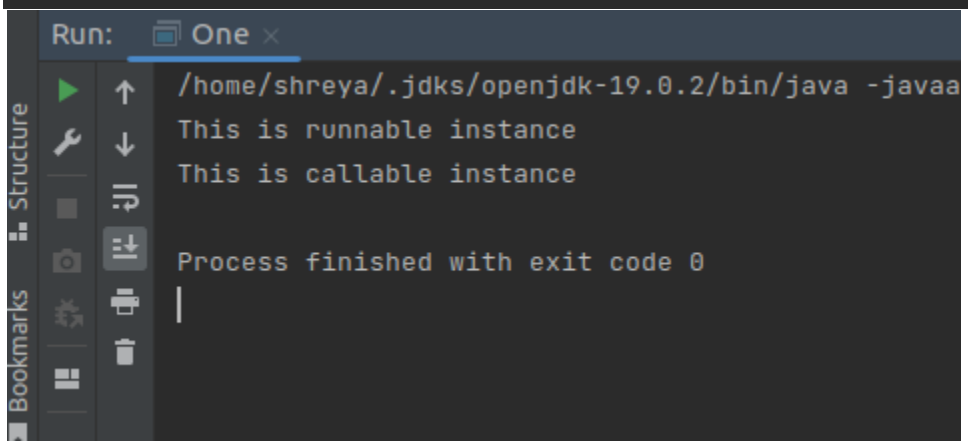
Basics of Multithreading part-2

Assignment

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

```
import java.util.concurrent.*;
public class One {
        public static void main(String[] args) throws ExecutionException,
InterruptedException {
            ExecutorService executor = Executors.newSingleThreadExecutor();
            Runnable runnable = () -> {
                System.out.println("This is runnable instance");
            };
            executor.execute(runnable);
            Callable<String> callable = () -> {
                return "This is callable instance";
            };
            Future<String> future = executor.submit(callable);
            String result = future.get();
            System.out.println(result);
            executor.shutdown();
        }
}
```



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

```
import java.util.concurrent.locks.ReentrantLock;

public class BankAccount {
   private float balance;
   public static ReentrantLock lock = new ReentrantLock();
   private int accountNumber;

   public BankAccount(int accountNumber, float balance) {
       this.accountNumber = accountNumber;
       this.balance = balance;
   }
```

```
   public void withdrawal(float amount) {
       lock.lock();

       try{
       if (balance < amount) {
           System.out.println("Not enough balance");
           return;
       } else {
           balance -= amount;
           System.out.println("Withdrawal of amount " + amount + " is
successful. Balance is " + balance);
       }
   }
 finally {
       System.out.println("lock released "+Thread.currentThread().getName());
       lock.unlock();
   }
}
   public void deposit(float amount) {
       lock.lock();
       try {
           balance += amount;
           System.out.println("Balance is " + balance);
       }
       finally {
           System.out.println("lock released
"+Thread.currentThread().getName());
           lock.unlock();
       }
   }
}
public class LockEx{
   public static void main(String[] args){
       BankAccount account1=new BankAccount(1234,344.56F);
       BankAccount account2=new BankAccount(1235,344.56F);
       new Thread(new Runnable() {
           @Override
           public void run() {
               account1.withdrawal(45);
               account2.deposit(12334);
           }
       }).start();
       new Thread(new Runnable() {
           @Override
           public void run() {
               account2.withdrawal(45);
               account1.deposit(1223);
           }
```

```
        }).start();
    }
}
```

```
DeadlockEx   LockEx
/home/shreya/.jdks/openjdk-19.0.2/bin/java -javaagent:/snap/intellij-idea-community/409/lib/idea_rt
Withdrawal of amount 45.0 is successful. Balance is 299.56
lock released Thread-0
Balance is 12678.56
lock released Thread-0
Withdrawal of amount 45.0 is successful. Balance is 12633.56
lock released Thread-1
Balance is 1522.56
lock released Thread-1

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.Arrays;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ThreadPoolEx {
   public static void main(String[] args) throws InterruptedException {
       List<Runnable> tasks= Arrays.asList(
               ()->System.out.println("Task 1"),
               ()->System.out.println("Task 2"),
               ()->System.out.println("Task 3"),
               ()->System.out.println("Task 4")
       );
       ExecutorService single= Executors.newSingleThreadExecutor();
       ExecutorService cached=Executors.newCachedThreadPool();
       ExecutorService fixed=Executors.newFixedThreadPool(2);
       for(Runnable task: tasks){
           single.submit(task);
           cached.submit(task);
           fixed.submit(task);
       }
       single.shutdown();
       cached.shutdown();
       fixed.shutdown();
```

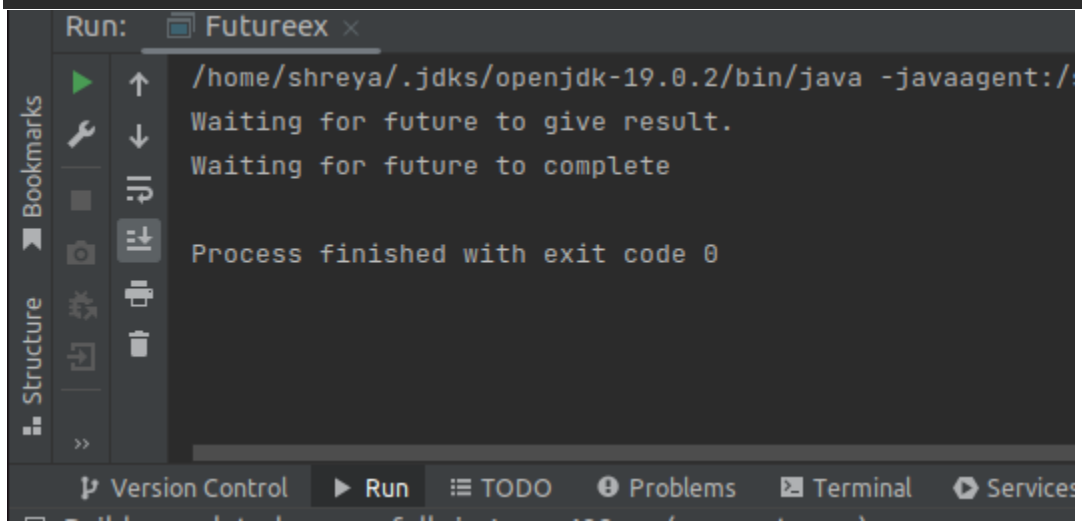
```
        single.awaitTermination(100, TimeUnit.MILLISECONDS);
        cached.awaitTermination(100, TimeUnit.MILLISECONDS);
        fixed.awaitTermination(100, TimeUnit.MILLISECONDS);
    }
}
```

```
Run:  ThreadPoolEx
/home/shreya/.jdks/openjdk-19.0.2/bin/java -javaagent:/snap/intellij
Task 1
Task 2
Task 1
Task 1
Task 2
Task 3
Task 2
Task 3
Task 4
Task 3
Task 4
Task 4

Process finished with exit code 0
```

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

```
import java.util.Random;
import java.util.concurrent.*;
public class Futureex {
   public static void main(String[] args){
   ExecutorService executor = Executors.newSingleThreadExecutor();

   Future<Integer> futureRandom= executor.submit(new Callable<Integer>() {
       @Override
       public Integer call() throws Exception {
           Random num=new Random();
           int randomNum=num.nextInt(100);
           Thread.sleep(100);
           return randomNum;
       }
   }) ;
   System.out.println("Waiting for future to give result.");
       if(futureRandom.isDone()){
       try {
           System.out.println("Future is complete  "+futureRandom.get());
       } catch (InterruptedException e) {
```

```
            throw new RuntimeException(e);
        } catch (ExecutionException e) {
            throw new RuntimeException(e);
        }
    }
    else  System.out.println("Waiting for future to complete");
    executor.shutdown();
}
}
```



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

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class Fifth {
    public static void main(String[] args) throws InterruptedException {
        ExecutorService executor = Executors.newFixedThreadPool(2);
        for (int i = 1; i < 4; i++) {
            executor.submit(new Task(i));
        }
        Thread.sleep(1000);
        executor.shutdown();
        System.out.println("Called shutdown ");
        while(!executor.isTerminated()){
            System.out.println("Waiting for termination.");
        executor.awaitTermination(500, TimeUnit.MILLISECONDS);
    }
        executor=Executors.newFixedThreadPool(2);
        for (int i = 1; i < 4; i++) {
            executor.submit(new Task(i));
        }
        Thread.sleep(1000);

        executor.shutdownNow();
```

```java
        System.out.println("Called shutdownNow ");
        while(!executor.isTerminated()){
            System.out.println("Waiting for termination.");
            executor.awaitTermination(500, TimeUnit.MILLISECONDS);
        }
        System.out.println("All Tasks finished..");
    }
}
```

```
Run:  Fifth
/home/shreya/.jdks/openjdk-19.0.2/bin/java -javaagent:/snap/intellij
Starting Task 1
Starting Task 2
Finished task 2
Finished task 1
Starting Task 3
Called shutdown
Waiting for termination.
Finished task 3
Starting Task 1
Starting Task 2
Finished task 1
Finished task 2
Starting Task 3
Called shutdownNow
Waiting for termination.
Interruption occurred.
All Tasks finished..

Process finished with exit code 0
```