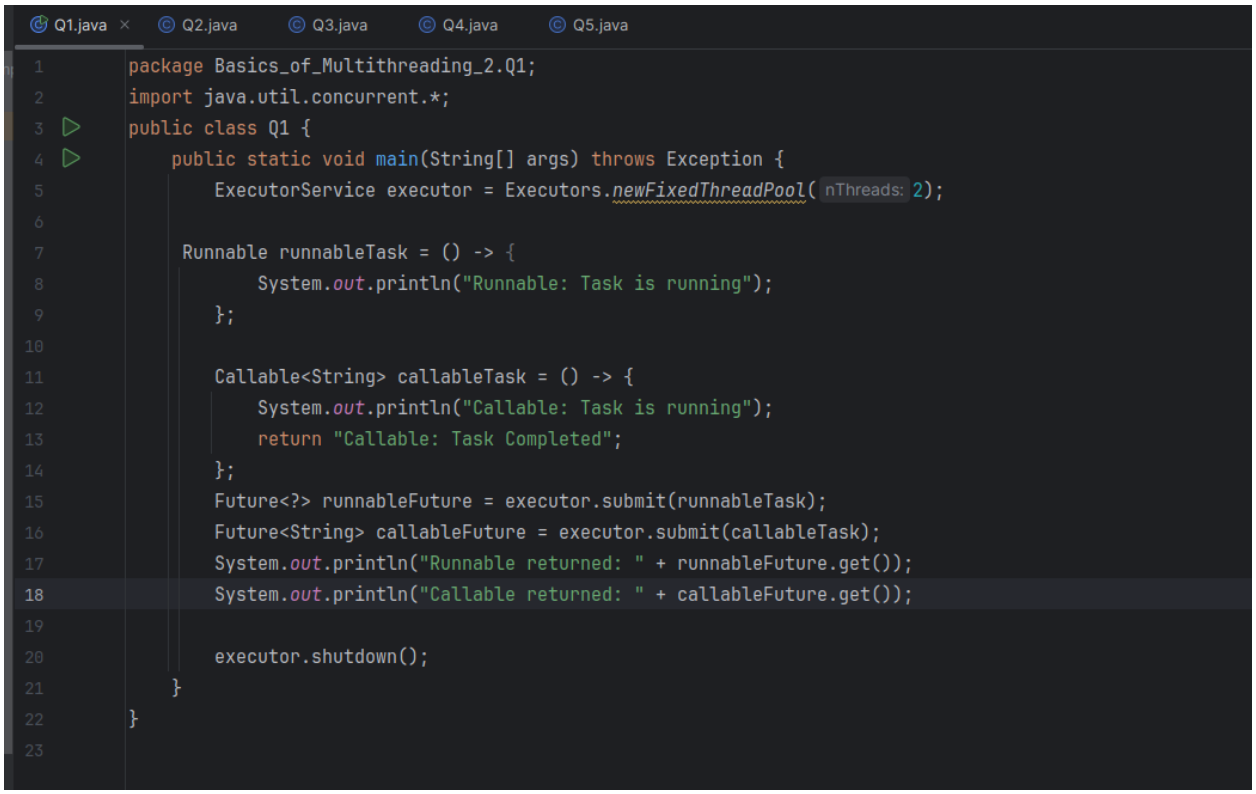
Assignment - Basics of Multithreading 2

Q1)WAP to show usage of Callable and demonstrate how it is different from Runnable



```java
package Basics_of_Multithreading_2.Q1;
import java.util.concurrent.*;
public class Q1 {
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 2);

     Runnable runnableTask = () -> {
            System.out.println("Runnable: Task is running");
        };

        Callable<String> callableTask = () -> {
            System.out.println("Callable: Task is running");
            return "Callable: Task Completed";
        };
        Future<?> runnableFuture = executor.submit(runnableTask);
        Future<String> callableFuture = executor.submit(callableTask);
        System.out.println("Runnable returned: " + runnableFuture.get());
        System.out.println("Callable returned: " + callableFuture.get());

        executor.shutdown();
    }
}
```

```
/usr/lib/jvm/java-1.21.0-openjdk-amd64/bin/java -javaagent:/home/akash/Downloads/idea-IU-251.26094.121/li
Callable: Task is running
Runnable: Task is running
Runnable returned: null
Callable returned: Callable: Task Completed

Process finished with exit code 0
```

Q2)Improve the code written in Basics of Multi Threading Part 1 exercise question 4 to handle the deadlock using reentract lock.

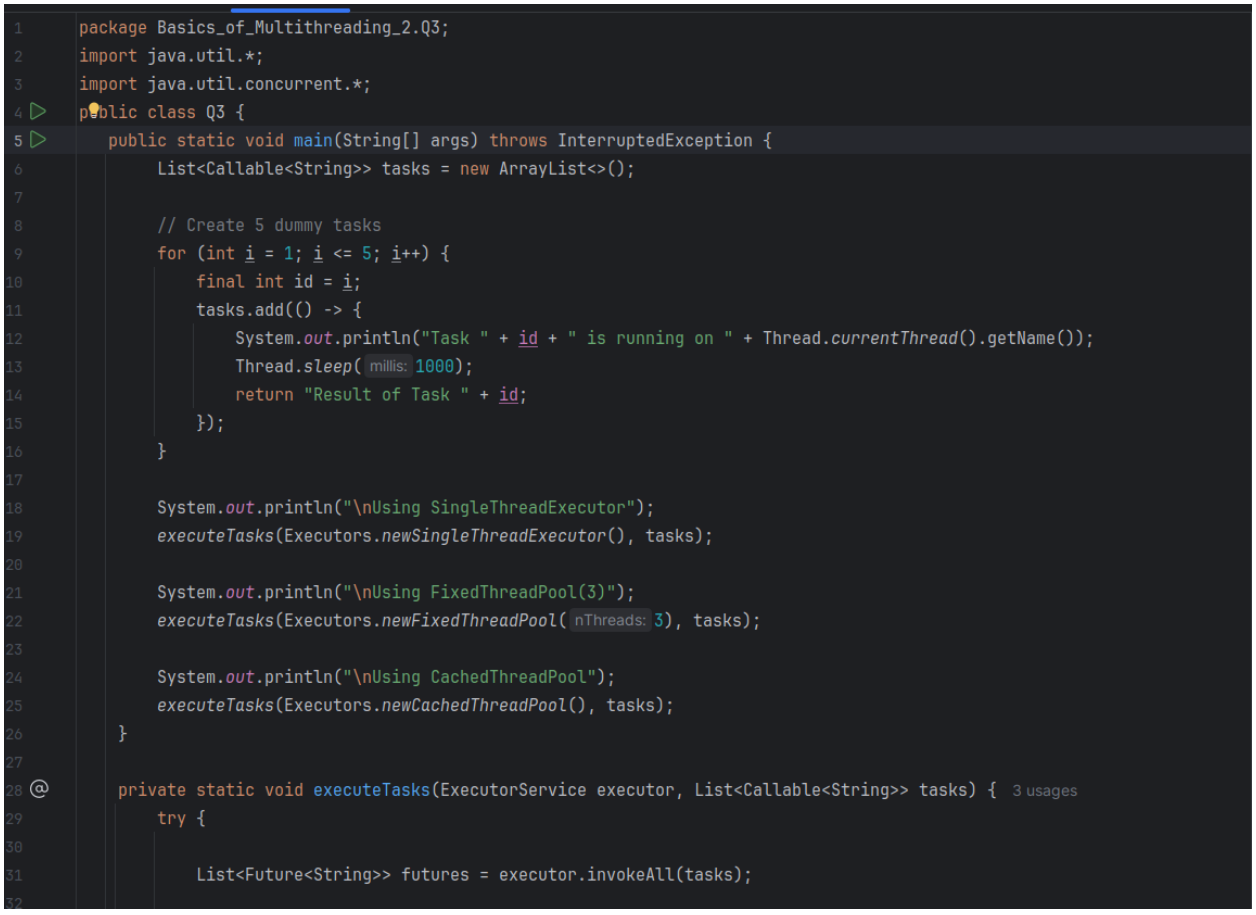
```java
package Basics_of_Multithreading_2.Q2;


import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.TimeUnit;
public class Q2 {

    public static void main(String[] args) throws InterruptedException {
        Account a1 = new Account( accountNumber: "1234",  amount: 100000.0);
        Account a2 = new Account( accountNumber: "5678",  amount: 200000.0);
        Q2 transferHelper = new Q2();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 500; i++) {
                transferHelper.transfer(a1, a2,  amount: 100.0);
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 500; i++) {
                transferHelper.transfer(a2, a1,  amount: 100.0);
            }
        });

        t1.start();
        t2.start();
        t1.join();
        t2.join();

        System.out.println("Final A1 Balance: " + a1.getAmount());
        System.out.println("Final A2 Balance: " + a2.getAmount());
```

```java
public class Q2 {
    }

    private void transfer(Account from, Account to, double amount) {  2 usages
        try {
            if (from.getLock().tryLock( timeout: 100, TimeUnit.MILLISECONDS)) {
                try {
                    if (to.getLock().tryLock( timeout: 100, TimeUnit.MILLISECONDS)) {
                        try {
                            if (from.getAmount() >= amount) {
                                from.setAmount(from.getAmount() - amount);
                                to.setAmount(to.getAmount() + amount);
                            }
                        } finally {
                            to.getLock().unlock();
                        }
                    }
                } finally {
                    from.getLock().unlock();
                }
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt(); // Best practice
        }
    }
}


class Account {  6 usages
    private String accountNumber;  2 usages
    private double amount;  3 usages
```

```java
class Account {  6 usages
    private String accountNumber;  2 usages
    private double amount;  3 usages
    private final ReentrantLock lock = new ReentrantLock();  1 usage

    public Account(String accountNumber, double amount) {  2 usages
        this.accountNumber = accountNumber;
        this.amount = amount;
    }

    public String getAccountNumber() {  no usages
        return accountNumber;
    }

    public double getAmount() {  5 usages
        return amount;
    }

    public void setAmount(double amount) {  2 usages
        this.amount = amount;
    }

    public ReentrantLock getLock() {  4 usages
        return lock;
    }
}

```

```
/usr/lib/jvm/java-1.21.0-openjdk-amd64/bin/java -javaagent:/home/akash/Downloads/
Final A1 Balance: 99900.0
Final A2 Balance: 200100.0

Process finished with exit code 0
```

Q3)Use a singleThreadExecutor, newCachedThreadPool() and newFixedThreadPool() to submit a list of tasks and wait for completion of all tasks.



```java
package Basics_of_Multithreading_2.Q3;
import java.util.*;
import java.util.concurrent.*;
public class Q3 {
    public static void main(String[] args) throws InterruptedException {
        List<Callable<String>> tasks = new ArrayList<>();

        // Create 5 dummy tasks
        for (int i = 1; i <= 5; i++) {
            final int id = i;
            tasks.add(() -> {
                System.out.println("Task " + id + " is running on " + Thread.currentThread().getName());
                Thread.sleep( millis: 1000);
                return "Result of Task " + id;
            });
        }

        System.out.println("\nUsing SingleThreadExecutor");
        executeTasks(Executors.newSingleThreadExecutor(), tasks);

        System.out.println("\nUsing FixedThreadPool(3)");
        executeTasks(Executors.newFixedThreadPool( nThreads: 3), tasks);

        System.out.println("\nUsing CachedThreadPool");
        executeTasks(Executors.newCachedThreadPool(), tasks);
    }

    private static void executeTasks(ExecutorService executor, List<Callable<String>> tasks) {  3 usages
        try {

            List<Future<String>> futures = executor.invokeAll(tasks);
```

```java
public class Q3 {
    public static void main(String[] args) throws InterruptedException {
        executeTasks(Executors.newSingleThreadExecutor(), tasks);

        System.out.println("\nUsing FixedThreadPool(3)");
        executeTasks(Executors.newFixedThreadPool( nThreads: 3), tasks);

        System.out.println("\nUsing CachedThreadPool");
        executeTasks(Executors.newCachedThreadPool(), tasks);
    }

    private static void executeTasks(ExecutorService executor, List<Callable<String>> tasks) {  3 usages
        try {

            List<Future<String>> futures = executor.invokeAll(tasks);

            for (Future<String> future : futures) {
                System.out.println("Completed: " + future.get());
            }
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        } finally {
            executor.shutdown();
        }
    }
}
```

```
/usr/lib/jvm/java-1.21.0-openjdk-amd64/bin/java -javaagent:/home/akash/Downloads/idea-IU-251.26

Using SingleThreadExecutor
Task 1 is running on pool-1-thread-1
Task 2 is running on pool-1-thread-1
Task 3 is running on pool-1-thread-1
Task 4 is running on pool-1-thread-1
Task 5 is running on pool-1-thread-1
Completed: Result of Task 1
Completed: Result of Task 2
Completed: Result of Task 3
Completed: Result of Task 4
Completed: Result of Task 5

Using FixedThreadPool(3)
Task 1 is running on pool-2-thread-1
Task 2 is running on pool-2-thread-2
Task 3 is running on pool-2-thread-3
Task 4 is running on pool-2-thread-1
Task 5 is running on pool-2-thread-2
Completed: Result of Task 1
Completed: Result of Task 2
Completed: Result of Task 3
Completed: Result of Task 4
Completed: Result of Task 5
```

```
Using CachedThreadPool
Task 1 is running on pool-3-thread-1
Task 2 is running on pool-3-thread-2
Task 3 is running on pool-3-thread-3
Task 4 is running on pool-3-thread-4
Task 5 is running on pool-3-thread-5
Completed: Result of Task 1
Completed: Result of Task 2
Completed: Result of Task 3
Completed: Result of Task 4
Completed: Result of Task 5

Process finished with exit code 0
```

Q4)WAP to return a random integert value from a thread execution using Future.

```java
public class Q4 {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newSingleThreadExecutor();

        Callable<Integer> task = () -> {
            Random rand = new Random();
            int result = rand.nextInt( bound: 100);
            System.out.println("Generated number: " + result + " on thread " + Thread.currentThread().getName());
            return result;
        };

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

        try {
            Integer randomValue = future.get();
            System.out.println("Result from thread: " + randomValue);
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        } finally {
            executor.shutdown();
        }
    }
}
```

```
/usr/lib/jvm/java-1.21.0-openjdk-amd64/bin/java -javaagent:/home/akash/Downloads/idea-IU-251.26094.121
Generated number: 78 on thread pool-1-thread-1
Result from thread: 78

Process finished with exit code 0
```

Q5)WAP to showcase the difference between shutdown() and shutdownNow().

```
public class Q5 {
    public static void main(String[] args) throws InterruptedException {
            });
        }

        executor1.shutdown();
        System.out.println("Executor 1 shutdown called\n");

        Thread.sleep( millis: 3000);

        System.out.println("Using shutdownNow()");
        for (int i = 4; i <= 6; i++) {
            int taskId = i;
            executor2.submit(() -> {
                System.out.println("Task " + taskId + " started (shutdownNow)");
                try {
                    Thread.sleep( millis: 4000);
                } catch (InterruptedException e) {
                    System.out.println("Task " + taskId + " interrupted (shutdownNow)");
                }
                System.out.println("Task " + taskId + " finished (shutdownNow)");
            });
        }

        List<Runnable> pending = executor2.shutdownNow();
        System.out.println("Executor 2 shutdownNow called. Pending tasks: " + pending.size());
    }
}
```

```
package Basics_of_Multithreading_2.Q5;
import java.util.List;
import java.util.concurrent.*;

public class Q5 {
    public static void main(String[] args) throws InterruptedException {
        ExecutorService executor1 = Executors.newFixedThreadPool( nThreads: 2);
        ExecutorService executor2 = Executors.newFixedThreadPool( nThreads: 2);

        System.out.println("Using shutdown()");
        for (int i = 1; i <= 3; i++) {
            int taskId = i;
            executor1.submit(() -> {
                System.out.println("Task " + taskId + " started (shutdown)");
                try {
                    Thread.sleep( millis: 2000);
                } catch (InterruptedException e) {
                    System.out.println("Task " + taskId + " interrupted (shutdown)");
                }
                System.out.println("Task " + taskId + " finished (shutdown)");
            });
        }

        executor1.shutdown();
        System.out.println("Executor 1 shutdown called\n");
```

```
/usr/lib/jvm/java-1.21.0-openjdk-amd64/bin/java -javaagent:/home/akash/Downloads/idea-IU-251.26094.121/lib/idea_rt.ja
Using shutdown()
Executor 1 shutdown called

Task 1 started (shutdown)
Task 2 started (shutdown)
Task 1 finished (shutdown)
Task 2 finished (shutdown)
Task 3 started (shutdown)
Using shutdownNow()
Task 4 started (shutdownNow)
Task 5 started (shutdownNow)
Executor 2 shutdownNow called. Pending tasks: 1
Task 5 interrupted (shutdownNow)
Task 4 interrupted (shutdownNow)
Task 5 finished (shutdownNow)
Task 4 finished (shutdownNow)
Task 3 finished (shutdown)

Process finished with exit code 0
```