(A). AIM

Write a Java program that implements producer-consumer problem.

THEORY

Producer is producing some items, whereas there is one Consumer that is consuming the items produced by the Producer. The same memory buffer is shared by both producers and consumers which is of fixed-size.

The task of the Producer is to produce the item, put it into the memory buffer, and again start producing items. Where as the task of the Consumer is to consume the item from the memory buffer.

ALGORITHM

STEP 1 : START

STEP 2 : Create the Producer and Consumer classes and create as Threads.

STEP 3 : Create shared resource Q class which is having synchronized methods

STEP 4 : Start the Threads.

STEP 5 : Use notify() and wait() alternatively access the shared data.

STEP 6 : Producer produce the data.

STEP 7 : Consumer consume the data.

STEP 8 : STOP

SOURCE CODE

```
package firstThread;

class DataStore

{

  int i;

  boolean item=false;

  public synchronized void set(int i)

  {
```

```
  if(item)
  {
     try{
     wait();
     }
     catch(InterruptedException e){}
  }

     this.i=i;
     item=true;
     System.out.println("Produced"+i);
     notify();

}
public synchronized void get()
{

  if(!item)
  {
     try{
     wait();
     }
     catch(InterruptedException e){}

  }
```

```
        item=false;
        System.out.println("Consumer"+i);
        notify();

    }
}
class Producer implements Runnable{
    DataStore d;
    Producer(DataStore d)
    {
     this.d=d;
     Thread t=new Thread(this,"Producer");
     t.start();
    }
    public void run()
    { int i=0;
       while(true){
         d.set(i++);
         try{
             Thread.sleep(500);
         }
         catch(InterruptedException a){}

       }


    }
```

```
}
class Consumer implements Runnable{
    DataStore d;
   Consumer(DataStore d)
   {
     this.d=d;
     Thread t=new Thread(this,"Producer");
     t.start();
   }
   public void run()
   {
     while(true)
     {
         d.get();
         try{
             Thread.sleep(500);
         }
         catch(InterruptedException a){}
     }

   }
}

public class ProducerConsumerEx {

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

```
        DataStore d1=new DataStore();
        new Producer(d1);
        new Consumer(d1);

    }
}
```

OUTPUT:

Put: 0

Got: 0

Put: 1

Got: 1

Put: 2

Got: 2

Put: 3

Got: 3

Put: 4

Got: 4

.

.

.

.

etc

VIVA VOCE

1. What is inter thread communication?

   Inter-thread communication in Java is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.

2. What is the use of wait()?

   In Java, the wait() method is used to pause the execution of a thread until another thread signals that it can resume. When a thread calls wait() on an object, it releases the lock on the object and waits until another thread calls notify() or notifyAll() on the same object.

3. What is the use of notify()?

   The notify() method is defined in the Object class, which is Java's top-level class. It's used to wake up only one thread that's waiting for an object, and that thread then begins execution. The thread class notify() method is used to wake up a single thread.

4. What is the purpose of notifyAll()?

   notifyAll() wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods. The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object.

5. What is Synchronization?

   Synchronization in java is the capability to control the access of multiple threads to any shared resource. In the Multithreading concept, multiple threads try to access the shared resources at a time to produce inconsistent results. The synchronization is necessary for reliable communication between threads.