

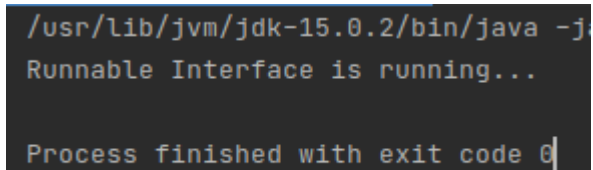
Basics of Multi Threading Exercise

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

Sol - Program File-Ques1.java

Runnable Interface

```
class Multi3 implements Runnable {  
    public void run() {  
        System.out.println("Runnable Interface is running...");  
    }  
}  
  
public class Ques1RunnableInterface {  
    public static void main(String[] args) {  
        Multi3 m1=new Multi3();  
        Thread t1 =new Thread(m1);  
        t1.start();  
    }  
}
```



```
/usr/lib/jvm/jdk-15.0.2/bin/java -j  
Runnable Interface is running...  
  
Process finished with exit code 0
```

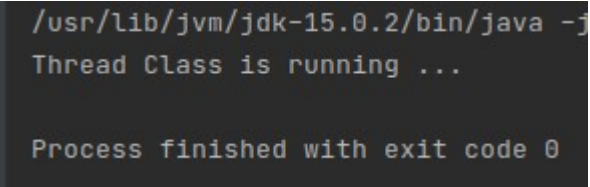
Thread class

```
class Multi extends Thread {  
    public void run() {  
        System.out.println("Thread Class is running ...");  
    }  
}  
  
public class Ques1ThreadClass {  
    public static void main(String[] args) {  
        Multi t1=new Multi();
```

```

        t1.start();
    }
}

```



```

/usr/lib/jvm/jdk-15.0.2/bin/java -j
Thread Class is running ...

Process finished with exit code 0

```

2. Use sleep and join methods with thread.

Sol - Program File-Ques2.java

```

public class Ques2 extends Thread {
    @Override
    public void run() {
        for (int i=1; i<=2; i++)
        {
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(i);
        }
    }

    public static void main(String[] args) {
        Ques2 t1 = new Ques2();
        Ques2 t2 = new Ques2();
        Ques2 t3 = new Ques2();
    }
}

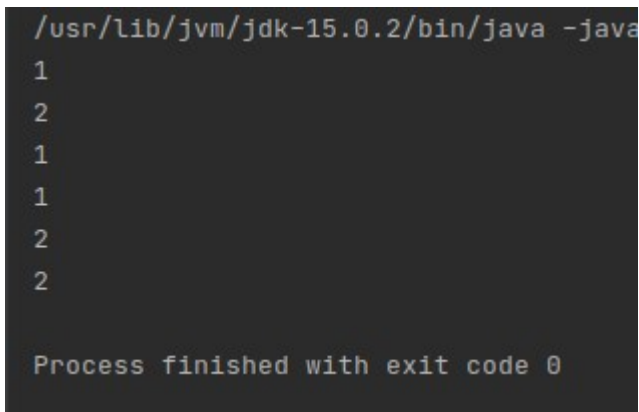
```

```

        t1.start();
        try {
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        t2.start();
        t3.start();

    }
}

```



```

/usr/lib/jvm/jdk-15.0.2/bin/java -java
1
2
1
1
2
2
Process finished with exit code 0

```

3. Use a singleThreadExecutor to submit multiple threads.

Sol - Program File-Ques3.java

```

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

public class Ques3 {
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newFixedThreadPool(2);
        Runnable task1 = new Runnable() {
            @Override
            public void run() {

```

```

        System.out.println(Thread.currentThread().getName());
        System.out.println("My task1 started..");
        try {
            TimeUnit.SECONDS.sleep(2);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("My task1 end..");
    }
};

Runnable task2 = new Runnable() {
    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName());
        System.out.println("My task2 started..");
        try {
            TimeUnit.SECONDS.sleep(2);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("My task2 end..");
    }
};

```

```

Runnable task3 = new Runnable() {
    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName());
        System.out.println("My task3 started..");
        try {

```

```

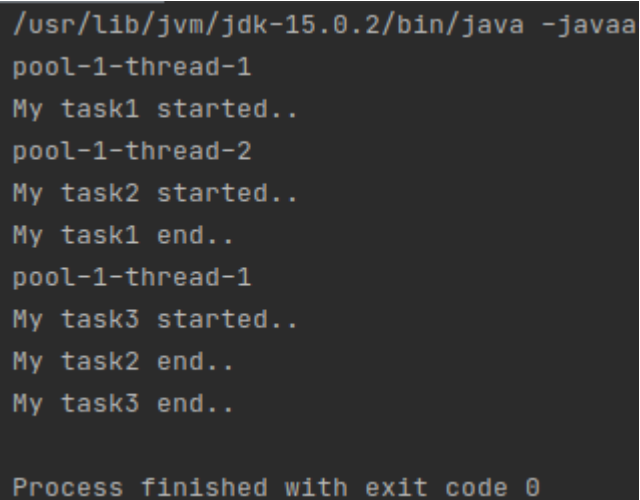
        TimeUnit.SECONDS.sleep(2);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("My task3 end..");
}
};

```

```

    executorService.submit(task1);
    executorService.submit(task2);
    executorService.submit(task3);
    executorService.shutdown();
}
}

```



```

/usr/lib/jvm/jdk-15.0.2/bin/java -javaa
pool-1-thread-1
My task1 started..
pool-1-thread-2
My task2 started..
My task1 end..
pool-1-thread-1
My task3 started..
My task2 end..
My task3 end..

Process finished with exit code 0

```

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

Sol - Program File-Ques4.java

shutdown() initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted. This method does not wait for previously submitted tasks (but not started executing) to complete execution.

ShutdownNow() attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.

awaitTermination(long timeout, TimeUnit unit) blocks until all tasks have completed execution **after a shutdown request**, or the timeout occurs, or the current thread is interrupted, whichever happens first.

- In short, shutdown() means the executor service takes no more incoming tasks.
- Remember that `awaitTermination()` is invoked after a `shutdown()` request.

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

```
public class Ques4 {
    public static void main(String[] args) {
        try {
            testShutDown(100);
            testShutDownNow(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
public static void testShutDown(int startNo) throws InterruptedException {
    ExecutorService executorService = Executors.newFixedThreadPool(2);
    for (int i = 0; i < 5; i++) {
```

```

        executorService.execute(getTask(i + startNo));
    }
    executorService.shutdown();
    executorService.awaitTermination(1, TimeUnit.DAYS);
    System.out.println("shutDown - all thread shutdown");
}

```

```

public static void testShutDownNow(int startNo) throws InterruptedException
{
    ExecutorService executorService = Executors.newFixedThreadPool(2);
    for (int i = 0; i < 5; i++) {
        executorService.execute(getTask(i + startNo));
    }
    executorService.shutdownNow();
    executorService.awaitTermination(1, TimeUnit.DAYS);
    System.out.println("shutdownNow - all thread shutdown");
}

```

```

public static Runnable getTask(int threadNo) {
    final Random random = new Random();
    final int no = threadNo;
    Runnable task = new Runnable() {
        @Override
        public void run() {
            try {
                System.out.println(no + " - " + random.nextInt(10));
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

```

```

    };
    return task;
}
}

```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -javaagent:/snap/intelli
100 - 7
101 - 5
102 - 2
103 - 0
104 - 7
shutDown - all thread shutdown
200 - 5
201 - 0
java.lang.InterruptedException: sleep interrupted
java.lang.InterruptedException: sleep interrupted
shutdownNow - all thread shutdown

Process finished with exit code 0

```

5. Use `isShutDown()` and `isTerminated()` with `ExecutorService`.

Sol - Program File-Ques5.java

```

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

public class Ques5 {
    public static void main(String[] args) {
        try {
            testShutDown(100);
            testShutDownNow(200);
        } catch (InterruptedException e) {

```



```
        e.printStackTrace();
    }
}
```

```
public static void testShutDown(int startNo) throws InterruptedException {
    ExecutorService executorService = Executors.newFixedThreadPool(2);
    for (int i = 0; i < 5; i++) {
        executorService.execute(getTask(i + startNo));
    }
    executorService.shutdown();
    executorService.awaitTermination(1, TimeUnit.DAYS);
    System.out.println("shutDown - all thread shutdown");
}
```

//isTerminated() Method

```
System.out.println(" Is Executor isTerminated :"+
executorService.isTerminated());
```

//isShutdown() Method

```
System.out.println(" Is Executor isShutdown :"+
executorService.isShutdown());
```

```
}
```

```
public static void testShutDownNow(int startNo) throws InterruptedException
{
```

```
    ExecutorService executorService = Executors.newFixedThreadPool(2);
    for (int i = 0; i < 5; i++) {
        executorService.execute(getTask(i + startNo));
    }
    executorService.shutdownNow();
}
```

//isTerminated() Method

```
        System.out.println(" Is Executor isTerminated :"+
executorService.isTerminated());
```

```
        //isShutdown() Method
```

```
        System.out.println(" Is Executor isShutdown :"+
executorService.isShutdown());
```

```
        executorService.awaitTermination(1, TimeUnit.DAYS);
```

```
        System.out.println("shutdownNow - all thread shutdown");
```

```
    }
```

```
public static Runnable getTask(int threadNo) {
```

```
    final Random random = new Random();
```

```
    final int no = threadNo;
```

```
    Runnable task = new Runnable() {
```

```
        @Override
```

```
        public void run() {
```

```
            try {
```

```
                System.out.println(no + " - " + random.nextInt(10));
```

```
                Thread.sleep(1000);
```

```
            } catch (InterruptedException e) {
```

```
                System.out.println(e);
```

```
            }
```

```
        }
```

```
    };
```

```
    return task;
```

```
}
```

```
}
```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -javaagent:/snap/1
100 - 4
101 - 0
102 - 8
103 - 3
104 - 8
shutdown - all thread shutdown
  Is Executor isTerminated :true
  Is Executor isShutdown :true
200 - 7
java.lang.InterruptedException: sleep interrupted
  Is Executor isTerminated :false
201 - 7
  Is Executor isShutdown :true
java.lang.InterruptedException: sleep interrupted
shutdownNow - all thread shutdown

Process finished with exit code 0

```

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.

Sol - Program File-Ques6.java

```
import java.util.Random;
```

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

```
public class Ques6 {
```

```
    public static void main(String[] args)
```

```
    {
```

```
        ExecutorService executorService = Executors.newCachedThreadPool();
```

```
        Future<Integer> future = executorService.submit(new
Callable<Integer>() {
```

```
            public Integer call()
```

```
            {
```

```
                Random random = new Random();
```

```

        int duration = random.nextInt(500);
        System.out.println("Starting...");
        try {
            Thread.sleep(duration);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Finished");
        return duration;
    }
});
executorService.shutdown();
try {
    System.out.println("result is : "+future.get());

    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (ExecutionException e) {
        e.printStackTrace();
    }
    System.out.println("future object done? :"+future.isDone());
    System.out.println("future object cancelled? :"+future.isCancelled());
}
}

```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -java
Starting...
Finished
result is : 237
future object done? :true
future object cancelled? :false

Process finished with exit code 0

```

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

Sol - Program File-Ques7.java

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

class Task implements Runnable
{
    @Override
    public void run() {
        Random random = new Random();
        Long duration = random.nextLong();

        try {
            System.out.println("Running Thread Name :
"+Thread.currentThread().getName());
            TimeUnit.SECONDS.sleep(5);
            System.out.println("Completed Thread Name :
"+Thread.currentThread().getName());
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Ques7 {
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newCachedThreadPool();
        executorService.submit(new Task());
    }
}
```

```

        executorService.submit(new Task());
        executorService.submit(new Task());
        executorService.submit(new Task());
        executorService.shutdown();
        try {
            executorService.awaitTermination(10000, TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Is Executor shutdown :
"+executorService.isShutdown());

    }
}

```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -javaagent
Running Thread Name : pool-1-thread-1
Running Thread Name : pool-1-thread-2
Running Thread Name : pool-1-thread-3
Running Thread Name : pool-1-thread-4
Completed Thread Name : pool-1-thread-1
Completed Thread Name : pool-1-thread-2
Completed Thread Name : pool-1-thread-3
Completed Thread Name : pool-1-thread-4
Is Executor shutdown : true

Process finished with exit code 0

```

8. Schedule task using `schedule()`, `scheduleAtFixedRate()` and `scheduleWithFixedDelay()`

Sol - Program File-Ques8.java

```

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

```

```

public class Ques8 {
    public static void main(String[] args) {
        ScheduledExecutorService ser = Executors.newScheduledThreadPool(4);
        ser.schedule(new RunClass(), 2, TimeUnit.SECONDS);
        ser.scheduleAtFixedRate(new RunClass(), 2, 5, TimeUnit.SECONDS);
        ser.scheduleWithFixedDelay(new RunClass(), 2, 3, TimeUnit.SECONDS);
    }
}

class RunClass implements Runnable
{
    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName()+ " Reading "+
        System.currentTimeMillis() + " milliseconds");
    }
}

```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -javaagent:/snap/intell
pool-1-thread-2 Reading 1614059286540 milliseconds
pool-1-thread-1 Reading 1614059286537 milliseconds
pool-1-thread-3 Reading 1614059286547 milliseconds
pool-1-thread-3 Reading 1614059289569 milliseconds
pool-1-thread-4 Reading 1614059291539 milliseconds
pool-1-thread-1 Reading 1614059292569 milliseconds
pool-1-thread-1 Reading 1614059295570 milliseconds
pool-1-thread-4 Reading 1614059296538 milliseconds
pool-1-thread-2 Reading 1614059298570 milliseconds
pool-1-thread-1 Reading 1614059301538 milliseconds
pool-1-thread-3 Reading 1614059301570 milliseconds
pool-1-thread-3 Reading 1614059304571 milliseconds
pool-1-thread-1 Reading 1614059306538 milliseconds
pool-1-thread-4 Reading 1614059307571 milliseconds
pool-1-thread-4 Reading 1614059310572 milliseconds
pool-1-thread-4 Reading 1614059311538 milliseconds

```

**9. Increase concurrency with Thread pools using
newCachedThreadPool() and newFixedThreadPool().**

Sol - Program File-Ques9.java

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

class CountdownClock extends Thread {
    private String clockName;

    public CountdownClock(String clockName) {
        this.clockName = clockName;
    }

    public void run() {
        String threadName = Thread.currentThread().getName();

        for (int i = 2; i >= 0; i--) {

            System.out.printf("%s -> %s: %d\n", threadName, clockName, i);

            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```



```
public class Ques9 {  
    public static void main(String[] args) {  
  
        ExecutorService pool = Executors.newCachedThreadPool();  
  
        pool.execute(new CountDownClock("A"));  
        pool.execute(new CountDownClock("B"));  
        pool.execute(new CountDownClock("C"));  
        pool.execute(new CountDownClock("D"));  
  
        pool.shutdown();  
  
        ExecutorService pool1 = Executors.newFixedThreadPool(2);  
        pool1.execute(new CountDownClock("A"));  
        pool1.execute(new CountDownClock("B"));  
        pool1.execute(new CountDownClock("C"));  
        pool1.execute(new CountDownClock("D"));  
  
        pool1.shutdown();  
    }  
}
```

```
/usr/lib/jvm/jdk-15.0.2/bin/java -javaagent
pool-1-thread-1 -> A: 2
pool-1-thread-2 -> B: 2
pool-2-thread-2 -> B: 2
pool-2-thread-1 -> A: 2
pool-1-thread-4 -> D: 2
pool-1-thread-3 -> C: 2
pool-1-thread-1 -> A: 1
pool-1-thread-2 -> B: 1
pool-2-thread-2 -> B: 1
pool-2-thread-1 -> A: 1
pool-1-thread-4 -> D: 1
pool-1-thread-3 -> C: 1
pool-1-thread-1 -> A: 0
pool-1-thread-2 -> B: 0
pool-2-thread-1 -> A: 0
pool-2-thread-2 -> B: 0
pool-1-thread-4 -> D: 0
pool-1-thread-3 -> C: 0
pool-2-thread-1 -> C: 2
pool-2-thread-2 -> D: 2
pool-2-thread-1 -> C: 1
pool-2-thread-2 -> D: 1
pool-2-thread-1 -> C: 0
pool-2-thread-2 -> D: 0
Process finished with exit code 0
```

newCachedThreadPool()

the number of threads is as equal as the number of submitted tasks. That's the key behavior of a cached thread pool: new threads are created as needed.

newFixedThreadPool()

The clocks A and B run first, while the clocks C and D are waiting in the queue. After A and B completes execution, the 2 threads continue executing the clocks C and D. That's the key behavior of a fixed thread pool: limiting the number of concurrent threads and queuing additional tasks.

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

Sol - Program File-Ques10.java

```
class BusStand
{
    synchronized public void getLine()
    {
        for (int i = 0; i < 3; i++)
        {
            System.out.println(i);
            try
            {
                Thread.sleep(400);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

```
class Bus extends Thread
{
    BusStand bs;
    Bus(BusStand bs)
    {
        this.bs = bs;
    }
}
```

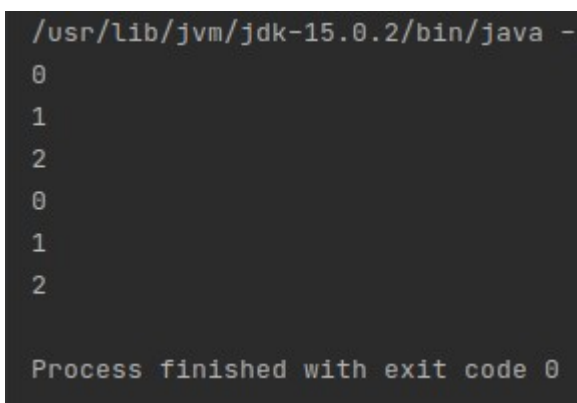
```

    }
    @Override
    public void run()
    {
        bs.getLine();
    }
}

public class Ques10 {
    public static void main(String[] args)
    {
        BusStand obj = new BusStand();

        Bus bus1 = new Bus(obj);
        Bus bus2 = new Bus(obj);
        bus1.start();
        bus2.start();
    }
}

```



```

/usr/lib/jvm/jdk-15.0.2/bin/java -
0
1
2
0
1
2

Process finished with exit code 0

```

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

Sol - Program File-Ques11.java

```
import java.util.Scanner;
```

```

class Account
{
    private int bal;
    public Account(int bal)
    {
        this.bal=bal;
    }
    public boolean isSufficientBalance(int w)
    {
        if (bal>w)
            return true;
        else
            return false;
    }
    public void withdraw(int amt)
    {
        bal = bal-amt;
        System.out.println("Withdrawl amount is "+amt);
        System.out.println("Your current balance is "+bal);
    }
}

class Customer implements Runnable
{
    private Account account;
    private String name;
    public Customer(Account account, String name)
    {
        this.account= account;
        this.name=name;
    }
}

```

```

public void run() {
    Scanner s = new Scanner(System.in);
    synchronized (account) {
        System.out.println(name + ", Enter amount to withdraw");
        int amt = s.nextInt();

        if (account.isSufficientBalance(amt)) {
            System.out.println(name);
            account.withdraw(amt);
        } else
            System.out.println("Insufficient Balance");
    }
}
}

public class Ques11 {
    public static void main(String[] args) {
        Account a1 = new Account(1000);
        Customer c1 = new Customer(a1, "Ram");
        Customer c2 = new Customer(a1, "Naveen");
        Thread t1 = new Thread(c1);
        Thread t2 = new Thread(c2);
        t1.start();
        t2.start();
    }
}

```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -jav
Ram, Enter amount to withdraw
600
Ram
Withdrawl amount is 600
Your current balance is 400
Naveen, Enter amount to withdraw
700
Insufficient Balance

Process finished with exit code 0

```

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

Sol - Program File-Ques12.java

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

public class Ques12 {
    public static void main(String[] args) {
        new AtomThread("A");
        new AtomThread("B");
        new AtomThread("C");
    }
}

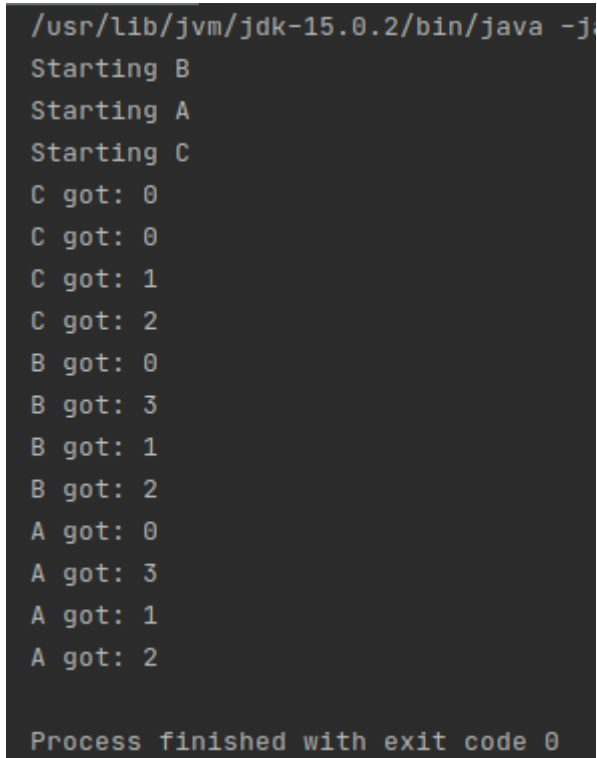
class Shared
{
    static AtomicInteger ai = new AtomicInteger(0);
}

class AtomThread implements Runnable{
    String name;
    AtomThread(String n)
    {
        name = n;
        new Thread(this).start();
    }
    public void run(){
        System.out.println("Starting "+ name);
        for(int i=0; i<=3; i++)
```

```

        System.out.println(name + " got: " + Shared.ai.getAndSet(i));
    }
}

```



```

/usr/lib/jvm/jdk-15.0.2/bin/java -j
Starting B
Starting A
Starting C
C got: 0
C got: 0
C got: 1
C got: 2
B got: 0
B got: 3
B got: 1
B got: 2
A got: 0
A got: 3
A got: 1
A got: 2

Process finished with exit code 0

```

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

Sol - Program File-Ques13.java

```

class C1 extends Thread {
    public void run()
    {
        synchronized(this)
        {
            System.out.println
                (Thread.currentThread().getName() + " starts");
            try {
                this.wait();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```



```

        }
        System.out.println
            (Thread.currentThread().getName() + " notified");
    }
}
} class C2 extends Thread {
    C1 c1;
    C2(C1 c1)
    {
        this.c1 = c1;
    }
    public void run()
    {
        synchronized(this.c1)
        {
            System.out.println
                (Thread.currentThread().getName() + " starts");

            try {
                this.c1.wait();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println
                (Thread.currentThread().getName() + " notified");
        }
    }
} class C3 extends Thread {
    C1 c1;
    C3(C1 c1)

```

```

{
    this.c1 = c1;
}
public void run()
{
    synchronized(this.c1)
    {
        System.out.println
            (Thread.currentThread().getName() + " starts");
        this.c1.notify();
        System.out.println
            (Thread.currentThread().getName() + " notified");
    }
}
} class Ques13 {
    public static void main(String[] args) throws InterruptedException
    {

        C1 c1 = new C1();
        C2 c2 = new C2(c1);
        C3 c3 = new C3(c1);
        Thread t1 = new Thread(c1, "Thread-1");
        Thread t2 = new Thread(c2, "Thread-2");
        Thread t3 = new Thread(c3, "Thread-3");
        t1.start();
        t2.start();
        Thread.sleep(100);

```

```

t3.start();

```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -javaagent
Thread-1 starts
Thread-2 starts
Thread-3 starts
Thread-3 notified
Thread-1 notified

Process finished with exit code 130 (interrupted)

```

```

}

```

```

}

```

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

Sol - Program File-Ques14.java

```
package com.Ques14;
```

```
class C1 extends Thread {
    public void run()
    {
        synchronized(this)
        {
            System.out.println
                (Thread.currentThread().getName() + " starts");
            try {
                this.wait();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println
                (Thread.currentThread().getName() + " notified");
        }
    }
}
class C2 extends Thread {
    C1 c1;
    C2(C1 c1)
    {
        this.c1 = c1;
    }
    public void run()
```

```

{
    synchronized(this.c1)
    {
        System.out.println
            (Thread.currentThread().getName() + " starts");

        try {
            this.c1.wait();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println
            (Thread.currentThread().getName() + " notified");
    }
}
} class C3 extends Thread {
    C1 c1;
    C3(C1 c1)
    {
        this.c1 = c1;
    }
    public void run()
    {
        synchronized(this.c1)
        {
            System.out.println
                (Thread.currentThread().getName() + " starts");
            this.c1.notifyAll();
            System.out.println
                (Thread.currentThread().getName() + " notified");
        }
    }
}

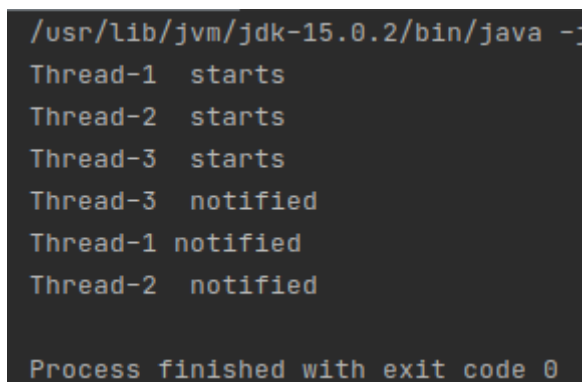
```

```

    }
}
} class Ques14 {
    public static void main(String[] args) throws InterruptedException
    {

        C1 c1 = new C1();
        C2 c2 = new C2(c1);
        C3 c3 = new C3(c1);
        Thread t1 = new Thread(c1, "Thread-1");
        Thread t2 = new Thread(c2, "Thread-2");
        Thread t3 = new Thread(c3, "Thread-3");
        t1.start();
        t2.start();
        Thread.sleep(100);
        t3.start();
    }
}

```



```

/usr/lib/jvm/jdk-15.0.2/bin/java -j
Thread-1  starts
Thread-2  starts
Thread-3  starts
Thread-3  notified
Thread-1  notified
Thread-2  notified

Process finished with exit code 0

```

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

Sol - Program File-Ques15.java

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

```

public class Ques15 {
    public static void main(String[] args) {
        ReentrantLock lock = new ReentrantLock();
        new LockThread(lock, "A");
        new LockThread(lock, "B");
    }
}

class SharedClass{
    static int count = 0;
}

class LockThread implements Runnable{
    String name;
    ReentrantLock lock;
    LockThread(ReentrantLock lk, String n){
        lock = lk;
        name = n;
        new Thread(this).start();
    }
    public void run(){
        System.out.println("Starting "+ name);
        try{
            System.out.println(name+" is waiting to lock count");
            lock.lock();
            System.out.println(name+ " is locking count");
            SharedClass.count++;
            System.out.println(name + ": "+ SharedClass.count);
            System.out.println(name + " is sleeing");
            Thread.sleep(1000);
        }catch (InterruptedException e){

```

```

        System.out.println(e);
    }finally {
        System.out.println(name + " is unlocking count");
        lock.unlock();
    }
}
}

```

```

/usr/lib/jvm/jdk-15.0.2/bin/java -javaagent
Starting A
A is waiting to lock count
A is locking count
Starting B
B is waiting to lock count
A: 1
A is sleeing
A is unlocking count
B is locking count
B: 2
B is sleeing
B is unlocking count

Process finished with exit code 0

```

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

Sol - Program File-Ques16.java

```
package com.Ques16.Deadlock;
```

```

import java.util.Random;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
class Account{
    private int balance=10000;
    public void deposit(int amount){

```

```

        balance += amount;
    }
    public void withdraw(int amount){
        balance -= amount;
    }
    public int getBalance(){
        return balance;
    }
    public static void transfer(Account acc1, Account acc2, int amount){
        acc1.withdraw(amount);
        acc2.deposit(amount);
    }
}

class Runner{
    private Account acc1= new Account();
    private Account acc2= new Account();

    private Lock lock1 = new ReentrantLock();
    private Lock lock2 = new ReentrantLock();

    private void acquireLocks(Lock firstLock, Lock secondLock) throws
    InterruptedException{
        while(true){
            //Acquire locks
            boolean gotFirstLock = false;
            boolean gotSecondLock = false;
            try {
                gotFirstLock = firstLock.tryLock();
                gotSecondLock = secondLock.tryLock();
            }
            finally {

```



```

        if(gotFirstLock && gotSecondLock){
            return;
        }
        if(gotFirstLock){
            firstLock.unlock();
        }
        if(gotSecondLock){
            secondLock.unlock();
        }
    }
    //Locks not acquired
    Thread.sleep(1);
}
}

```

```

public void firstThread() throws InterruptedException{
    Random random = new Random();
    for(int i=0;i<1000; i++) {
        acquireLocks(lock1, lock2);
        try {
            Account.transfer(acc1, acc2, random.nextInt(100));
        }finally {
            lock1.unlock();
            lock2.unlock();
        }
    }
}

public void secondThread() throws InterruptedException{
    Random random = new Random();
    for(int i=0;i<1000; i++){
        acquireLocks(lock2, lock1);
    }
}

```

```

        try {
            Account.transfer(acc2, acc1, random.nextInt(100));
        } finally {
            lock1.unlock();
            lock2.unlock();
        }
    }
}

public void finished(){
    System.out.println("Account 1 balance: "+ acc1.getBalance());
    System.out.println("Account 2 balance: "+ acc2.getBalance());
    System.out.println("Total balance: "+ (acc1.getBalance() +
acc2.getBalance()));
}
}

```

```

public class Ques16 {
    public static void main(String[] args) throws Exception {
        final Runner runner = new Runner();
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    runner.firstThread();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
    }
}

```

```

Thread t2 = new Thread(new Runnable() {

```

```
@Override
public void run() {
    try {
        runner.secondThread();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
});
```

```
t1.start();
t2.start();
t1.join();
t2.join();
runner.finished();
}
}
```

```
/usr/lib/jvm/jdk-15.0.2/bin/java -ja
Account 1 balance: 10787
Account 2 balance: 9213
Total balance: 20000

Process finished with exit code 0
|
```