Session : Basics of Multi Threading

1. **Write a program to demonstrate the use of volatile keyword.**

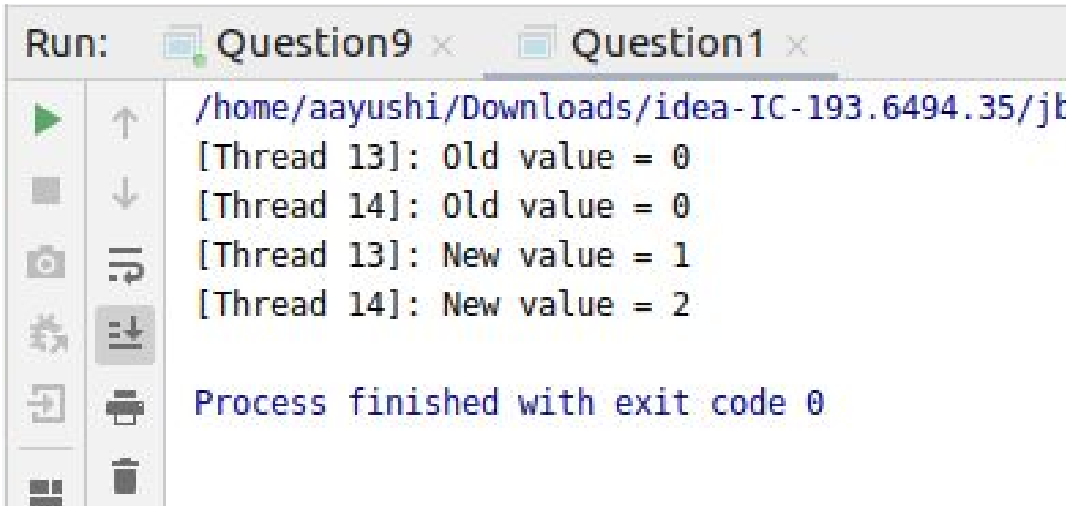
Suppose that two threads are working on SharedObj. If two threads run on different processors each thread may have its own local copy of sharedVariable. If one thread modifies its value the change might not reflect in the original one in the main memory instantly. This depends on the write policy of cache. Now the other thread is not aware of the modified value which leads to data inconsistency.
In Java, a global ordering is imposed on the read and write operations concerning a volatile variable. A thread that access a volatile field, will first read its current value from the main memory, instead of using a potential cached value.Therefore, any change to a volatile variable is always visible to other threads.

CODE :

```
package aayushi;

class VolatileData {

  private volatile int counter = 0;

  public int getCounter() {
     return counter;
  }

  public void increaseCounter() {
     ++counter;
  }
}

class VolatileThread extends Thread {
  private final VolatileData data;

  public VolatileThread(VolatileData data) {
     this.data = data;
  }

  @Override
  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);
    }
}

public class Question1 {
    private final static int TOTAL_THREADS = 2;
    public static void main(String[] args) throws InterruptedException {
        VolatileData volatileData = new VolatileData();

        Thread[] threads = new Thread[TOTAL_THREADS];
        for(int i = 0; i < TOTAL_THREADS; ++i)
            threads[i] = new VolatileThread(volatileData);

        //Start all reader threads.
        for(int i = 0; i < TOTAL_THREADS; ++i)
            threads[i].start();

        //Wait for all threads to terminate.
        for(int i = 0; i < TOTAL_THREADS; ++i)
            threads[i].join();
    }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jb
[Thread 13]: Old value = 0
[Thread 14]: Old value = 0
[Thread 13]: New value = 1
[Thread 14]: New value = 2

Process finished with exit code 0
```

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

we can define a thread in the following two ways:

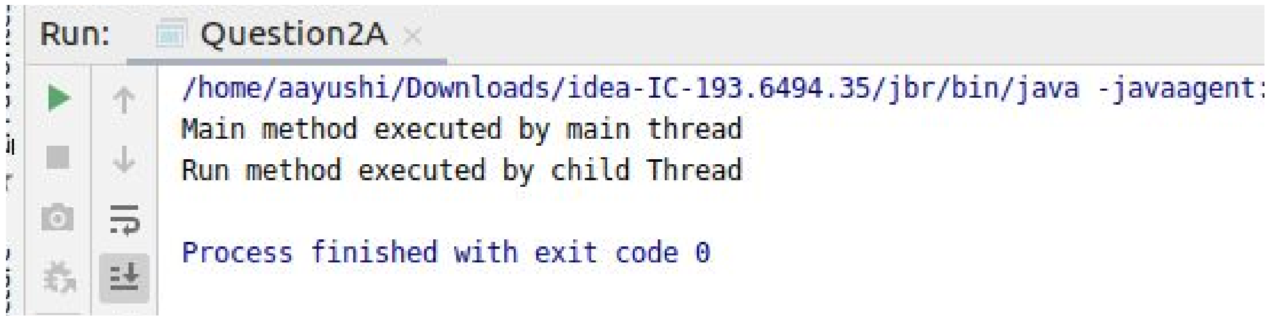
- By extending Thread class
- By implementing Runnable interface

In the first approach, Our class always extends the Thread class. There is no chance of extending any other class. Hence we are missing Inheritance benefits. In the second approach, while implementing Runnable interface we can extend any other class. Hence we are able to use the benefits of Inheritance.
Because of the above reasons, implementing a Runnable interface approach is recommended rather than extending Thread class.

CODE :THREAD CLASS

```
package aayushi;
//create a thread using Thread class
class TestA extends Thread {
  public void run() {
     System.out.println("Run method executed by child Thread");
  }
}

public class Question2A {
  public static void main(String[] args)
  {
     TestA t = new TestA();
     t.start();
     System.out.println("Main method executed by main thread");
  }
}
```

OUTPUT



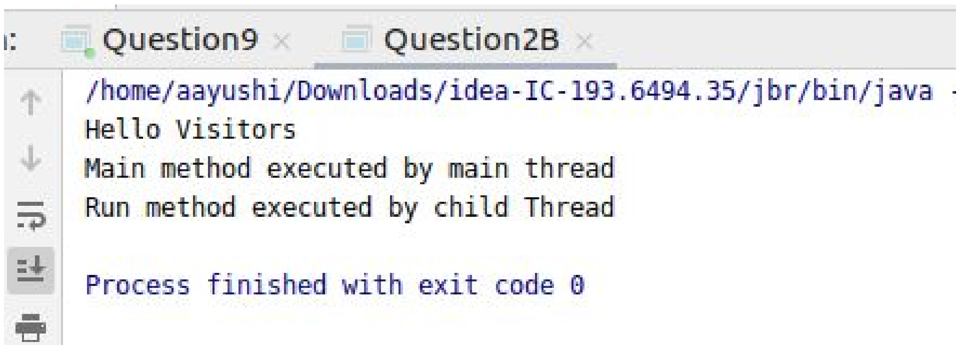
```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/java -javaagent:
Main method executed by main thread
Run method executed by child Thread

Process finished with exit code 0
```

CODE :RUNNABLE INTERFACE

```
package aayushi;
```

```
//create a thread using Runnable interface

class TestB implements Runnable {
  public void run()
  {
     System.out.println("Run method executed by child Thread");
  }
  public void m1()
  {
     System.out.println("Hello Visitors");
  }
}

public class Question2B {
  public static void main(String[] args) {
     TestB t = new TestB();
     t.m1();
     Thread t1 = new Thread(t);
     t1.start();
     System.out.println("Main method executed by main thread");
  }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/java
Hello Visitors
Main method executed by main thread
Run method executed by child Thread

Process finished with exit code 0
```

3. **Write a program using synchronization block and synchronization method.**

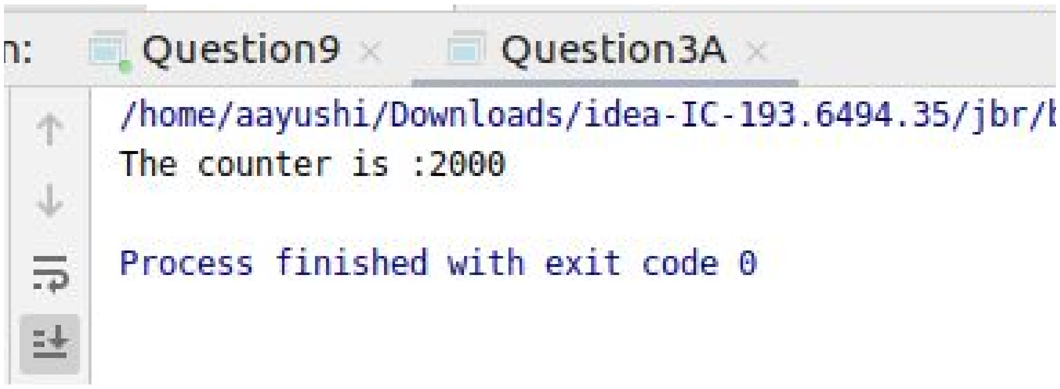
Java is a multithreaded language where multiple threads runs parallel to complete their execution. We need to synchronize the shared resources to ensure that at a time only one thread is able to access the shared resource.

A. CODE :Synchronization Method

```
package aayushi;
```

```
public class Question3A {
  private int count=0;
  public synchronized void counter()
  {
     count++;
  }

  public static void main(String[] args) throws InterruptedException {
     Question3A t1 = new Question3A();
     t1.dowork();
  }

  public void dowork() throws InterruptedException {
     Thread t1 = new Thread(new Runnable() {
        @Override
        public void run() {
           for(int i =0;i<1000;i++)
              counter();
        }
     });

     Thread t2 = new Thread(new Runnable() {
        @Override
        public void run() {
           for(int i =0;i<1000;i++)
              counter();
        }
     });
     t1.start();
     t2.start();

     try{
        t1.join();
        t2.join();
     }

     catch(InterruptedException ex){
        ex.printStackTrace();
     }
     System.out.println("The counter is :"+count);
  }
}
```

OUTPUT

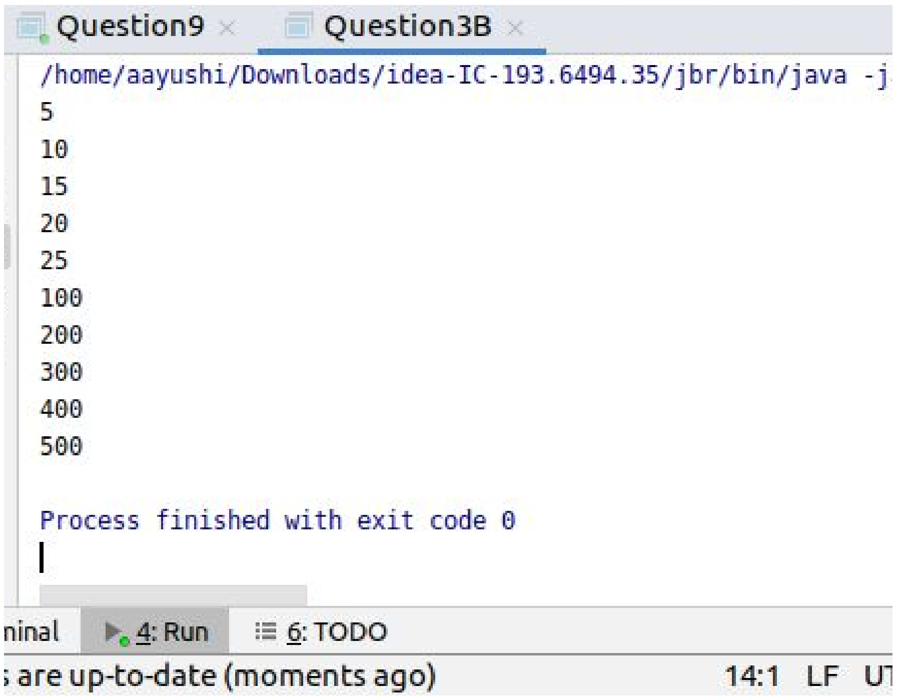


```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/b
The counter is :2000

Process finished with exit code 0
```

B. CODE :Synchronization Block

```
package aayushi;

class Table{
  void printTable(int n){
     synchronized(this){//synchronized block
        for(int i=1;i<=5;i++){
           System.out.println(n*i);
           try{
              Thread.sleep(400);
           }catch(Exception e){System.out.println(e);}
        }
     }
  }//end of the method
}

class MyThread1 extends Thread{
  Table t;
  MyThread1(Table t){
     this.t=t;
  }
  public void run(){
     t.printTable(5);
  }
}

class MyThread2 extends Thread{
  Table t;
  MyThread2(Table t){
     this.t=t;
  }
  public void run(){
     t.printTable(100);
  }
}

public class Question3B{
  public static void main(String args[]){
     Table obj = new Table();//only one object
     MyThread1 t1=new MyThread1(obj);
     MyThread2 t2=new MyThread2(obj);
     t1.start();
     t2.start();
  }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/java -j
5
10
15
20
25
100
200
300
400
500

Process finished with exit code 0
```

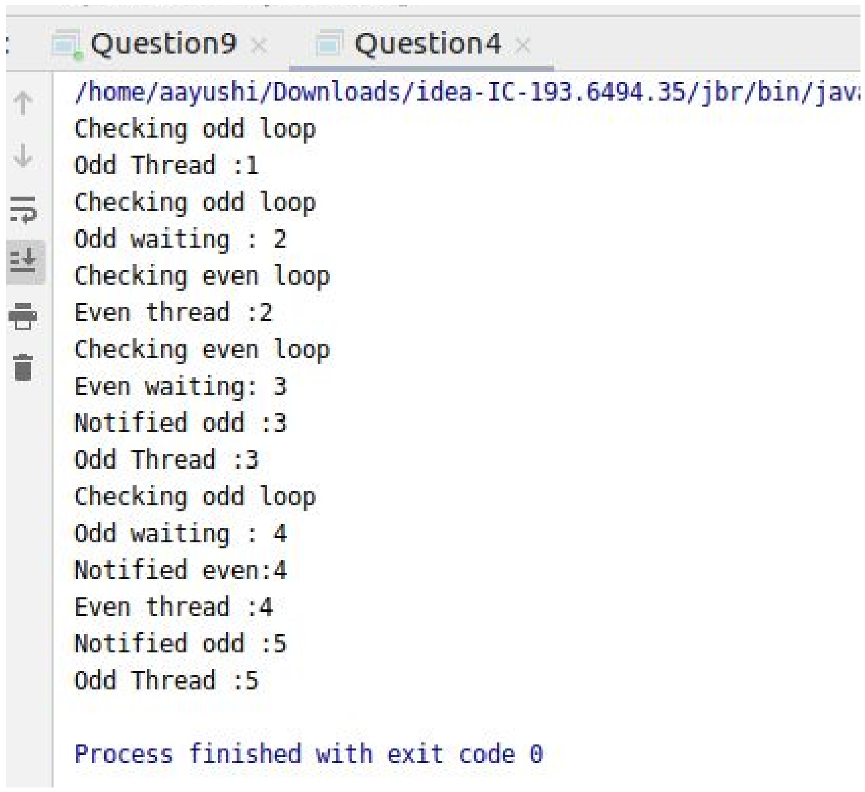
4. **Write a program to create a Thread pool of 2 threads where one Thread will print even numbers and other will print odd numbers.**

CODE :

```
package aayushi;
public class Question4 {
  boolean odd;
```

```
int count = 1;
int MAX = 5;

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) {

    Question4 obj = new Question4();
    obj.odd = true;
    Thread t1 = new Thread(new Runnable() {
        @Override
        public void run() {
            obj.printEven();
        }
    });
    Thread t2 = new Thread(new Runnable() {
        @Override
        public void run() {
            obj.printOdd();
        }
    });

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

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

}
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/jav
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
Notified odd :5
Odd Thread :5

Process finished with exit code 0
```

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

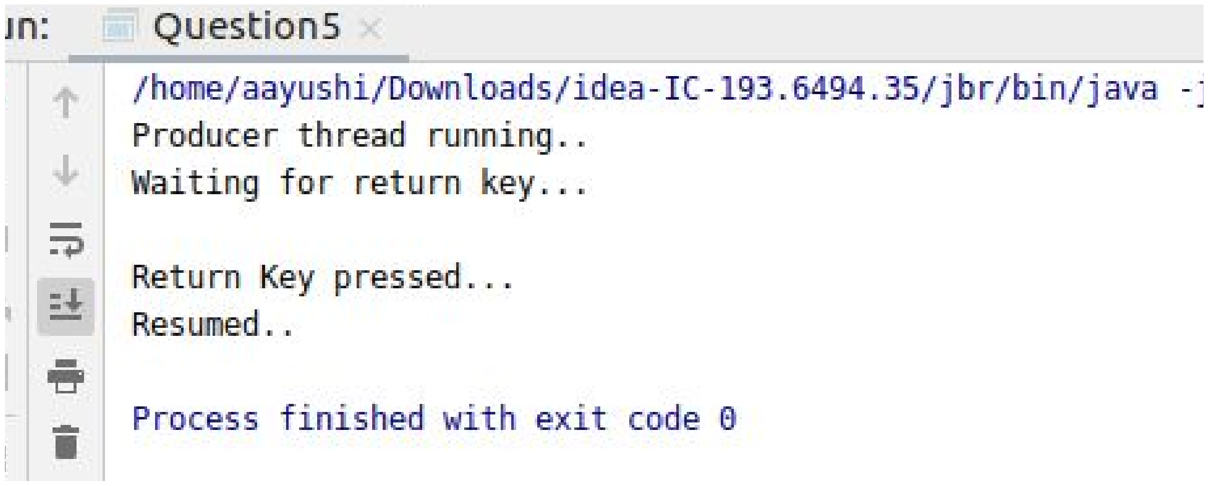
The process of testing a condition repeatedly till it becomes true is known as polling.
Polling is usually implemented with the help of loops to check whether a particular condition is true or not. If it is true, certain action is taken. This waste many CPU cycles and makes the implementation inefficient.
For example, in a classic queuing problem where one thread is producing data and other is consuming it.
wait()-It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify().
notify()-It wakes up one single thread that called wait() on the same object. It should be noted that calling notify() does not actually give up a lock on a resource.

CODE :

```
package aayushi;

import java.util.Scanner;

public class Question5{
  public void produce() throws InterruptedException
```

```
{
    synchronized (this)
    {
        System.out.println("Producer thread running..");
        wait();
        System.out.println("Resumed..");
    }
}

public void consumer() throws InterruptedException
{
    Scanner sc = new Scanner(System.in);
    Thread.sleep(2000);

    synchronized (this)
    {
        System.out.println("Waiting for return key...");
        sc.nextLine();
        System.out.println("Return Key pressed...");
        notify();
        Thread.sleep(5000);
    }
}

public static void main(String[] args) throws InterruptedException
{
    Question5 processor = new Question5();
    Thread t1 = new Thread(new Runnable() {
        @Override
        public void run() {
            try{
                processor.produce();
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    });

    Thread t2 = new Thread(new Runnable() {
        @Override
        public void run() {
            try{
               processor.consumer();
            }
            catch (InterruptedException e)
            {
               e.printStackTrace();
            }
         }
      });

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

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

   }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/java -j
Producer thread running..
Waiting for return key...

Return Key pressed...
Resumed..

Process finished with exit code 0
```

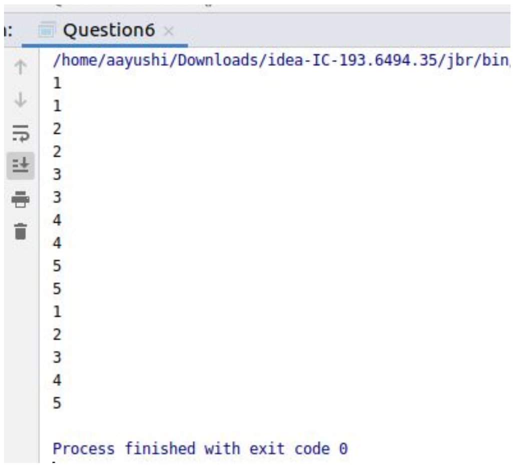
6. **Write a program to demonstrate sleep and join methods.**

sleep(): This method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.
join(): The join() method of a Thread instance is used to join the start of a thread's execution to end of another thread's execution such that a thread does not start running until another thread ends.

CODE :

```
package aayushi;
public class Question6  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) throws InterruptedException {
     Question6 t1=new Question6();
     Question6 t2=new Question6();
     Question6 t3=new Question6();
     t1.start();
     t2.start();

     try{
        t1.join();
        t2.join();

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

     t3.start(); //t3 thread starts with the t1 and t2 finish execution
  }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin
1
1
2
2
3
3
4
4
5
5
1
2
3
4
5

Process finished with exit code 0
```

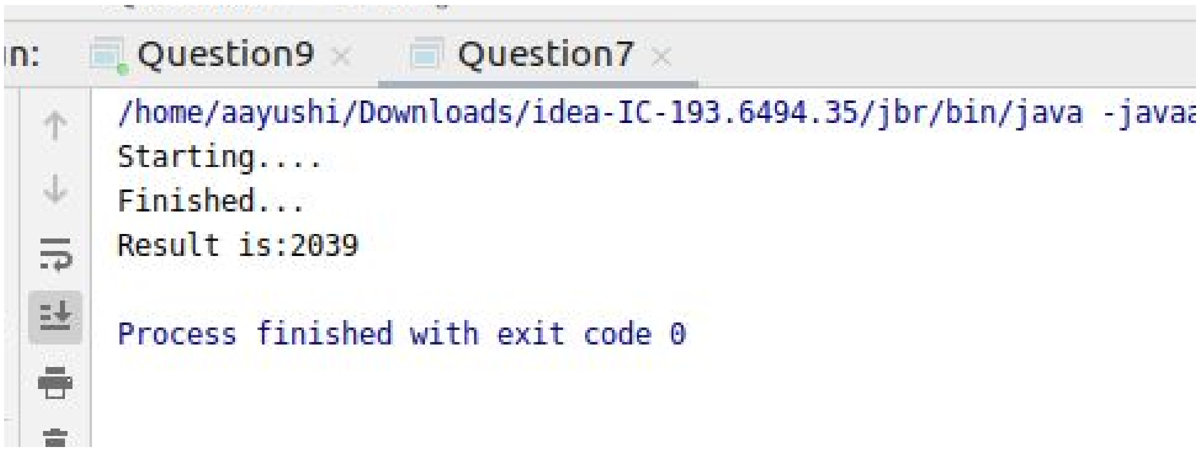
7. **Run a task with the help of callable and store it's result in the Future.**

There are two ways of creating threads – one by extending the Thread class and other by creating a thread with a Runnable. However, one feature lacking in Runnable is that we cannot make a thread return result when it terminates, i.e. when run() completes. For supporting this feature, the Callable interface is present in Java.

CODE :

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

public class Question7 {
  public static void main(String[] args) {
     ExecutorService executor = Executors.newCachedThreadPool();

     Future<Integer> future = executor.submit(new Callable<Integer>() {
        @Override
        public Integer call() throws Exception {
           Random random = new Random();
           int duration = random.nextInt(4000);
           System.out.println("Starting....");

           Thread.sleep(duration);
           System.out.println("Finished...");
           return duration;
        }
     });
     executor.shutdown();

     try{
        System.out.println("Result is:"+ future.get());
     }
     catch (InterruptedException | ExecutionException e)
     {
        e.printStackTrace();
     }
  }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/java -javaa
Starting....
Finished...
Result is:2039

Process finished with exit code 0
```

8. **Write a program to demonstrate the use of semaphore**

In short, a semaphore maintains a set of permits (tickets), each acquire() will take a permit (ticket) from semaphore, each release() will return back the permit (ticket) back to the semaphore. If permits (tickets) are not available, acquire() will block until one is available.

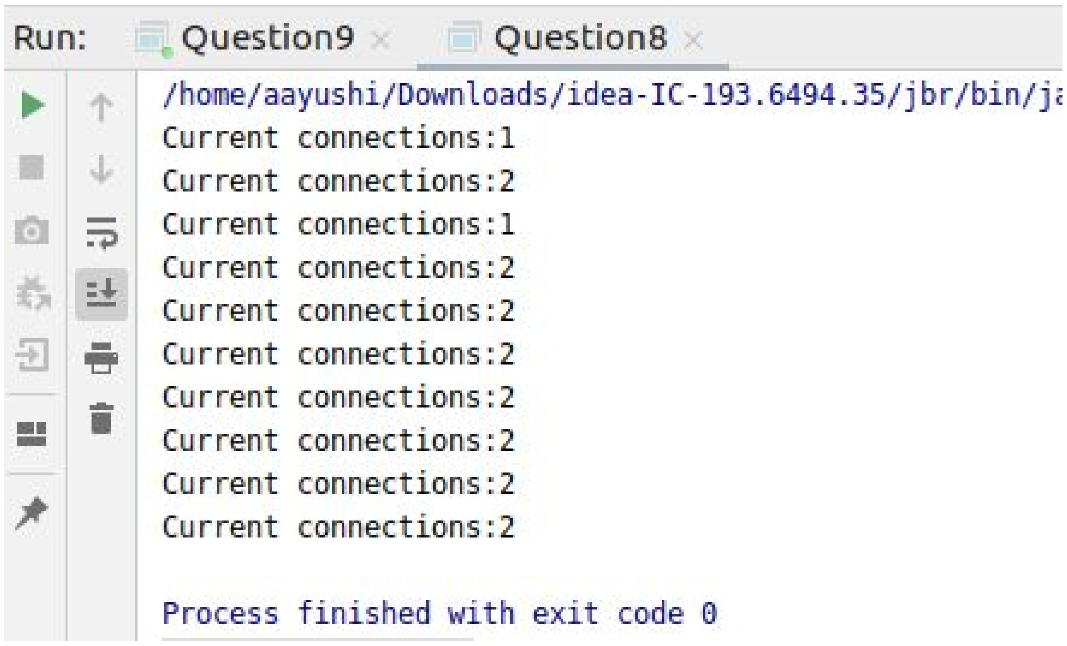
```
// 5 tickets
Semaphore semaphore = new Semaphore(5);

// take 1 ticket
semaphore.acquire();

// 4
semaphore.availablePermits();

// return back ticket
semaphore.release();

// 5
semaphore.availablePermits();
```

CODE :

```
package aayushi;

import java.util.concurrent.*;

class connection{
  private Semaphore sem = new Semaphore(2);
```

```
   private int connections=0;

   public void connect() throws InterruptedException {
      sem.acquire();
      synchronized (this)
      {
         connections++;
         System.out.println("Current connections:"+ connections);
      }
      Thread.sleep(2000);
      synchronized (this)
      {
         connections--;
      }
      sem.release();
   }
}

public class Question8 {
   public static void main(String[] args) throws Exception{
      ExecutorService executor = Executors.newCachedThreadPool();
      connection obj = new connection();

      for (int i=0;i<10;i++)
      {
         executor.submit(new Runnable() {
            @Override
            public void run() {
               try {
                  obj.connect();
               } catch (InterruptedException e) {
                  e.printStackTrace();
               }
            }
         });
      }
      executor.shutdown();
      executor.awaitTermination(1, TimeUnit.DAYS);
   }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/ja
Current connections:1
Current connections:2
Current connections:1
Current connections:2
Current connections:2
Current connections:2
Current connections:2
Current connections:2
Current connections:2
Current connections:2

Process finished with exit code 0
```

9. **Write a program to demonstrate the use of CountDownLatch.**

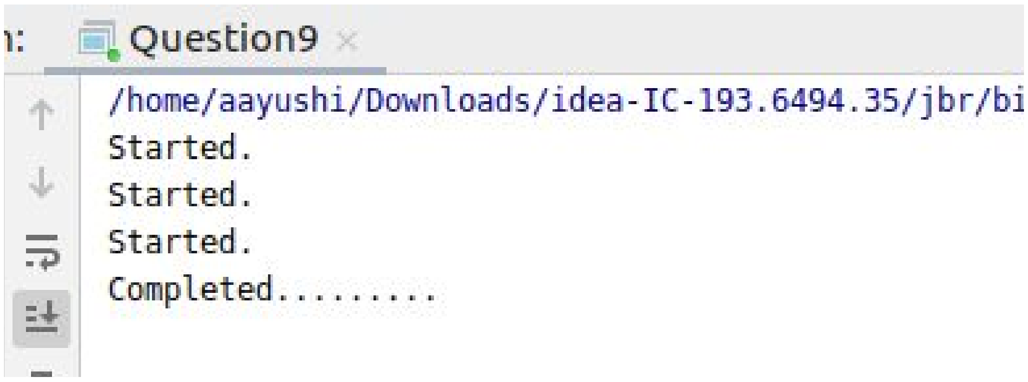
When we create an object of CountDownLatch, we specify the number of threads it should wait for, all such thread are required to do count down by calling CountDownLatch.countDown() once they are completed or ready to the job. As soon as count reaches zero, the waiting task starts running.

CODE :

```
package aayushi;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class Processor implements Runnable
{
  private CountDownLatch latch;
  public Processor(CountDownLatch latch)
  {
     this.latch=latch;
  }
  @Override
  public void run() {
     System.out.println("Started.");
```

```
        try {
            Thread.sleep(3000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        latch.countDown();
    }
}

public class Question9 {
    public static void main(String[] args) {
        CountDownLatch latch = new CountDownLatch(3);
        ExecutorService executor = Executors.newFixedThreadPool(3);

        for(int i=0;i<3;i++)
        {
            executor.submit(new Processor(latch));
        }

        try{
            latch.await();
        }
        catch (InterruptedException ex)
        {
            ex.printStackTrace();
        }

        System.out.println("Completed.........");
    }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bi
Started.
Started.
Started.
Completed.........
```

10. Write a program which creates deadlock between 2 threads.

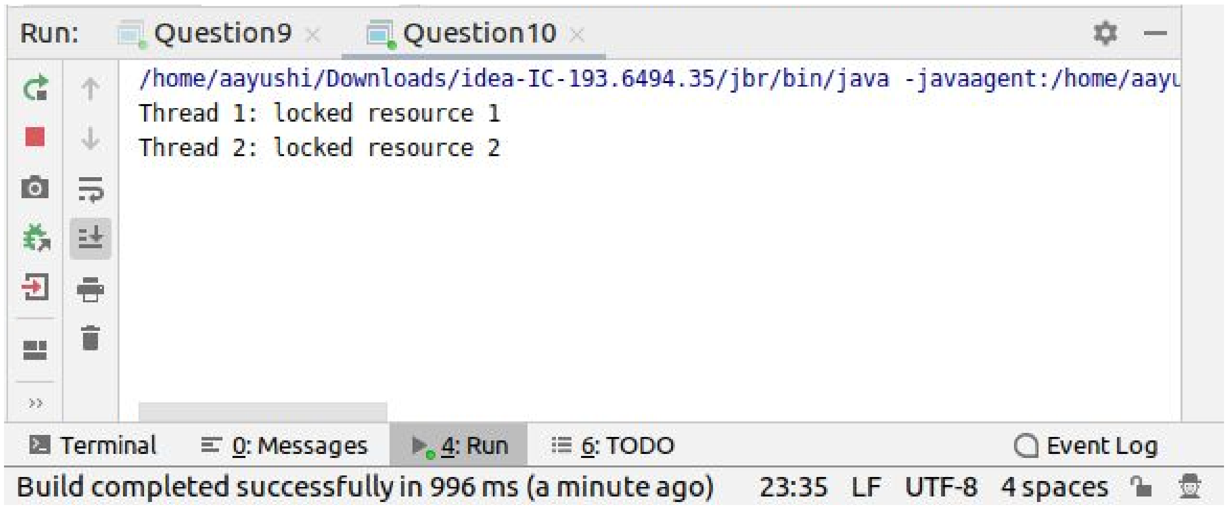
Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since both threads are waiting for each other to release the lock, the condition is called deadlock.

CODE :

```
package aayushi;
public class Question10 {
  public static void main(String[] args) {
     final String resource1 = "Aayushi";
     final String resource2 = "Pragya";
     // t1 tries to lock resource1 then resource2
     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");
              }
           }
        }
     };

     // t2 tries to lock resource2 then resource1
     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();
    }
}
```

OUTPUT



```
/home/aayushi/Downloads/idea-IC-193.6494.35/jbr/bin/java -javaagent:/home/aayu
Thread 1: locked resource 1
Thread 2: locked resource 2
```

Build completed successfully in 996 ms (a minute ago) 23:35 LF UTF-8 4 spaces