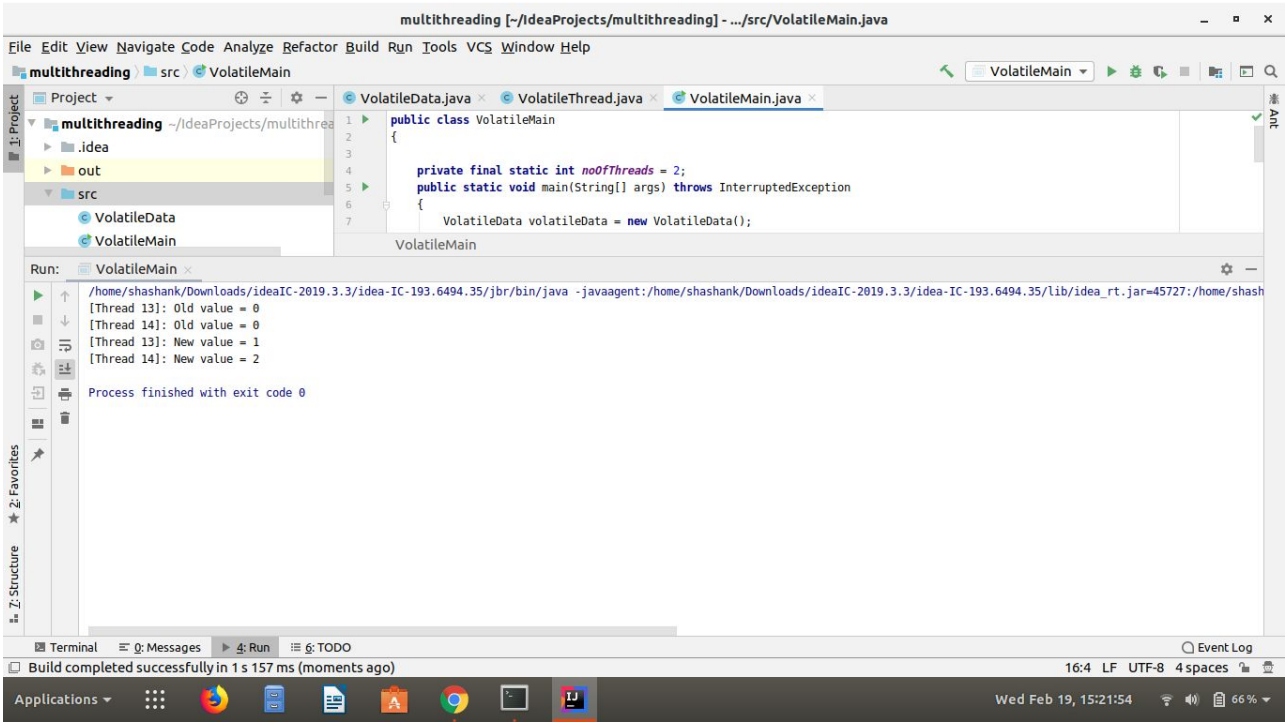
JAVA EXERCISE 3 (MULTITHREADING)

1. Write a programe do to demonstrate the use of volatile keyword.

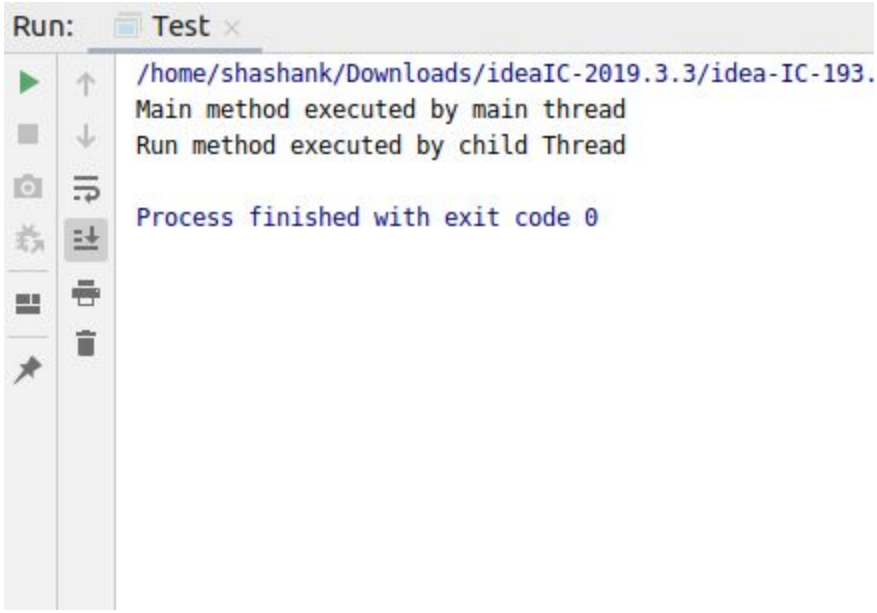
```
public class VolatileData
{
  private volatile int counter = 0;
  public int getCounter()
  {
     return counter;
  }
  public void increaseCounter()
  {
     ++counter;
  }
}
public class VolatileThread extends Thread
{
  private final VolatileData data;
  public VolatileThread(VolatileData data)
  {
     this.data = data;
  }
  public void run()
  {
     int oldValue = data.getCounter();
     System.out.println("[Thread " + Thread.currentThread().getId() + "]: Old value = " + oldValue);
     data.increaseCounter();
     int newValue = data.getCounter();
     System.out.println("[Thread " + Thread.currentThread().getId() + "]: New value = " + newValue);
  }
}
```

```java
public class VolatileMain
{

  private final static int noOfThreads = 2;
  public static void main(String[] args) throws InterruptedException
  {
    VolatileData volatileData = new VolatileData();
    Thread[] threads = new Thread[noOfThreads];
    for(int i = 0; i < noOfThreads; ++i)
      threads[i] = new VolatileThread(volatileData);
    for(int i = 0; i < noOfThreads; ++i)
      threads[i].start();
    for(int i = 0; i < noOfThreads; ++i)
      threads[i].join();
  }
}
```



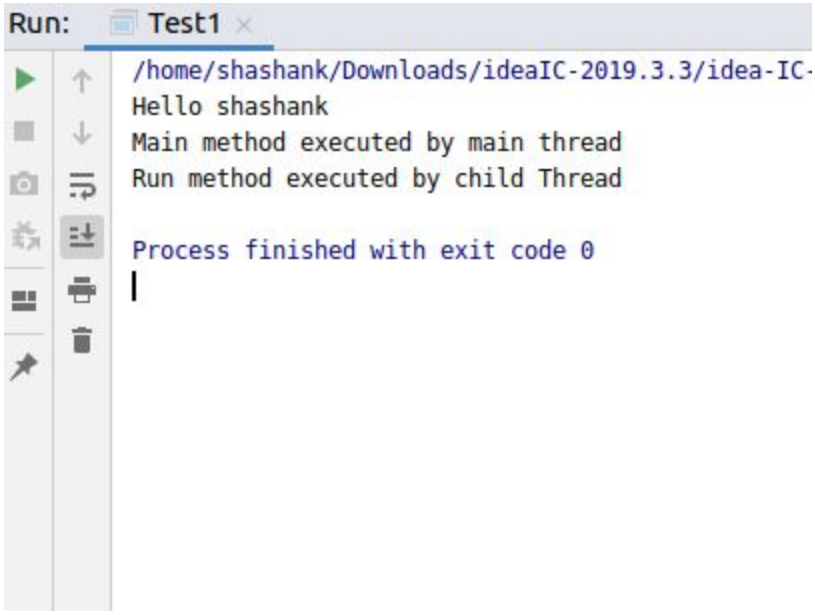
2. Write a program to create a thread using Thread class and Runnable interface each.

```
class Test extends Thread
{
  public void run()
  {
    System.out.println("Run method executed by child Thread");
  }
  public static void main(String[] args)
  {
    Test t = new Test();
    t.start();
    System.out.println("Main method executed by main thread");
  }
}
```




```
class test2{
  public static void m1()
  {
    System.out.println("Hello shashank");
```

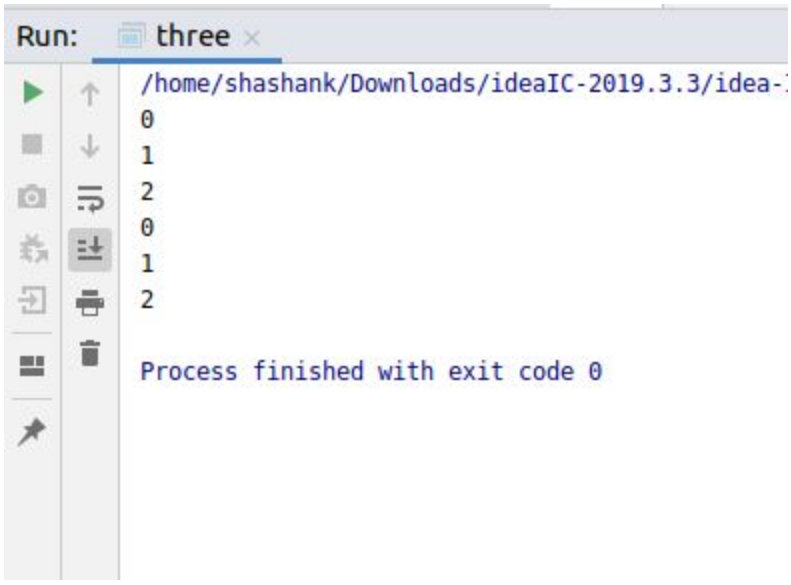
```
    }
}
class Test1 extends test2 implements Runnable {
    public void run() {
        System.out.println("Run method executed by child Thread");
    }

    public static void main(String[] args) {
        Test1 t = new Test1();
        t.m1();
        Thread t1 = new Thread(t);
        t1.start();
        System.out.println("Main method executed by main thread");
    }
}
```



3. Write a program using synchronization block and synchronization method

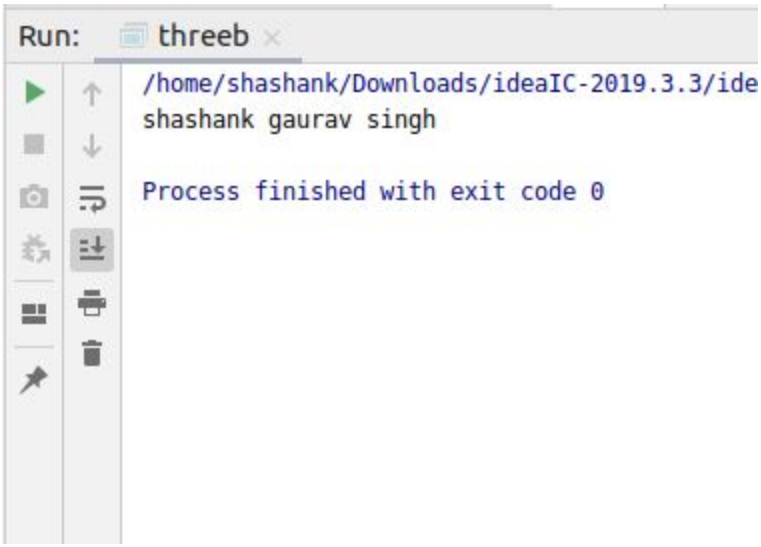
```
import java.io.*;

class Line

{ synchronized public void getLine()

  {

     for (int i = 0; i < 3; i++)

     {

        System.out.println(i);

        try

        {

           Thread.sleep(400);

        }

        catch (Exception e)

        {

           System.out.println(e);

        }

     }

  }

}


class Train extends Thread

{ Line line;


  Train(Line line)

  {

     this.line = line;

  }


  public void run()
```

```
  {
      line.getLine();
  }
}

public class three
{
  public static void main(String[] args)
  {
      Line obj = new Line();
      Train t1 = new Train(obj);
      Train t2 = new Train(obj);
      t1.start();
      t2.start();
  }
}
```




```
import java.io.*;
import java.util.*;
class bsyn
{
```

```
  String name = "";
  public int c = 0;
  public void name(String n, List<String> list)
  { synchronized(this)
     {
        name =n;
        c++;
     }
     list.add(n);
  }
}
class threeb
{
  public static void main (String[] args)
  {
     bsyn n1 = new bsyn();
     List<String> list = new ArrayList<String>();
     n1.name("shashank gaurav singh", list);
     System.out.println(n1.name);

  }
}
```



4. **Write a program to create a Thread pool of 2 threads where one Thread will print even numbers and other will print odd numbers.**

```
public class mfour{
  boolean odd;
  int count = 1;
  int MAX = 6;
  public void printOdd() {
    synchronized (this) {
      while (count < MAX) {
        System.out.println("Checking odd loop");

        while (!odd) {
          try {
            System.out.println("Odd waiting : " + count);
            wait();
            System.out.println("Notified odd :" + count);
          } catch (InterruptedException e) {
            e.printStackTrace();
          }
```

```
            }
            System.out.println("Odd Thread :" + count);
            count++;
            odd = false;
            notify();
        }
    }
}

public void printEven() {

    try {
        Thread.sleep(1000);
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    }
    synchronized (this) {
        while (count < MAX) {
            System.out.println("Checking even loop");

            while (odd) {
                try {
                    System.out.println("Even waiting: " + count);
                    wait();
                    System.out.println("Notified even:" + count);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("Even thread :" + count);
            count++;
            odd = true;
```

```
            notify();

        }
    }
}

public static void main(String[] args) {

    mfour n = new mfour();
    n.odd = true;
    Thread t1 = new Thread(new Runnable() {
        public void run() {
            n.printEven();
        }
    });
    Thread t2 = new Thread(new Runnable() {
        public void run() {
            n.printOdd();

        }
    });

    t1.start();
    t2.start();
}
}
```

```
Run:  mfour
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-1
Checking odd loop
Odd Thread :1
Checking odd loop
Odd waiting : 2
Checking even loop
Even thread :2
Checking even loop
Even waiting: 3
Notified odd :3
Odd Thread :3
Checking odd loop
Odd waiting : 4
Notified even:4
Even thread :4
Checking even loop
Even waiting: 5
Notified odd :5
Odd Thread :5
Notified even:6
Even thread :6

Process finished with exit code 0
```

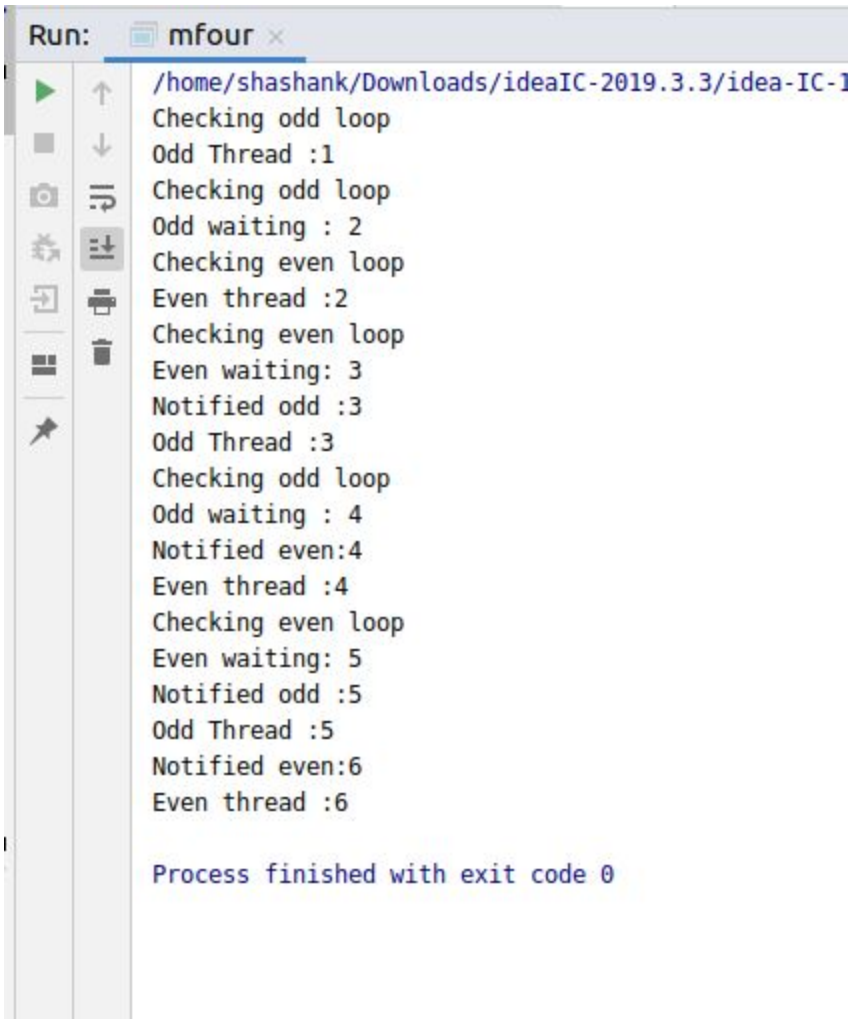
5. Write a program to demonstrate wait and notify methods.

```
public class mfour{
  boolean odd;
  int count = 1;
  int MAX = 6;
  public void printOdd() {
    synchronized (this) {
      while (count < MAX) {
        System.out.println("Checking odd loop");
```

```
            while (!odd) {
                try {
                    System.out.println("Odd waiting : " + count);
                    wait();
                    System.out.println("Notified odd :" + count);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("Odd Thread :" + count);
            count++;
            odd = false;
            notify();
        }
    }
}

public void printEven() {

    try {
        Thread.sleep(1000);
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    }
    synchronized (this) {
        while (count < MAX) {
            System.out.println("Checking even loop");

            while (odd) {
                try {
                    System.out.println("Even waiting: " + count);
                    wait();
```

```
                System.out.println("Notified even:" + count);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Even thread :" + count);
        count++;
        odd = true;
        notify();

    }
  }
}

public static void main(String[] args) {

  mfour n = new mfour();
  n.odd = true;
  Thread t1 = new Thread(new Runnable() {
    public void run() {
      n.printEven();
    }
  });
  Thread t2 = new Thread(new Runnable() {
    public void run() {
      n.printOdd();

    }
  });

  t1.start();
  t2.start();
```

```
    }
  }
```

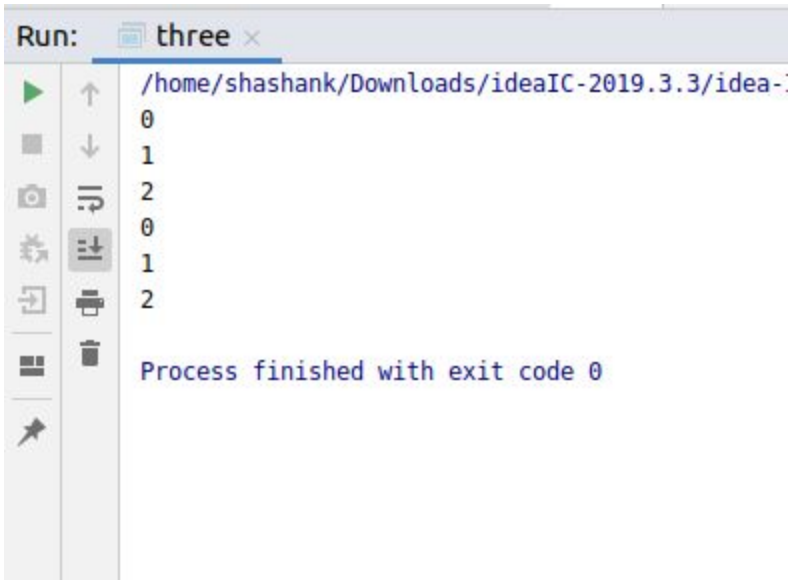


6. Write a program to demonstrate sleep and join methods.

```
import java.io.*;

class Line

{ synchronized public void getLine()

  {

    for (int i = 0; i < 3; i++)

    {

      System.out.println(i);

      try

      {
```

```
            Thread.sleep(400);
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
  }
}

class Train extends Thread
{ Line line;

  Train(Line line)
  {
      this.line = line;
  }

  public void run()
  {
      line.getLine();
  }
}

public class three
{
  public static void main(String[] args)
  {
      Line obj = new Line();
      Train t1 = new Train(obj);
      Train t2 = new Train(obj);
      t1.start();
```

```
        t2.start();
    }
}
```




```
class msix extends Thread{
    public void run(){
        for(int i=1;i<=5;i++){
            try{
                Thread.sleep(500);
            }catch(Exception e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[]){
        msix t1=new msix();
        msix t2=new msix();
        msix t3=new msix();

        t1.start();
        try{
            t1.join();
```

```
    }catch(Exception e){System.out.println(e);}


    t2.start();
    t3.start();
  }
}
```

```
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-193.6494.
1
2
3
4
5
1
1
2
2
3
3
4
4
5
5

Process finished with exit code 0
```

7. Run a task with the help of callable and store it's result in the Future.

```
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.concurrent.FutureTask;


class mseven1 implements Callable
{
```

```
    public Object call() throws Exception
    {
        Random generator = new Random();
        Integer randomNumber = generator.nextInt(5);

        Thread.sleep(randomNumber * 1000);

        return randomNumber;
    }

}

public class mseven
{
    public static void main(String[] args) throws Exception
    {
        FutureTask[] randomNumberTasks = new FutureTask[5];

        for (int i = 0; i < 5; i++)
        {
            Callable callable = new mseven1();

            randomNumberTasks[i] = new FutureTask(callable);
            Thread t = new Thread(randomNumberTasks[i]);
            t.start();
        }

        for (int i = 0; i < 5; i++)
        {
            System.out.println(randomNumberTasks[i].get());
```

```
    }
  }
}
```

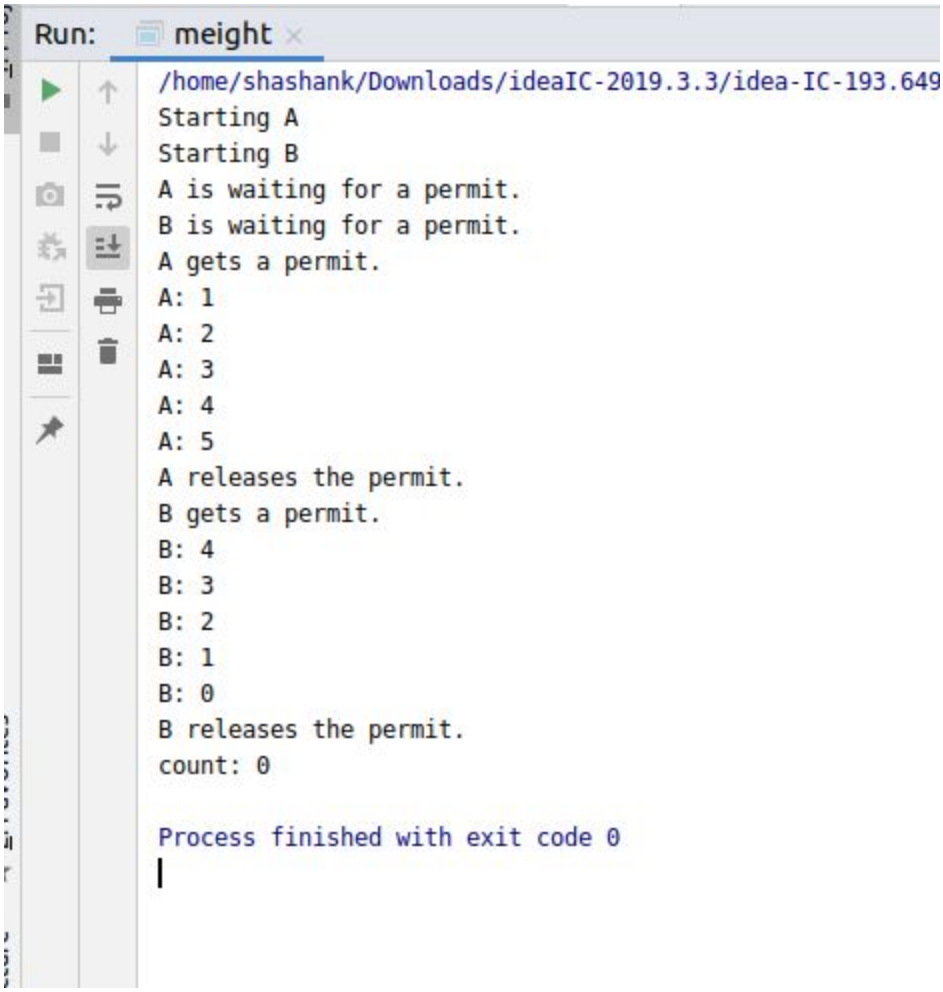
8. Write a program to demonstrate the use of semaphore

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

class Shared

{

  static int count = 0;

}


class MyThread extends Thread

{

  Semaphore sem;

  String threadName;

  public MyThread(Semaphore sem, String threadName)

  {

    super(threadName);

    this.sem = sem;
```

```
        this.threadName = threadName;
    }
    public void run() {
        if(this.getName().equals("A"))
        {
            System.out.println("Starting " + threadName);
            try
            {
                System.out.println(threadName + " is waiting for a permit.");
                sem.acquire();

                System.out.println(threadName + " gets a permit.");
                for(int i=0; i < 5; i++)
                {
                    Shared.count++;
                    System.out.println(threadName + ": " + Shared.count);
                    Thread.sleep(10);
                }
            } catch (InterruptedException exc) {
                System.out.println(exc);
            }

            System.out.println(threadName + " releases the permit.");
            sem.release();
        }

        else
        {
            System.out.println("Starting " + threadName);
            try
            {
                System.out.println(threadName + " is waiting for a permit.");
```

```
            sem.acquire();

            System.out.println(threadName + " gets a permit.");

            for(int i=0; i < 5; i++)
            {
                Shared.count--;
                System.out.println(threadName + ": " + Shared.count);

                Thread.sleep(10);
            }
        } catch (InterruptedException exc) {
            System.out.println(exc);
        }
        System.out.println(threadName + " releases the permit.");
        sem.release();
    }
  }
}

public class meight
{
  public static void main(String args[]) throws InterruptedException
  {
    Semaphore sem = new Semaphore(1);

    MyThread mt1 = new MyThread(sem, "A");
    MyThread mt2 = new MyThread(sem, "B");

    mt1.start();
    mt2.start();
```

```
        mt1.join();
        mt2.join();
        System.out.println("count: " + Shared.count);
    }
}
```

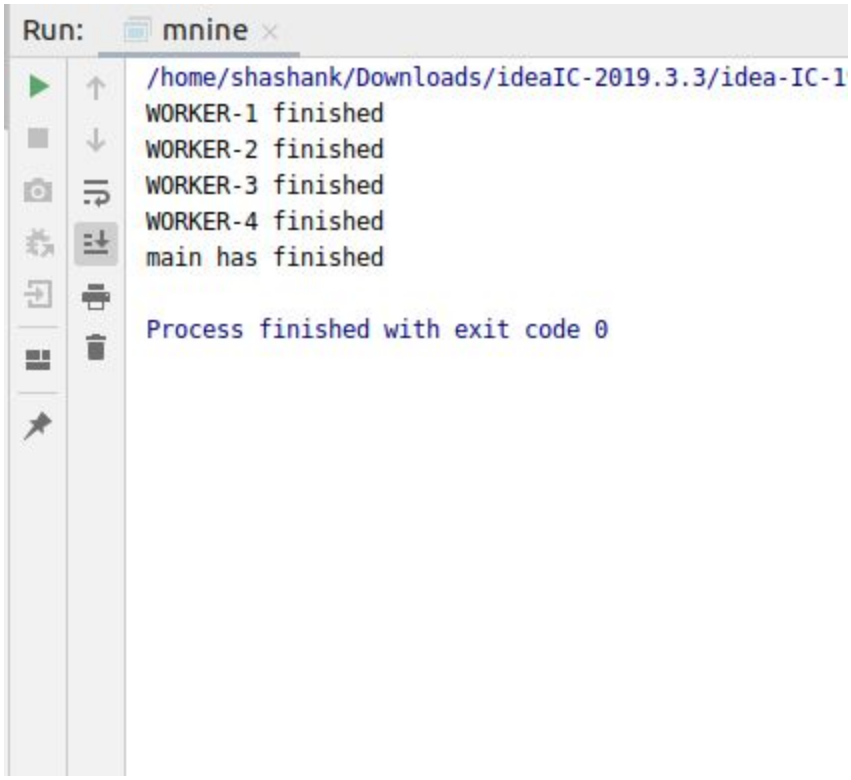


9. Write a program to demonstrate the use of CountDownLatch

```
import java.util.concurrent.CountDownLatch;


public class mnine
{
  public static void main(String args[])
        throws InterruptedException
```

```
  {
    CountDownLatch latch = new CountDownLatch(4);

    Worker first = new Worker(1000, latch,
        "WORKER-1");
    Worker second = new Worker(2000, latch,
        "WORKER-2");
    Worker third = new Worker(3000, latch,
        "WORKER-3");
    Worker fourth = new Worker(4000, latch,
        "WORKER-4");
    first.start();
    second.start();
    third.start();
    fourth.start();
    latch.await();

    System.out.println(Thread.currentThread().getName() +
        " has finished");
  }
}

class Worker extends Thread
{
  private int delay;
  private CountDownLatch latch;

  public Worker(int delay, CountDownLatch latch,
          String name)
  {
    super(name);
    this.delay = delay;
```

```
        this.latch = latch;
    }
    public void run()
    {
        try
        {
            Thread.sleep(delay);
            latch.countDown();
            System.out.println(Thread.currentThread().getName()
                    + " finished");
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```



10. Write a program which creates deadlock between 2 threads

```
public class mten {
  public static void main(String[] args) {
    final String resource1 = "shashank";
    final String resource2 = "singh";
    Thread t1 = new Thread() {
      public void run() {
        synchronized (resource1) {
          System.out.println("Thread 1: locked resource 1");

          try { Thread.sleep(100);} catch (Exception e) {}

          synchronized (resource2) {
            System.out.println("Thread 1: locked resource 2");
          }
        }
      }
    };

    Thread t2 = new Thread() {
      public void run() {
        synchronized (resource2) {
          System.out.println("Thread 2: locked resource 2");
```

```
                try { Thread.sleep(100);} catch (Exception e) {}

                synchronized (resource1) {
                    System.out.println("Thread 2: locked resource 1");
                }
            }
        }
    };


    t1.start();
    t2.start();
  }
}
```