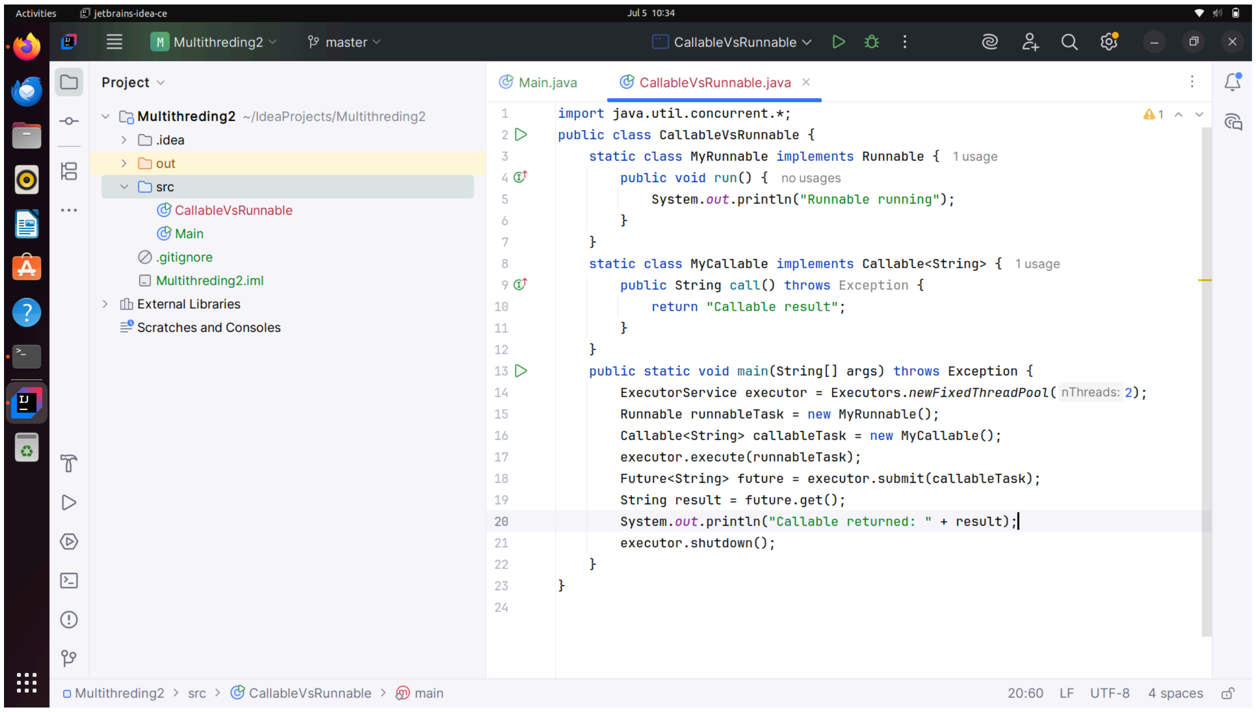
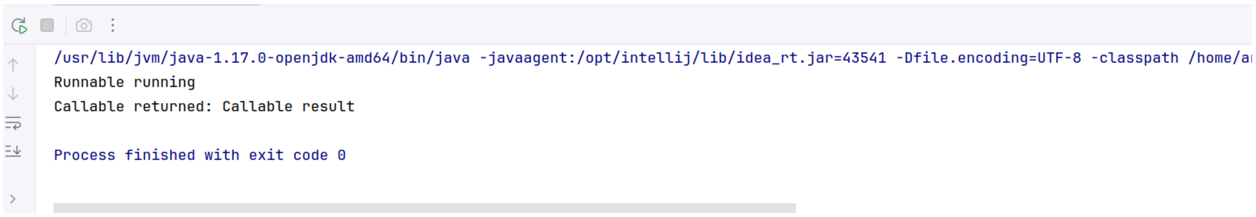
1. WAP to show usage of Callable and demonstrate how it is different from Runnable



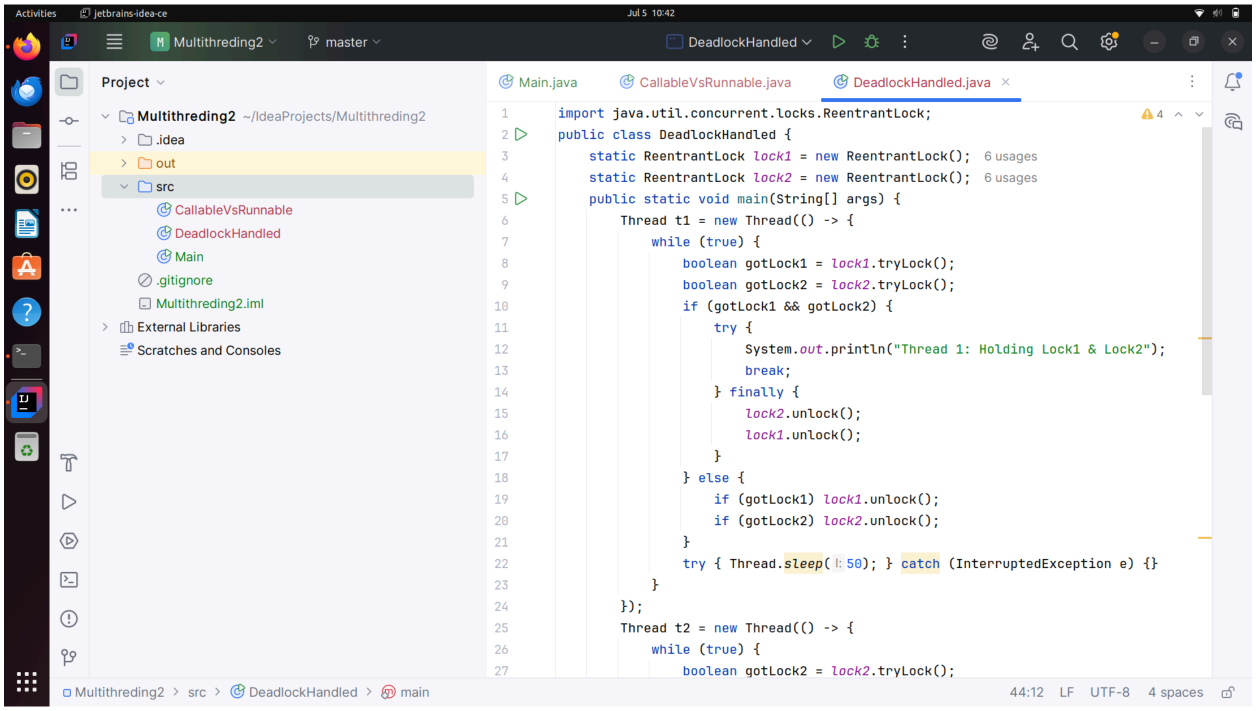
```java
import java.util.concurrent.*;
public class CallableVsRunnable {
    static class MyRunnable implements Runnable {
        public void run() {
            System.out.println("Runnable running");
        }
    }
    static class MyCallable implements Callable<String> {
        public String call() throws Exception {
            return "Callable result";
        }
    }
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 2);
        Runnable runnableTask = new MyRunnable();
        Callable<String> callableTask = new MyCallable();
        executor.execute(runnableTask);
        Future<String> future = executor.submit(callableTask);
        String result = future.get();
        System.out.println("Callable returned: " + result);
        executor.shutdown();
    }
}
```

Output:

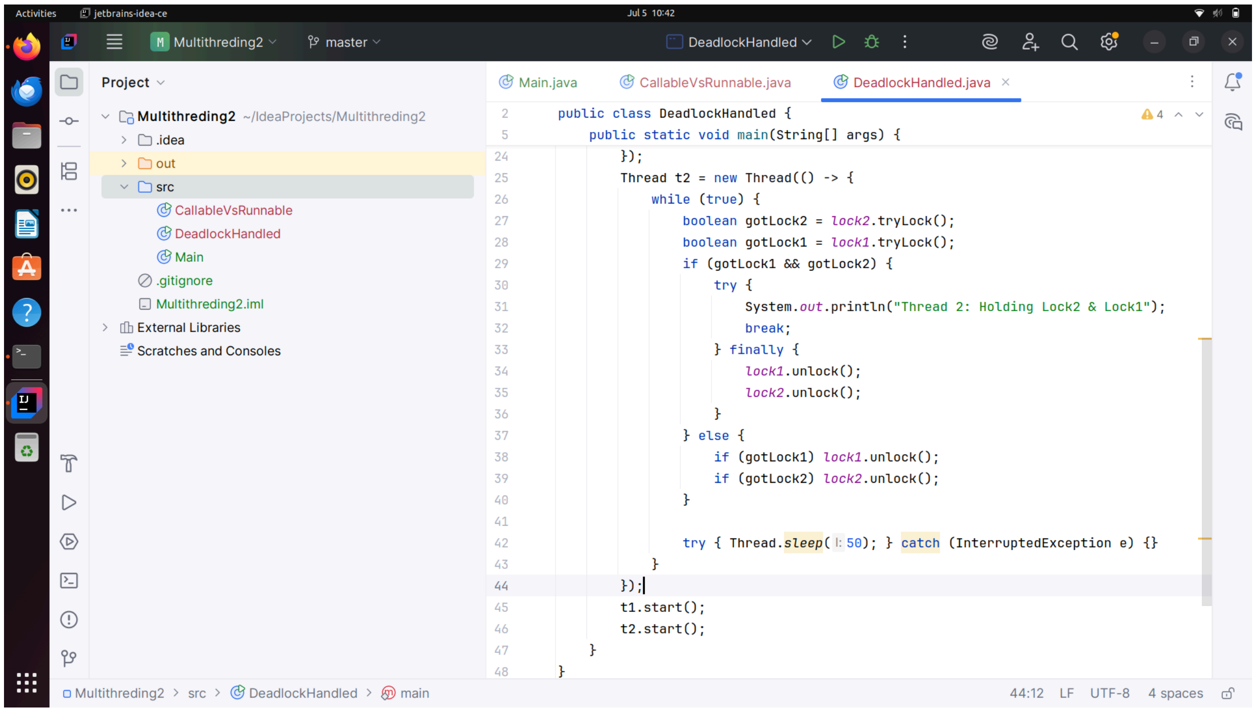


```
/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/java -javaagent:/opt/intellij/lib/idea_rt.jar=43541 -Dfile.encoding=UTF-8 -classpath /home/a
Runnable running
Callable returned: Callable result

Process finished with exit code 0
```

2. Improve the code written in Basics of Multi Threading Part 1 exercise question 4 to handle the deadlock using reentract lock.

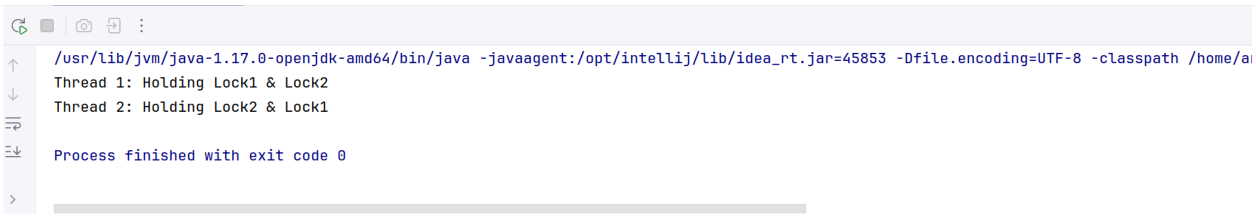


```java
import java.util.concurrent.locks.ReentrantLock;
public class DeadlockHandled {
    static ReentrantLock lock1 = new ReentrantLock();  6 usages
    static ReentrantLock lock2 = new ReentrantLock();  6 usages
    public static void main(String[] args) {
        Thread t1 = new Thread(() -> {
            while (true) {
                boolean gotLock1 = lock1.tryLock();
                boolean gotLock2 = lock2.tryLock();
                if (gotLock1 && gotLock2) {
                    try {
                        System.out.println("Thread 1: Holding Lock1 & Lock2");
                        break;
                    } finally {
                        lock2.unlock();
                        lock1.unlock();
                    }
                } else {
                    if (gotLock1) lock1.unlock();
                    if (gotLock2) lock2.unlock();
                }
                try { Thread.sleep( l: 50); } catch (InterruptedException e) {}
            }
        });
        Thread t2 = new Thread(() -> {
            while (true) {
                boolean gotLock2 = lock2.tryLock();
```



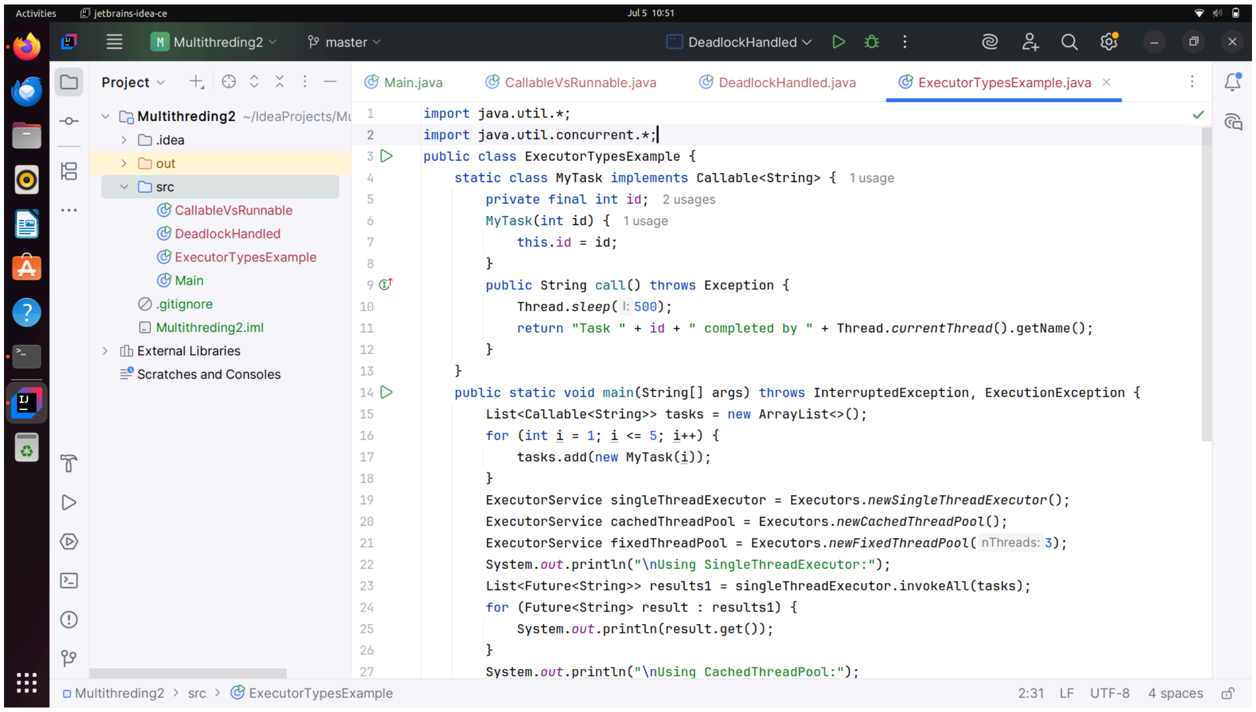
```java
public class DeadlockHandled {
    public static void main(String[] args) {
        });
        Thread t2 = new Thread(() -> {
            while (true) {
                boolean gotLock2 = lock2.tryLock();
                boolean gotLock1 = lock1.tryLock();
                if (gotLock1 && gotLock2) {
                    try {
                        System.out.println("Thread 2: Holding Lock2 & Lock1");
                        break;
                    } finally {
                        lock1.unlock();
                        lock2.unlock();
                    }
                } else {
                    if (gotLock1) lock1.unlock();
                    if (gotLock2) lock2.unlock();
                }

                try { Thread.sleep( l: 50); } catch (InterruptedException e) {}
            }
        });
        t1.start();
        t2.start();
    }
}
```

Output:

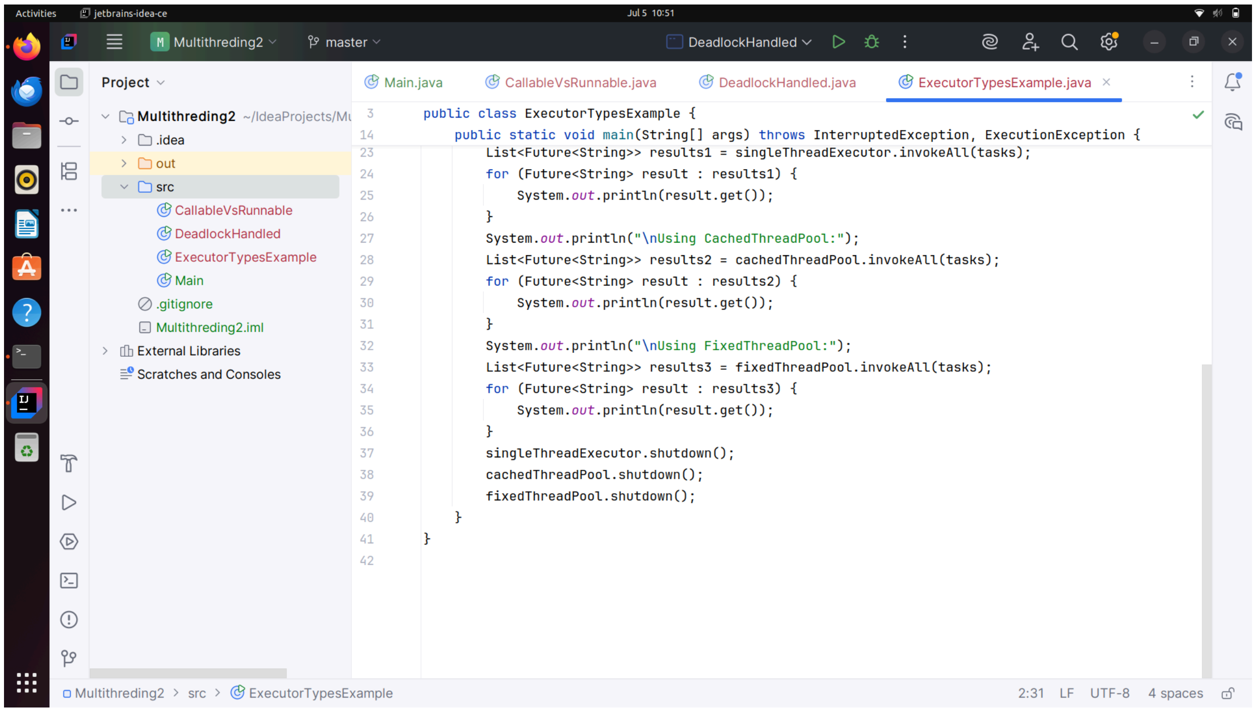


```
/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/java -javaagent:/opt/intellij/lib/idea_rt.jar=45853 -Dfile.encoding=UTF-8 -classpath /home/a
Thread 1: Holding Lock1 & Lock2
Thread 2: Holding Lock2 & Lock1

Process finished with exit code 0
```

3. **Use a singleThreadExecutor, newCachedThreadPool() and newFixedThreadPool() to submit a list of tasks and wait for completion of all tasks.**

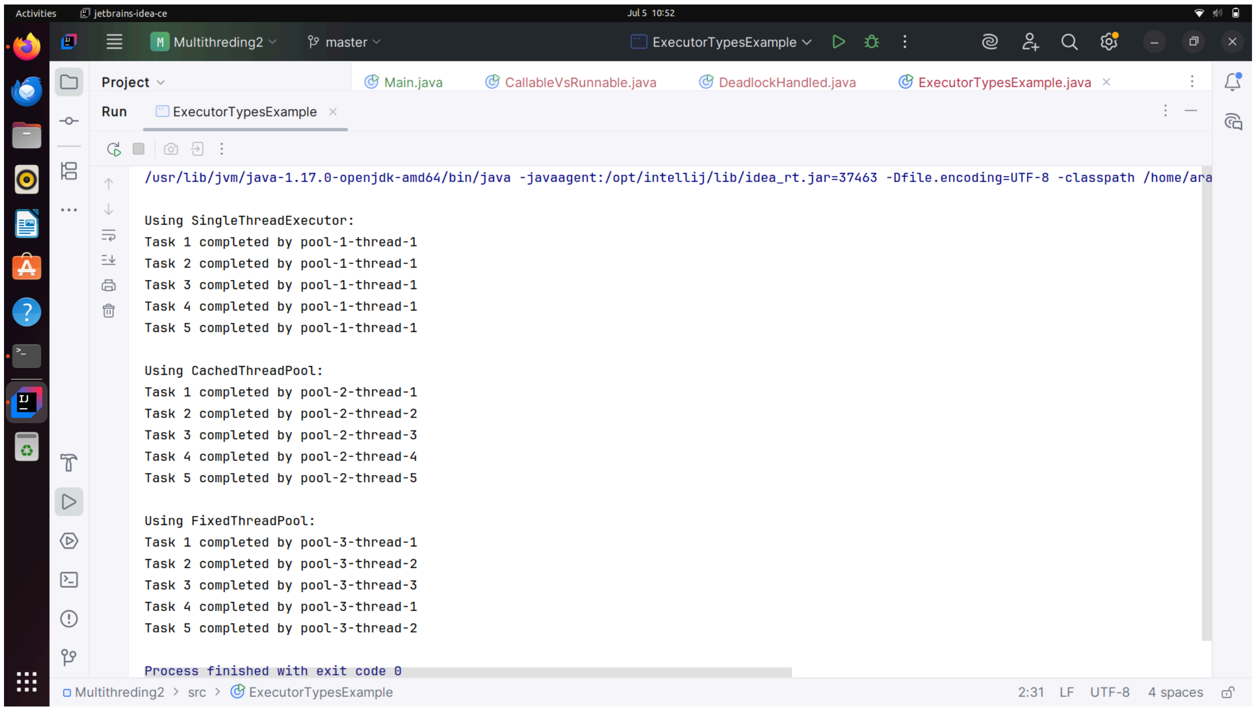


```java
import java.util.*;
import java.util.concurrent.*;
public class ExecutorTypesExample {
    static class MyTask implements Callable<String> {
        private final int id;
        MyTask(int id) {
            this.id = id;
        }
        public String call() throws Exception {
            Thread.sleep(500);
            return "Task " + id + " completed by " + Thread.currentThread().getName();
        }
    }
    public static void main(String[] args) throws InterruptedException, ExecutionException {
        List<Callable<String>> tasks = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            tasks.add(new MyTask(i));
        }
        ExecutorService singleThreadExecutor = Executors.newSingleThreadExecutor();
        ExecutorService cachedThreadPool = Executors.newCachedThreadPool();
        ExecutorService fixedThreadPool = Executors.newFixedThreadPool(3);
        System.out.println("\nUsing SingleThreadExecutor:");
        List<Future<String>> results1 = singleThreadExecutor.invokeAll(tasks);
        for (Future<String> result : results1) {
            System.out.println(result.get());
        }
        System.out.println("\nUsing CachedThreadPool:");
```



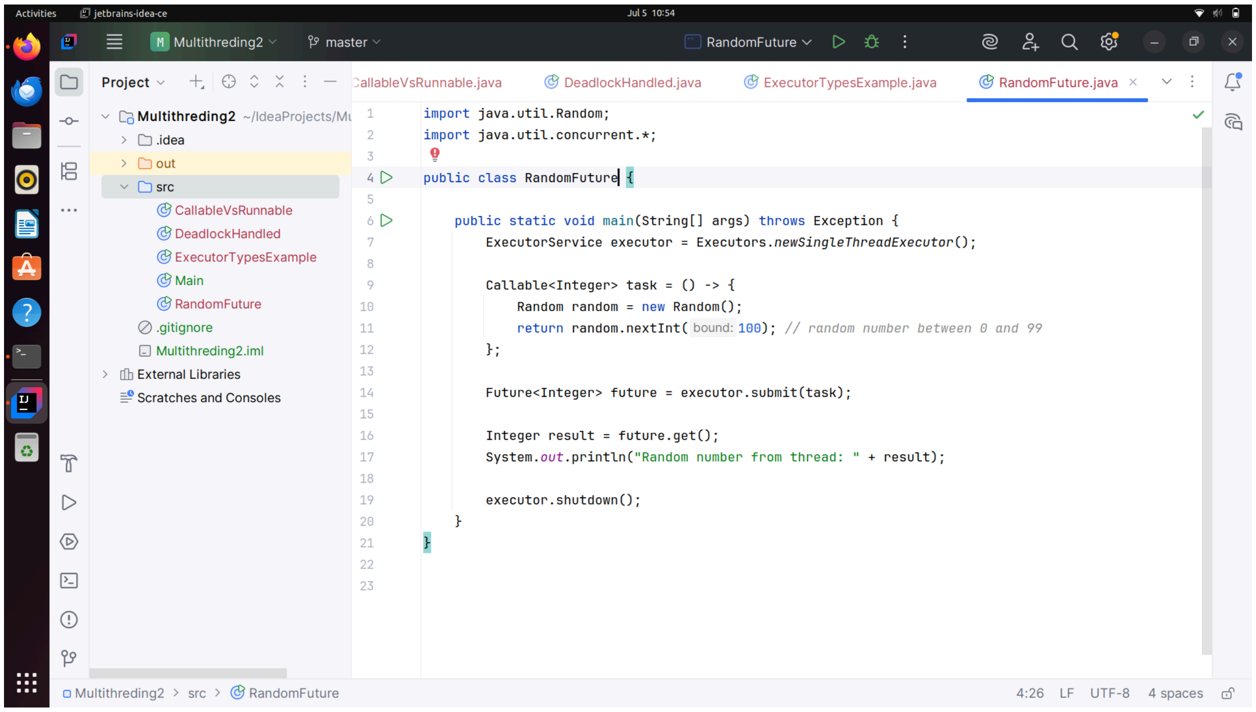
```java
public class ExecutorTypesExample {
    public static void main(String[] args) throws InterruptedException, ExecutionException {
        List<Future<String>> results1 = singleThreadExecutor.invokeAll(tasks);
        for (Future<String> result : results1) {
            System.out.println(result.get());
        }
        System.out.println("\nUsing CachedThreadPool:");
        List<Future<String>> results2 = cachedThreadPool.invokeAll(tasks);
        for (Future<String> result : results2) {
            System.out.println(result.get());
        }
        System.out.println("\nUsing FixedThreadPool:");
        List<Future<String>> results3 = fixedThreadPool.invokeAll(tasks);
        for (Future<String> result : results3) {
            System.out.println(result.get());
        }
        singleThreadExecutor.shutdown();
        cachedThreadPool.shutdown();
        fixedThreadPool.shutdown();
    }
}
```

Output:



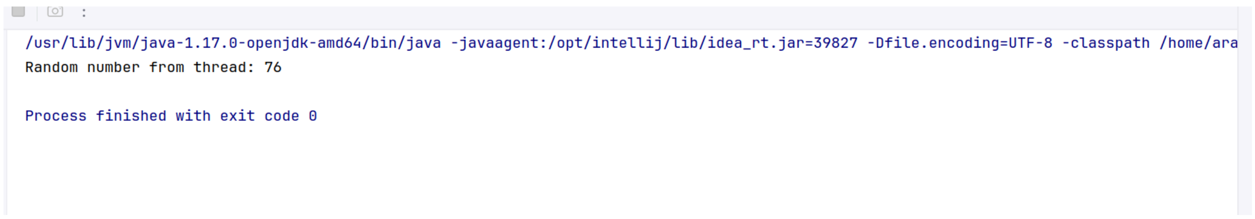
```
/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/java -javaagent:/opt/intellij/lib/idea_rt.jar=37463 -Dfile.encoding=UTF-8 -classpath /home/ara

Using SingleThreadExecutor:
Task 1 completed by pool-1-thread-1
Task 2 completed by pool-1-thread-1
Task 3 completed by pool-1-thread-1
Task 4 completed by pool-1-thread-1
Task 5 completed by pool-1-thread-1

Using CachedThreadPool:
Task 1 completed by pool-2-thread-1
Task 2 completed by pool-2-thread-2
Task 3 completed by pool-2-thread-3
Task 4 completed by pool-2-thread-4
Task 5 completed by pool-2-thread-5

Using FixedThreadPool:
Task 1 completed by pool-3-thread-1
Task 2 completed by pool-3-thread-2
Task 3 completed by pool-3-thread-3
Task 4 completed by pool-3-thread-1
Task 5 completed by pool-3-thread-2

Process finished with exit code 0
```

4. WAP to return a random integert value from a thread execution using Future.



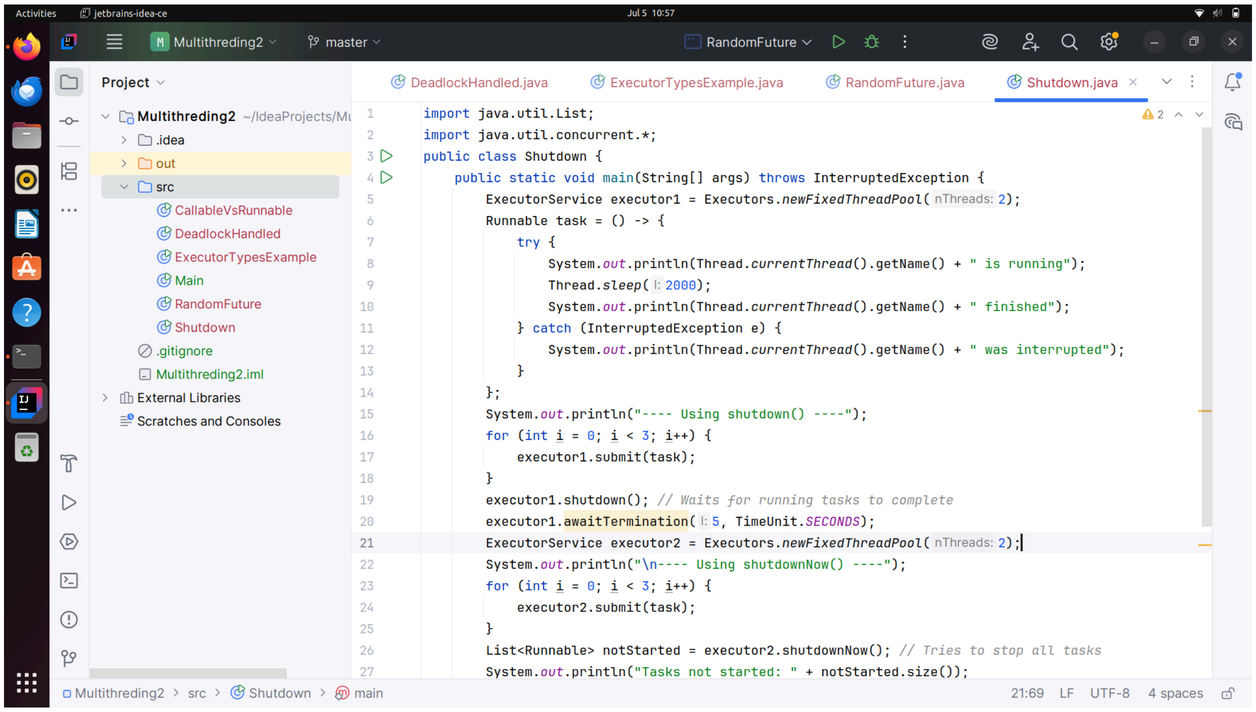
```java
import java.util.Random;
import java.util.concurrent.*;

public class RandomFuture {

    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newSingleThreadExecutor();

        Callable<Integer> task = () -> {
            Random random = new Random();
            return random.nextInt( bound: 100); // random number between 0 and 99
        };

        Future<Integer> future = executor.submit(task);

        Integer result = future.get();
        System.out.println("Random number from thread: " + result);

        executor.shutdown();
    }
}
```

Output:

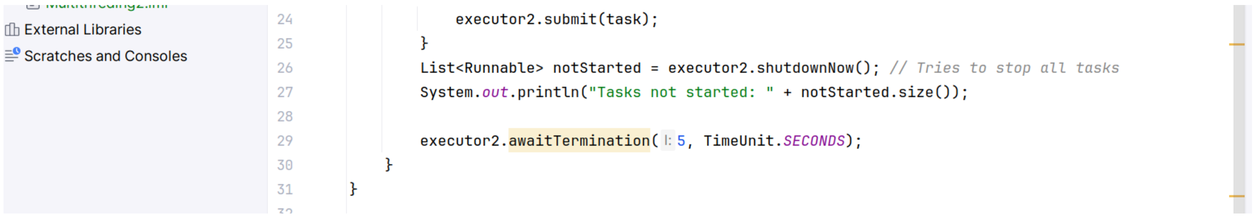


```
/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/java -javaagent:/opt/intellij/lib/idea_rt.jar=39827 -Dfile.encoding=UTF-8 -classpath /home/ara
Random number from thread: 76

Process finished with exit code 0
```

5. WAP to showcase the difference between shutdown() and shutdownNow().

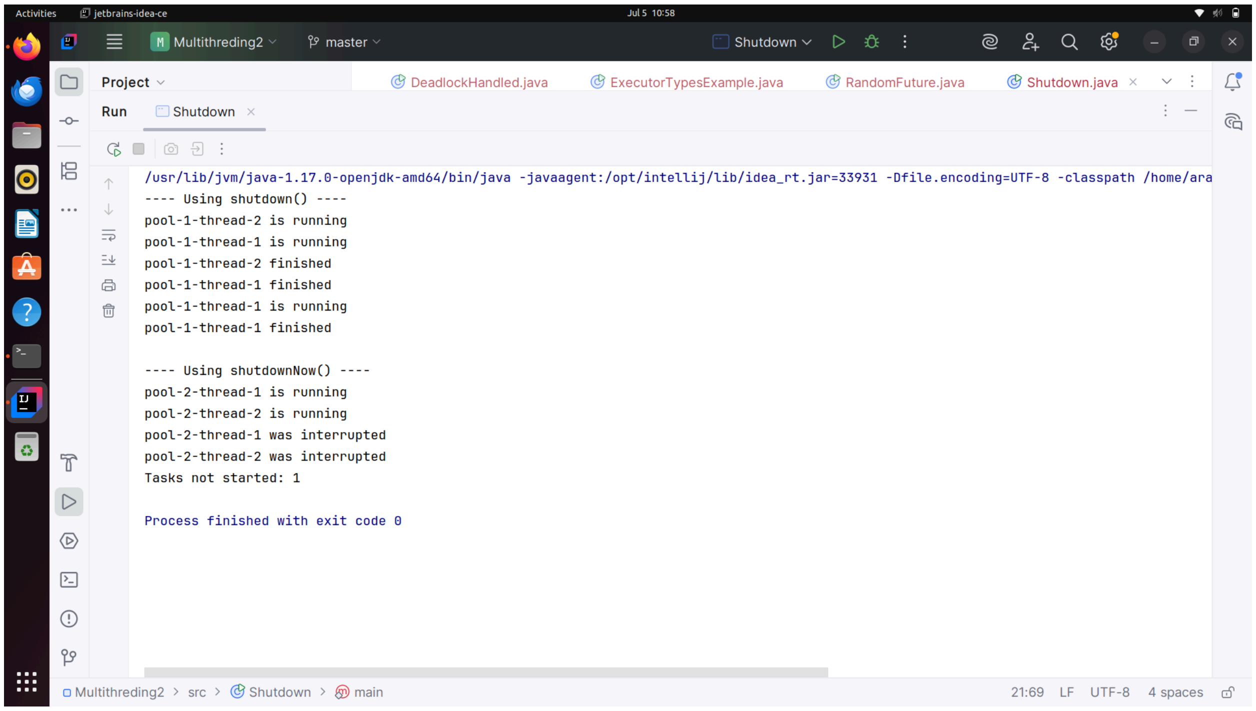


```java
import java.util.List;
import java.util.concurrent.*;
public class Shutdown {
    public static void main(String[] args) throws InterruptedException {
        ExecutorService executor1 = Executors.newFixedThreadPool( nThreads: 2);
        Runnable task = () -> {
            try {
                System.out.println(Thread.currentThread().getName() + " is running");
                Thread.sleep( l: 2000);
                System.out.println(Thread.currentThread().getName() + " finished");
            } catch (InterruptedException e) {
                System.out.println(Thread.currentThread().getName() + " was interrupted");
            }
        };
        System.out.println("---- Using shutdown() ----");
        for (int i = 0; i < 3; i++) {
            executor1.submit(task);
        }
        executor1.shutdown(); // Waits for running tasks to complete
        executor1.awaitTermination( l: 5, TimeUnit.SECONDS);
        ExecutorService executor2 = Executors.newFixedThreadPool( nThreads: 2);
        System.out.println("\n---- Using shutdownNow() ----");
        for (int i = 0; i < 3; i++) {
            executor2.submit(task);
        }
        List<Runnable> notStarted = executor2.shutdownNow(); // Tries to stop all tasks
        System.out.println("Tasks not started: " + notStarted.size());
```



```java
            executor2.submit(task);
        }
        List<Runnable> notStarted = executor2.shutdownNow(); // Tries to stop all tasks
        System.out.println("Tasks not started: " + notStarted.size());

        executor2.awaitTermination( l: 5, TimeUnit.SECONDS);
    }
}
```

Output:



```
/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/java -javaagent:/opt/intellij/lib/idea_rt.jar=33931 -Dfile.encoding=UTF-8 -classpath /home/ara
---- Using shutdown() ----
pool-1-thread-2 is running
pool-1-thread-1 is running
pool-1-thread-2 finished
pool-1-thread-1 finished
pool-1-thread-1 is running
pool-1-thread-1 finished

---- Using shutdownNow() ----
pool-2-thread-1 is running
pool-2-thread-2 is running
pool-2-thread-1 was interrupted
pool-2-thread-2 was interrupted
Tasks not started: 1

Process finished with exit code 0
```