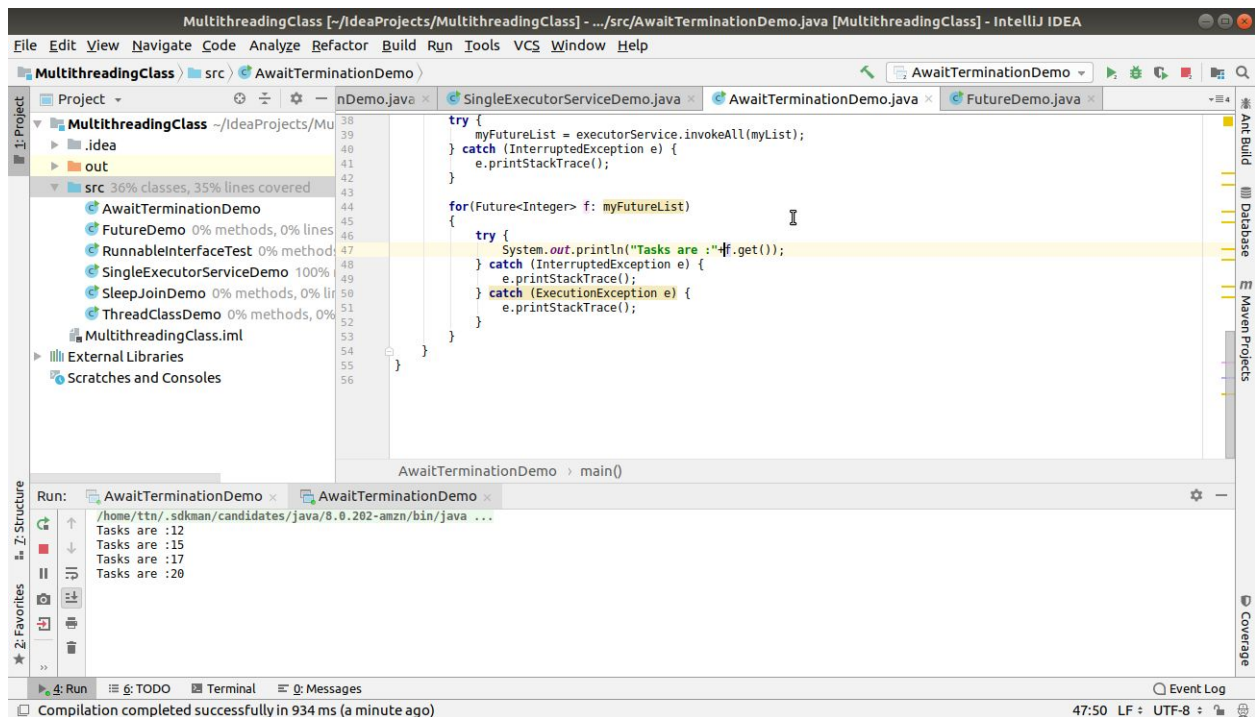
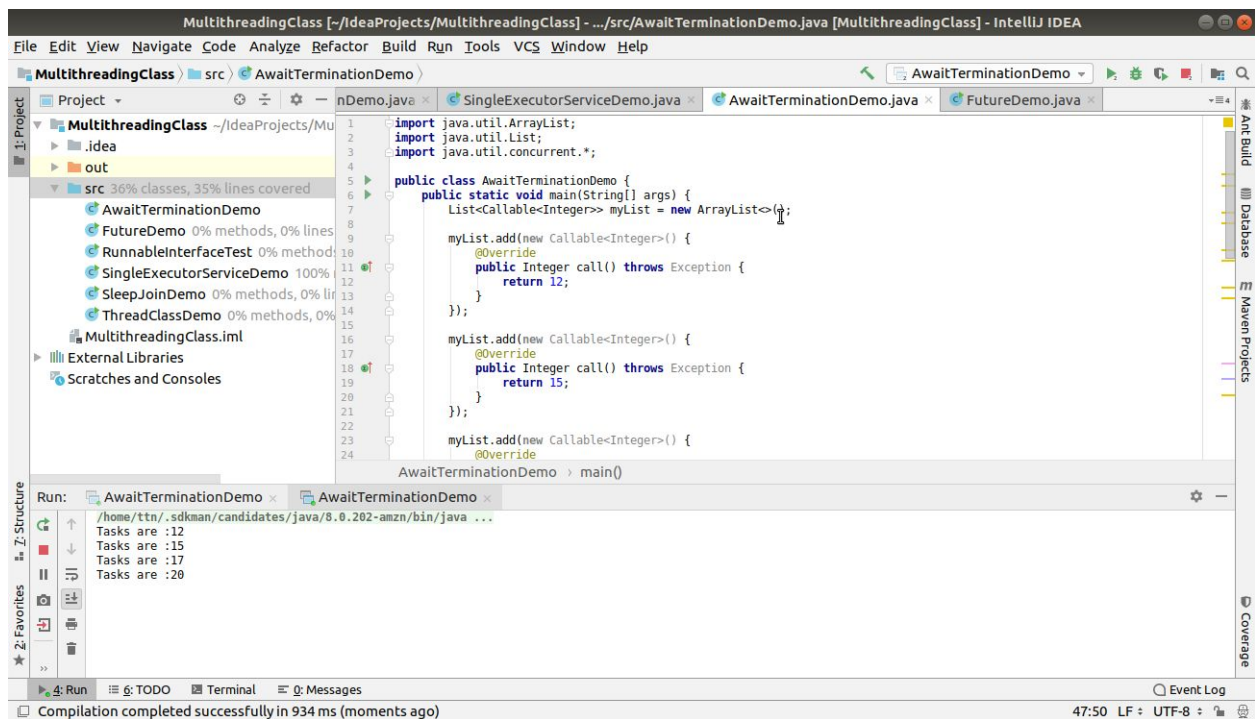
5. Use isShutDown() and isTerminate() with ExecutorService.



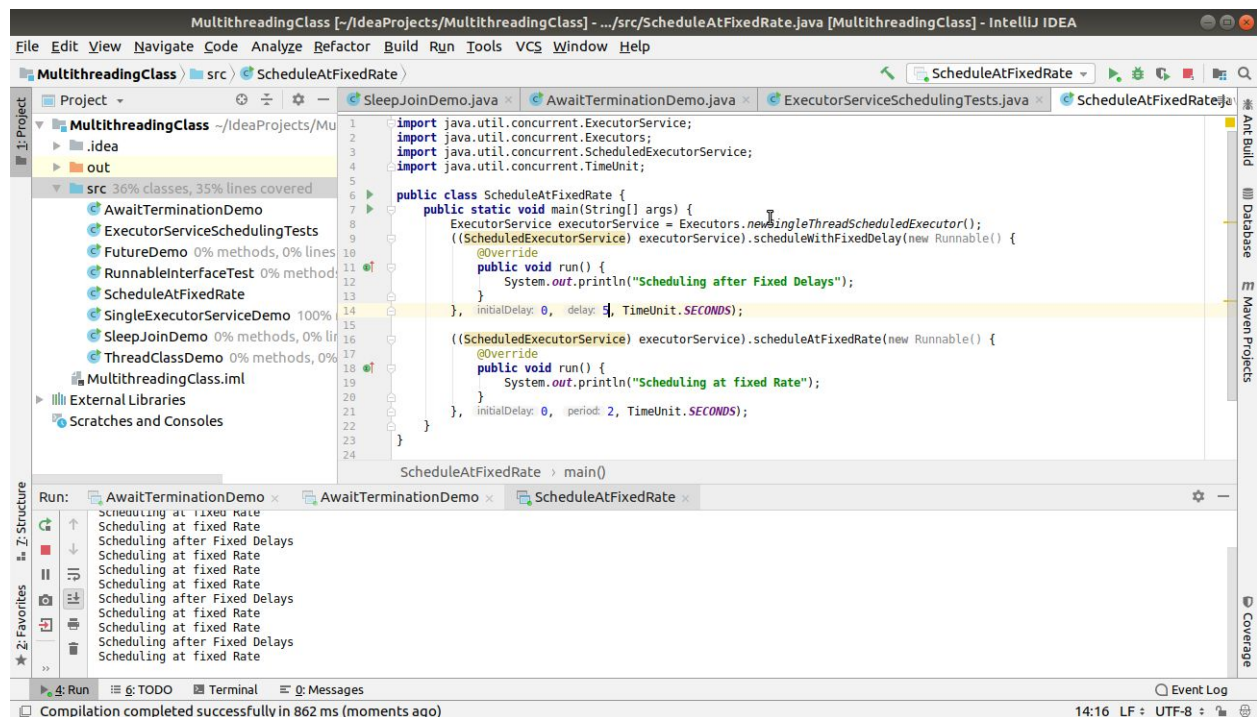
6. Return a Future from ExecutorService by using callable and use get(), isDone(), isCancelled() with the Future object to know the status of task submitted.



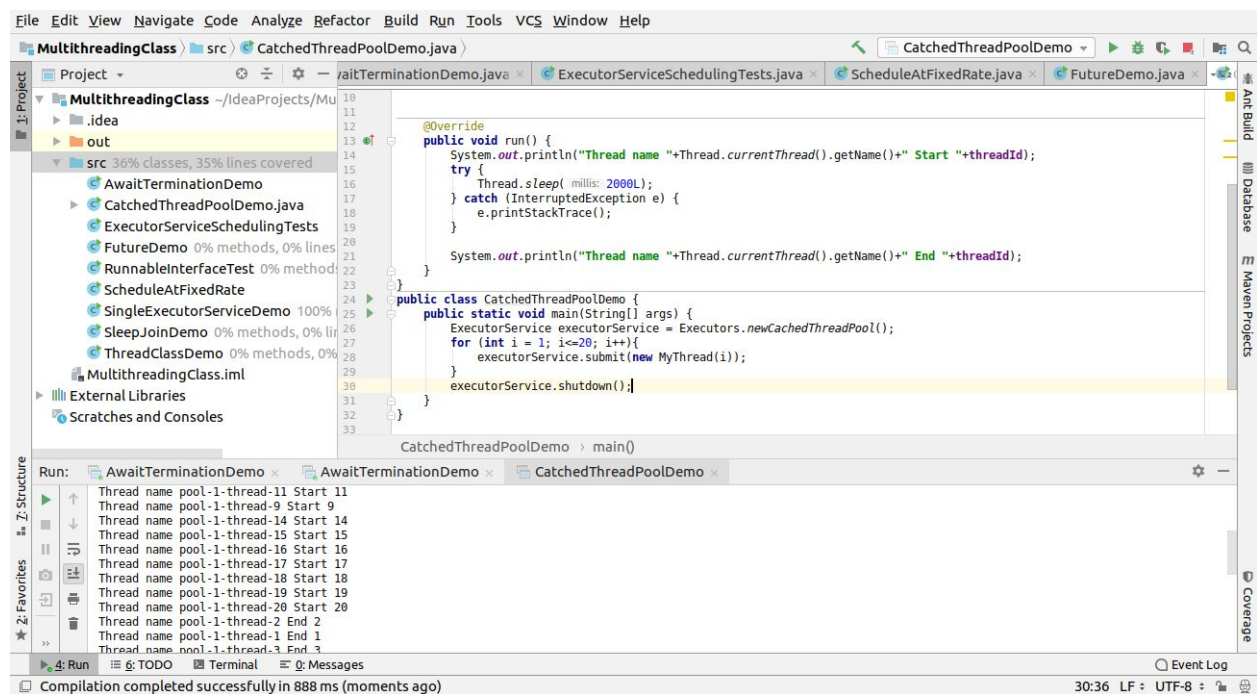
7. Submit List of tasks to ExecutorService and wait for the completion of all the tasks.

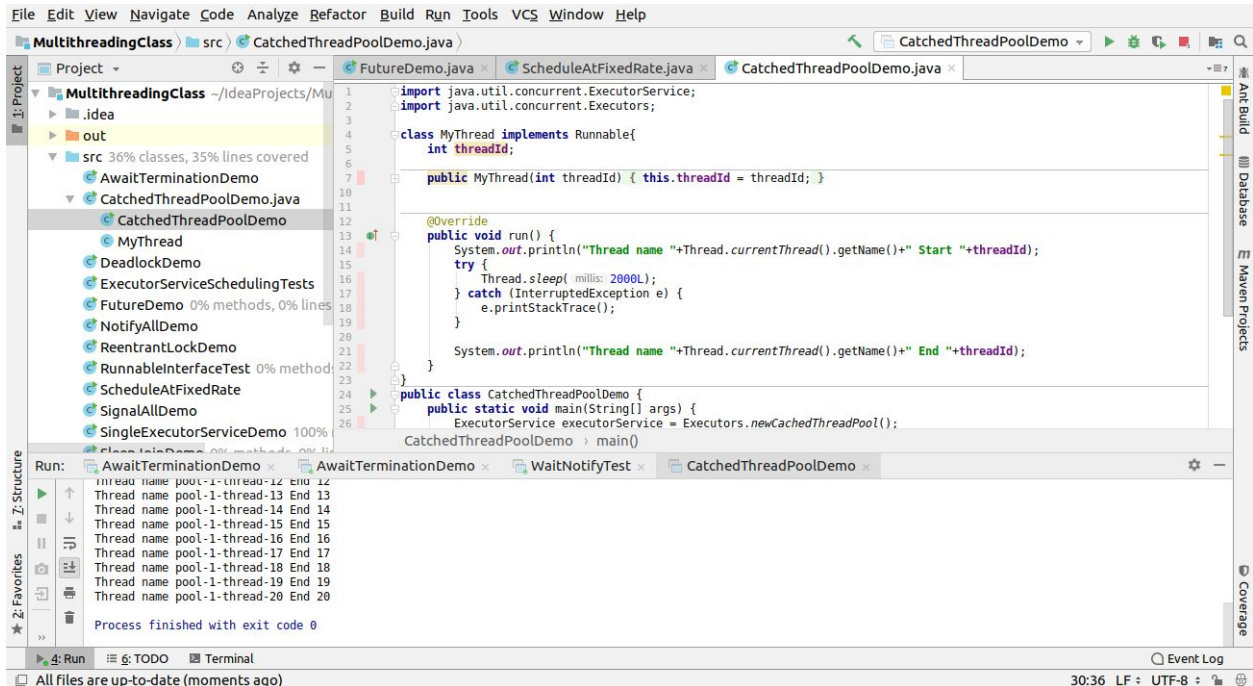


8. Schedule task using schedule(), scheduleAtFixedRate() and scheduleAtFixedDelay()

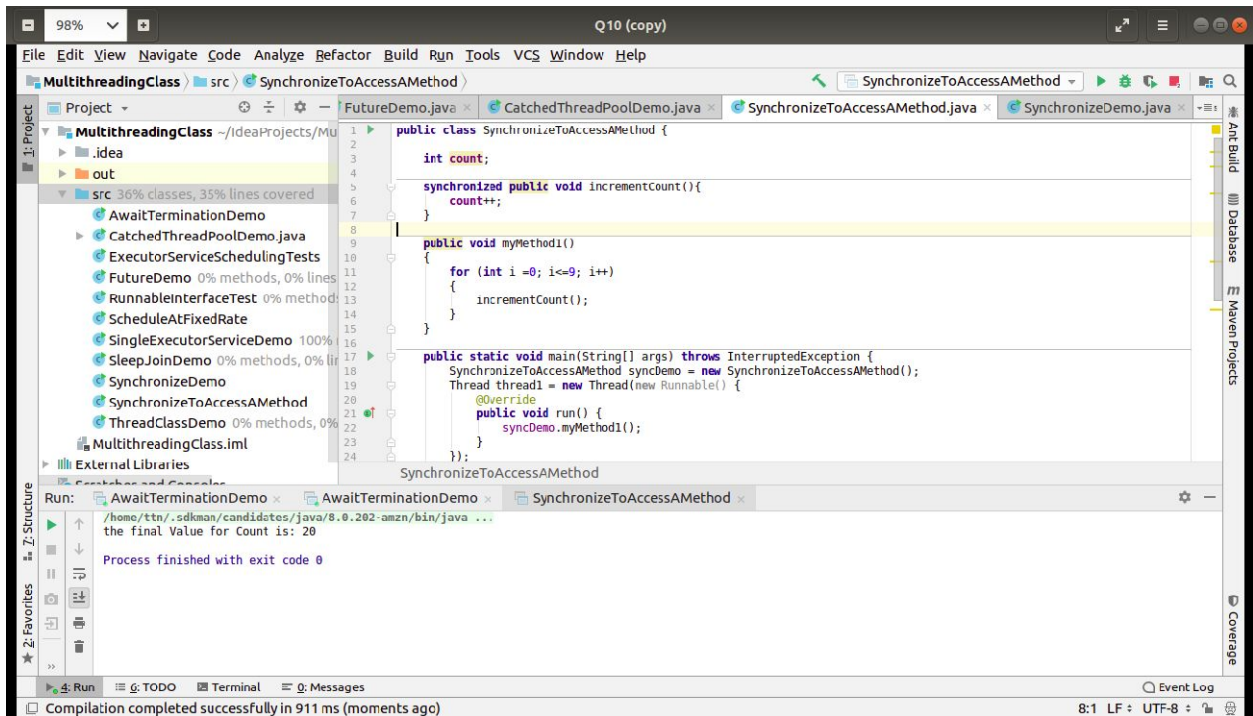


8. Increase concurrency with Thread pools using `newCachedThreadPool()` and `newFixedThreadPool()`.

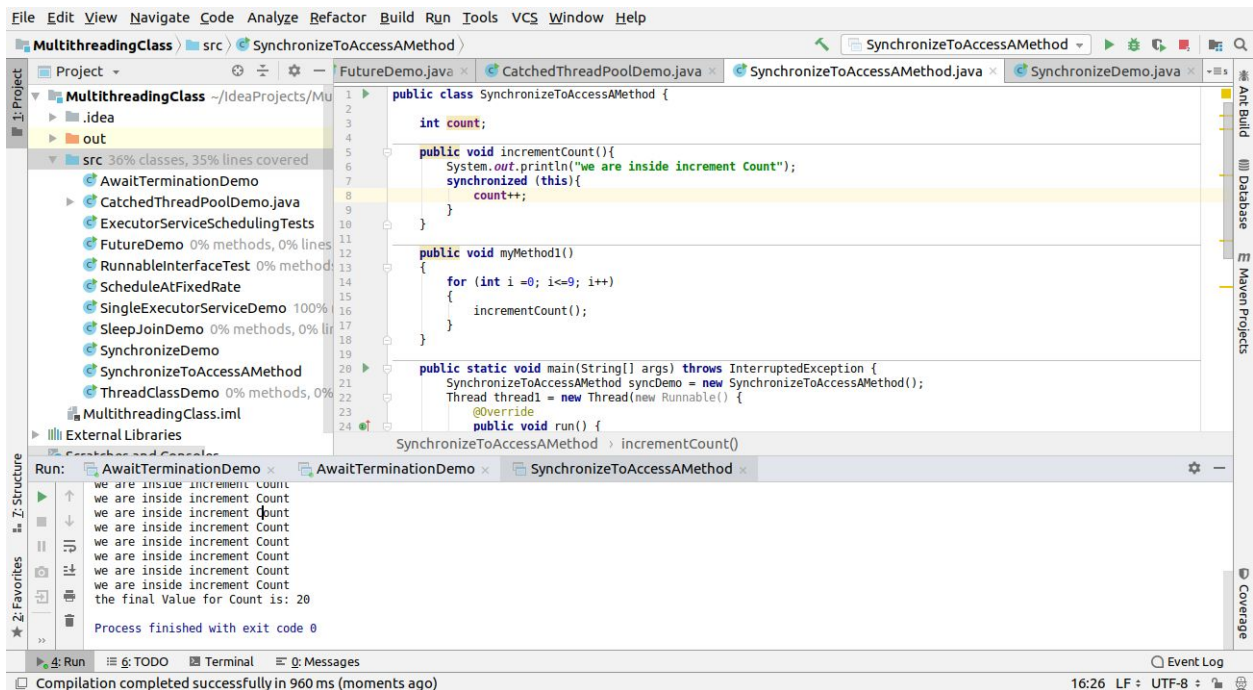




9. Use Synchronize method to enable synchronization between multiple threads trying to access method at same time.



11. Use Synchronize block to enable synchronization between multiple threads trying to access method at same time.



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MultithreadingClass src SynchronizeToAccessAMethod SynchronizeToAccessAMethod.java SynchronizeDemo.java
Project MultithreadingClass ~/IdeaProjects/Mu
  .idea
  out
  src 36% classes, 35% lines covered
    AwaitTerminationDemo
    CaughtThreadPoolDemo.java
    ExecutorServiceSchedulingTests
    FutureDemo 0% methods, 0% lines
    RunnableInterfaceTest 0% methods
    ScheduleAtFixedRate
    SingleExecutorServiceDemo 100%
    SleepJoinDemo 0% methods, 0% lines
    SynchronizeDemo
    SynchronizeToAccessAMethod
    ThreadClassDemo 0% methods, 0%
  MultithreadingClass.iml
  External Libraries
Run: AwaitTerminationDemo x AwaitTerminationDemo x SynchronizeToAccessAMethod x
  we are inside increment Count
  we are inside increment Count
  we are inside increment Count
  we are inside increment Count
  we are inside increment Count
  we are inside increment Count
  we are inside increment Count
  we are inside increment Count
  the final Value for Count is: 20
  Process finished with exit code 0
  Run TODO Terminal Messages Event Log
  Compilation completed successfully in 960 ms (moments ago) 16:26 LF : UTF-8
```

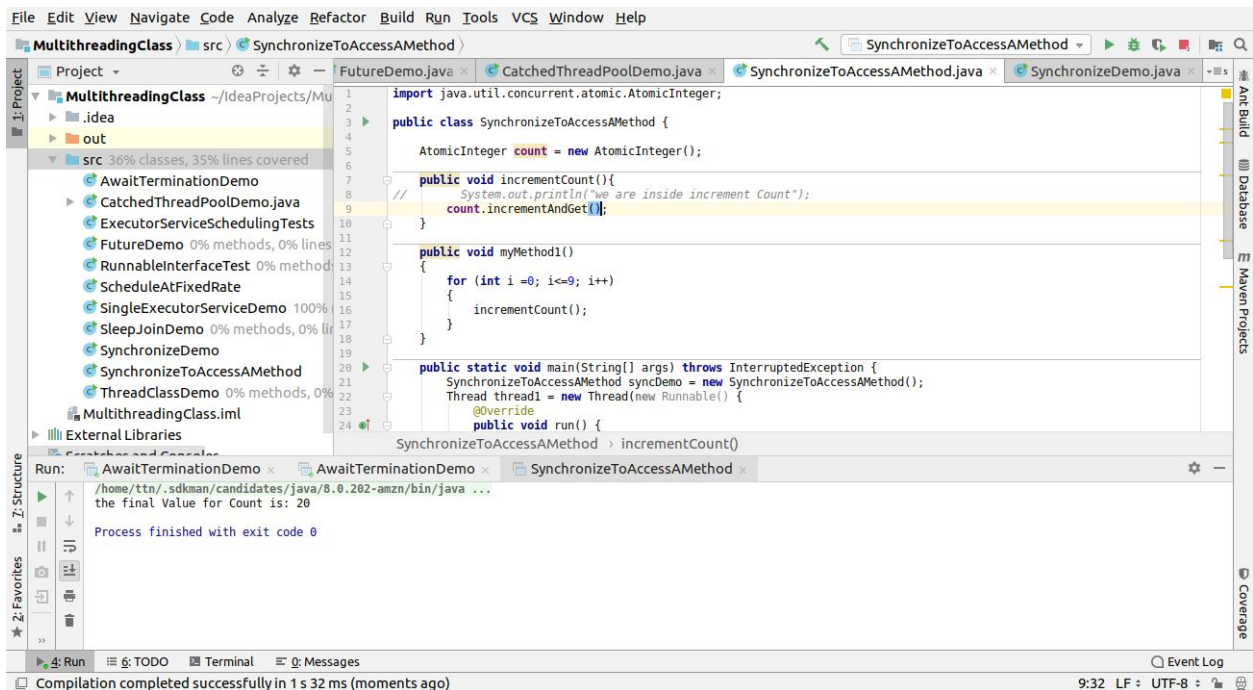
```
public class SynchronizeToAccessAMethod {
    int count;

    public void incrementCount(){
        System.out.println("we are inside increment Count");
        synchronized (this){
            count++;
        }
    }

    public void myMethod1()
    {
        for (int i =0; i<=9; i++)
        {
            incrementCount();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        SynchronizeToAccessAMethod syncDemo = new SynchronizeToAccessAMethod();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                SynchronizeToAccessAMethod incrementCount()
            }
        })
    }
}
```

12. Use Atomic Classes instead of Synchronize method and blocks.



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MultithreadingClass src SynchronizeToAccessAMethod SynchronizeToAccessAMethod.java SynchronizeDemo.java
Project MultithreadingClass ~/IdeaProjects/Mu
  .idea
  out
  src 36% classes, 35% lines covered
    AwaitTerminationDemo
    CaughtThreadPoolDemo.java
    ExecutorServiceSchedulingTests
    FutureDemo 0% methods, 0% lines
    RunnableInterfaceTest 0% methods
    ScheduleAtFixedRate
    SingleExecutorServiceDemo 100%
    SleepJoinDemo 0% methods, 0% lines
    SynchronizeDemo
    SynchronizeToAccessAMethod
    ThreadClassDemo 0% methods, 0%
  MultithreadingClass.iml
  External Libraries
Run: AwaitTerminationDemo x AwaitTerminationDemo x SynchronizeToAccessAMethod x
  /home/ttn/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
  the final Value for Count is: 20
  Process finished with exit code 0
  Run TODO Terminal Messages Event Log
  Compilation completed successfully in 1 s 32 ms (moments ago) 9:32 LF : UTF-8
```

```
import java.util.concurrent.atomic.AtomicInteger;

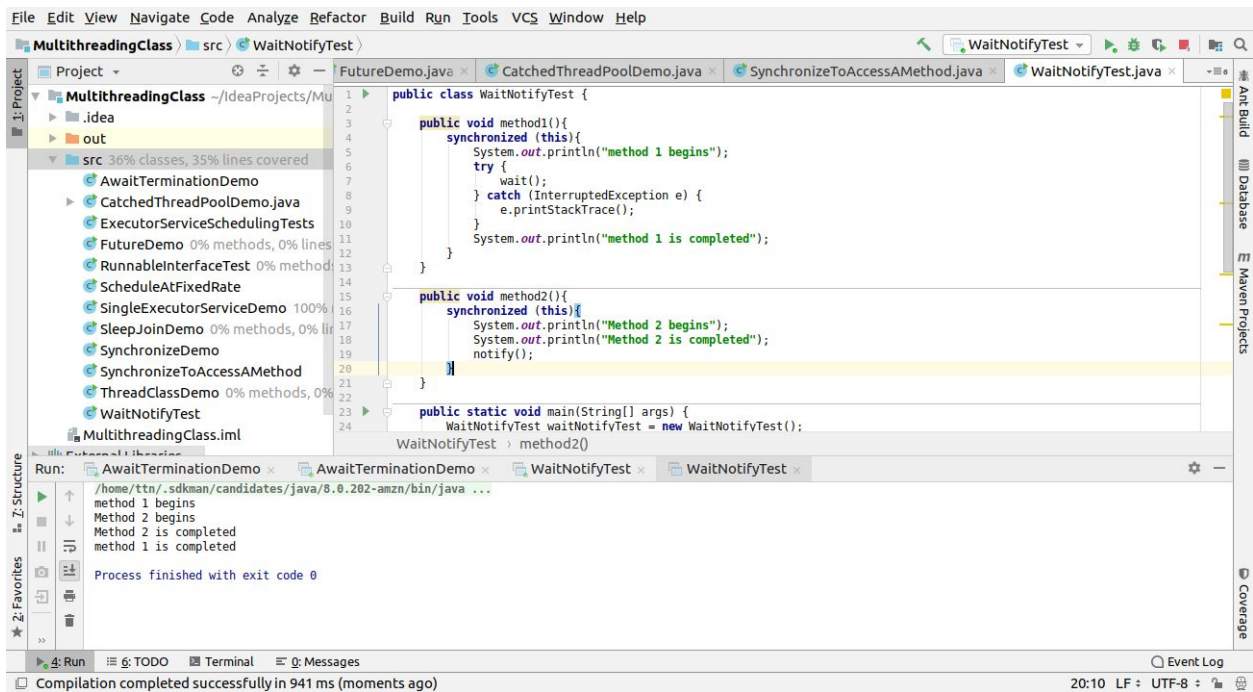
public class SynchronizeToAccessAMethod {
    AtomicInteger count = new AtomicInteger();

    public void incrementCount(){
        // System.out.println("we are inside increment Count");
        count.incrementAndGet();
    }

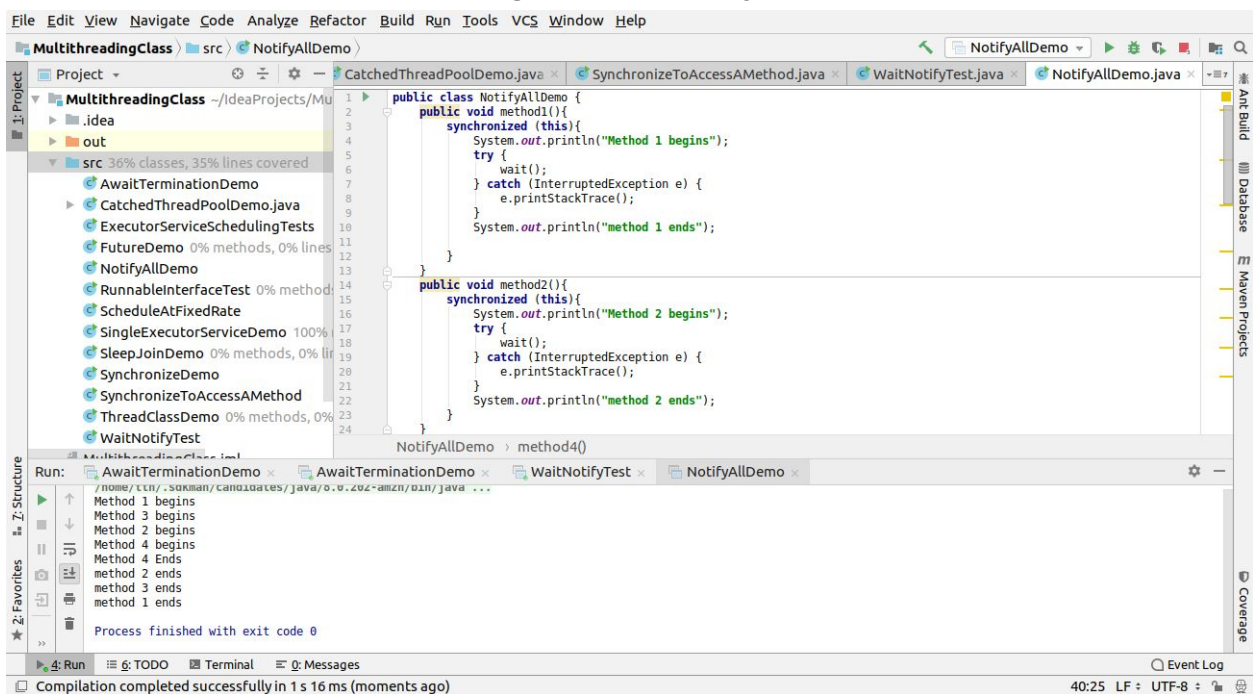
    public void myMethod1()
    {
        for (int i =0; i<=9; i++)
        {
            incrementCount();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        SynchronizeToAccessAMethod syncDemo = new SynchronizeToAccessAMethod();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                SynchronizeToAccessAMethod incrementCount()
            }
        })
    }
}
```

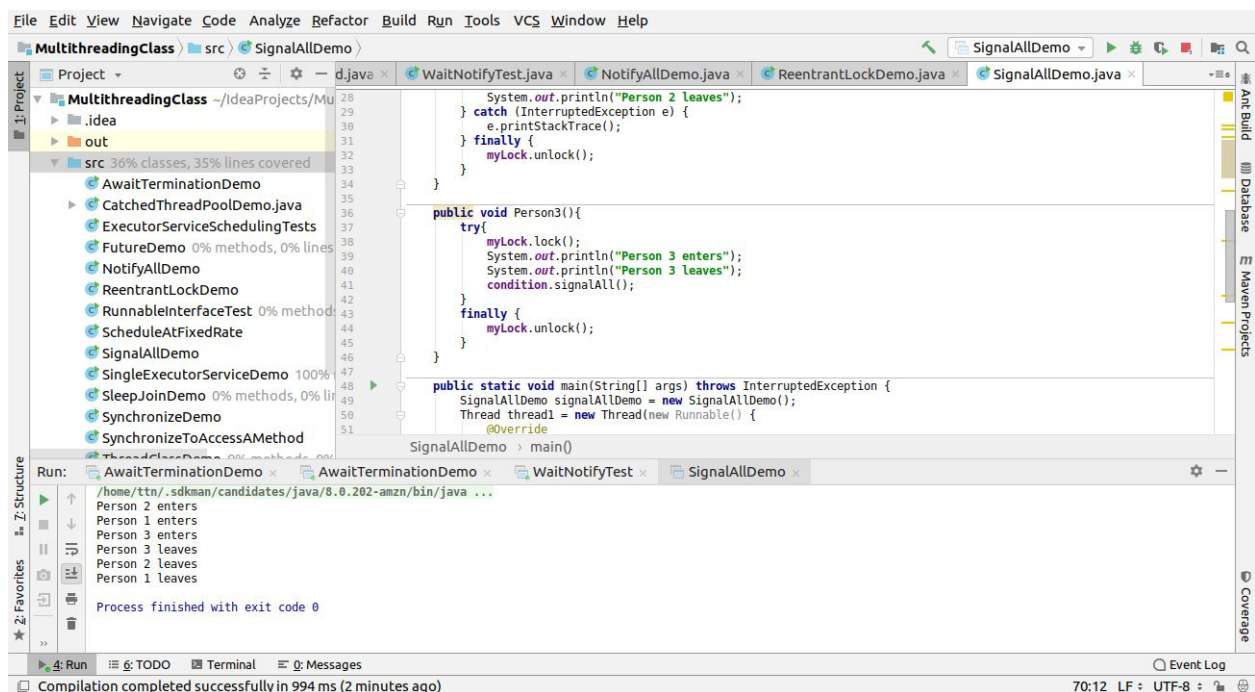
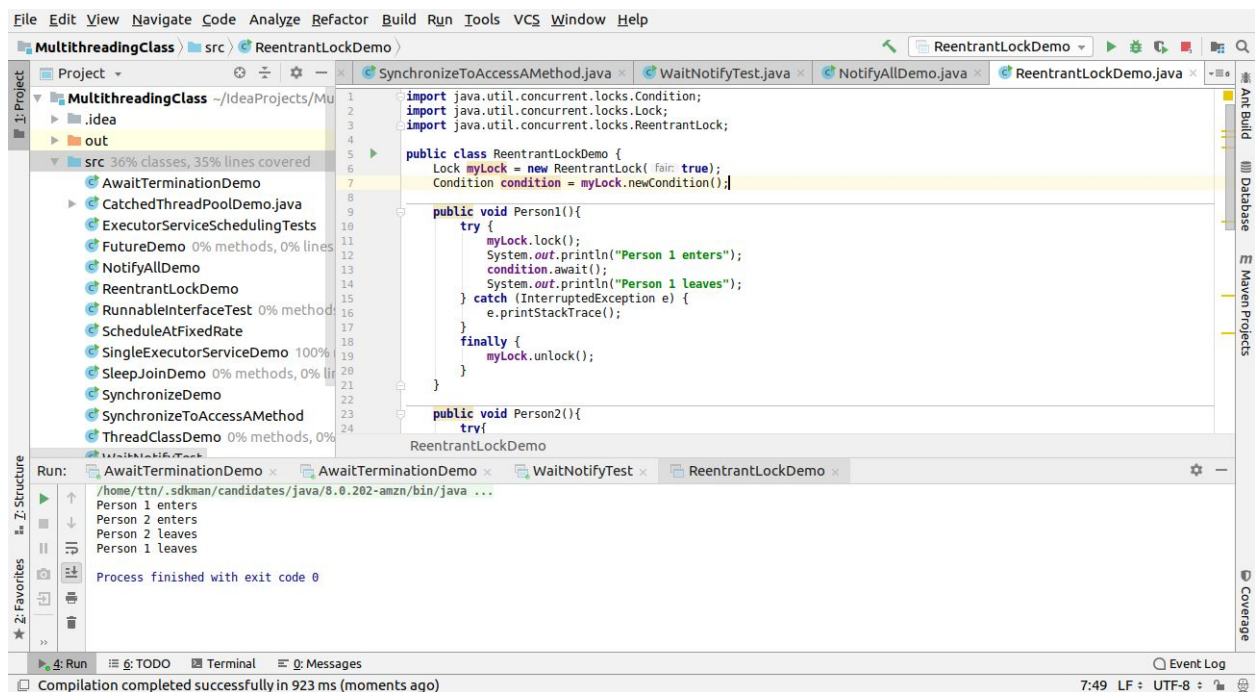

13. Coordinate 2 threads using wait() and notify().



14. Coordinate multiple threads using wait() and notifyAll()



15. Use Reentrant lock for coordinating 2 threads with signal(), signalAll() and wait().



16. Create a deadlock and Resolve it using tryLock().

The screenshot shows the IntelliJ IDEA IDE with the `DeadlockDemo.java` file open. The code defines a `DeadlockDemo` class with two locks, `myLock1` and `myLock2`, and two methods, `Person1()` and `Person2()`. `Person1()` locks `myLock1` and prints "Person 1 Lock 1", then attempts to lock `myLock2`. `Person2()` locks `myLock2` and prints "Person 2 Lock 2", then attempts to lock `myLock1`. This creates a deadlock state where both threads are waiting for the other to finish.

```
1 import java.util.concurrent.locks.Lock;
2 import java.util.concurrent.locks.ReentrantLock;
3
4 public class DeadlockDemo {
5     Lock myLock1 = new ReentrantLock( fair: true);
6     Lock myLock2 = new ReentrantLock( fair: true);
7
8     public void Person1(){
9         myLock1.lock();
10        System.out.println("Person 1 Lock 1");
11        myLock2.lock();
12        System.out.println("Person 1 Lock 2 ");
13        myLock2.unlock();
14        myLock1.unlock();
15    }
16
17    public void Person2(){
18        myLock2.lock();
19        System.out.println("Person 2 Lock 2");
20        myLock1.lock();
21        System.out.println("Person 2 Lock 1");
22        myLock2.unlock();
23        myLock1.unlock();
24    }
25 }
```

The Run window shows the execution of `DeadlockDemo` with the following output:

```
Person 1 Lock 1
Person 1 Lock 2
Person 2 Lock 2
Person 2 Lock 1
Process finished with exit code 0
```

The status bar indicates that the compilation completed successfully in 845 ms (a minute ago).

The screenshot shows the IntelliJ IDEA IDE with the `TryLockDemo.java` file open. The code defines a `TryLockDemo` class with two locks, `myLock1` and `myLock2`, and a method `main()`. `main()` locks `myLock1` and prints "Person 1 Lock 1", then attempts to lock `myLock2` using `tryLock()`. If it fails, it prints "Person 1 Lock 2" and unlocks `myLock1`. `main()` then locks `myLock2` and prints "Person 2 Lock 2", then attempts to lock `myLock1` using `tryLock()`. If it fails, it prints "Person 2 Lock 1" and unlocks `myLock2`. This resolves the deadlock by ensuring that the threads can finish their execution.

```
1 import com.sun.org.apache.xerces.internal.dom.PSVIAttrNSImpl;
2
3 import java.util.concurrent.locks.Lock;
4 import java.util.concurrent.locks.ReentrantLock;
5
6 public class TryLockDemo {
7
8     Lock myLock1 = new ReentrantLock( fair: true);
9     Lock myLock2 = new ReentrantLock( fair: true);
10
11     public void acquireLock(Lock myLock1, Lock myLock2){
12         boolean acquiremyLock1 = myLock1.tryLock();
13         boolean acquiremyLock2 = myLock2.tryLock();
14
15         if(acquiremyLock1 && acquiremyLock2)
16         {
17             return;
18         }
19
20         if(acquiremyLock1)
21         {
22             myLock1.unlock();
23         }
24     }
25 }
```

The Run window shows the execution of `TryLockDemo` with the following output:

```
Person 1 Lock 1
Person 1 Lock 2
Person 2 Lock 2
Person 2 Lock 1
Process finished with exit code 0
```

The status bar indicates that the compilation completed successfully in 989 ms (a minute ago).