UNIVERSITI PUTRA MALAYSIA

FAKULTI SAINS KOMPUTER DAN TEKNOLOGI MAKLUMAT

Assignment 6

ASSIGNMENT TITLE: Implementing Mutual Exclusion

ASSIGNMENT TYPE: Individual

OBJECTIVE:
The main objective of this assignment is to help student to understand the mutual exclusion algorithms used in operating system, by implementing the algorithm using computer program.

PROGRAMME OBJECTIVE (PO):
PO2: Coding algorithms used in operating system

RUBRIC:

CRITERIA	**LEVEL 1 Very Poor**	**LEVEL 2 Poor**	**LEVEL 3 Good**	**LEVEL 4 Very Good**	**LEVEL 5 Excellent**
The ability to perform programming task	Unable to perform programming task with very limited success (< 40%).	Able to perform programming task with limited success (≥ 40%,< 60%).	Able to perform programming task with some success (≥ 60%, < 80%).	Able to perform programming task with considerable success (≥ 80%, < 100%).	Able to perform programming task with outstanding success (100%).

INSTRUCTIONS:

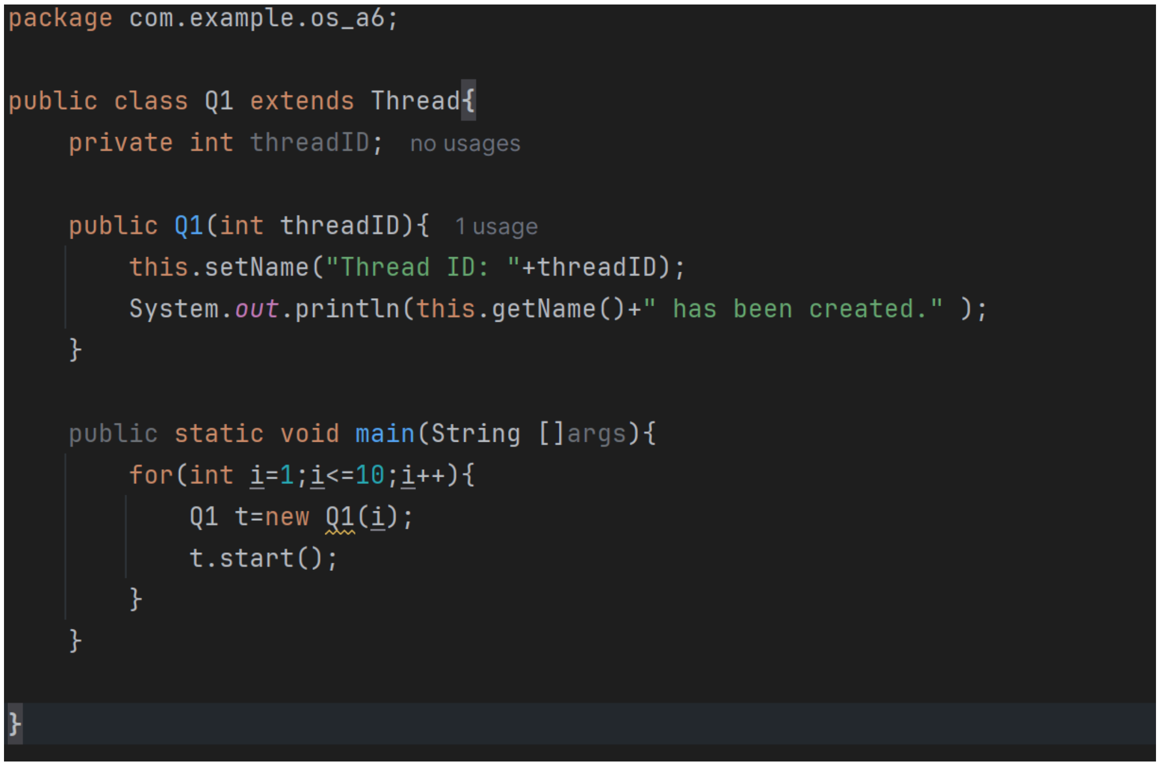
1. Implement multithreading in Java

Java allows a program to create threads during runtime by extending the Thread class or implementing the interface Runnable.

Build a class to represent a thread. Each thread should be able to store and will be given a unique id so that your program will be able to track them (if you extends the class Thread, you may use the getName() and setName() methods from the Thread class).

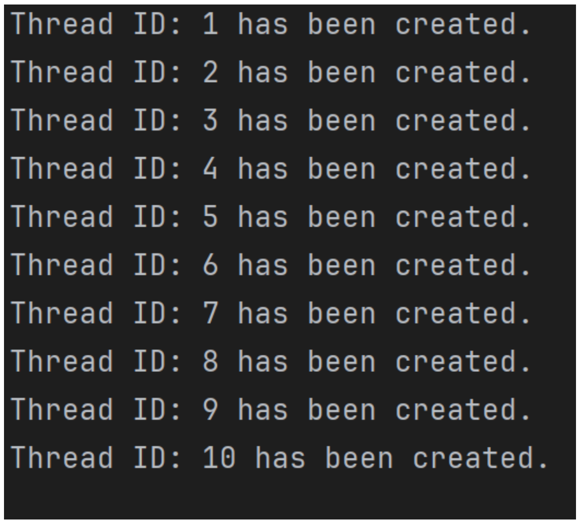
Build a Java program which creates 10 threads. Assign a unique id for each thread. You may use an integer as a counter and increase the counter every time a thread is created. A thread should print a message when it is created e.g. 'Thread 1 is created' (for thread with id 1).

SOURCE CODE :



```java
package com.example.os_a6;

public class Q1 extends Thread{
    private int threadID;  no usages

    public Q1(int threadID){  1 usage
        this.setName("Thread ID: "+threadID);
        System.out.println(this.getName()+" has been created." );
    }

    public static void main(String []args){
        for(int i=1;i<=10;i++){
            Q1 t=new Q1(i);
            t.start();
        }
    }

}
```

OUTPUT:



```
Thread ID: 1 has been created.
Thread ID: 2 has been created.
Thread ID: 3 has been created.
Thread ID: 4 has been created.
Thread ID: 5 has been created.
Thread ID: 6 has been created.
Thread ID: 7 has been created.
Thread ID: 8 has been created.
Thread ID: 9 has been created.
Thread ID: 10 has been created.
```

2. Implement concurrent processing using multithreading

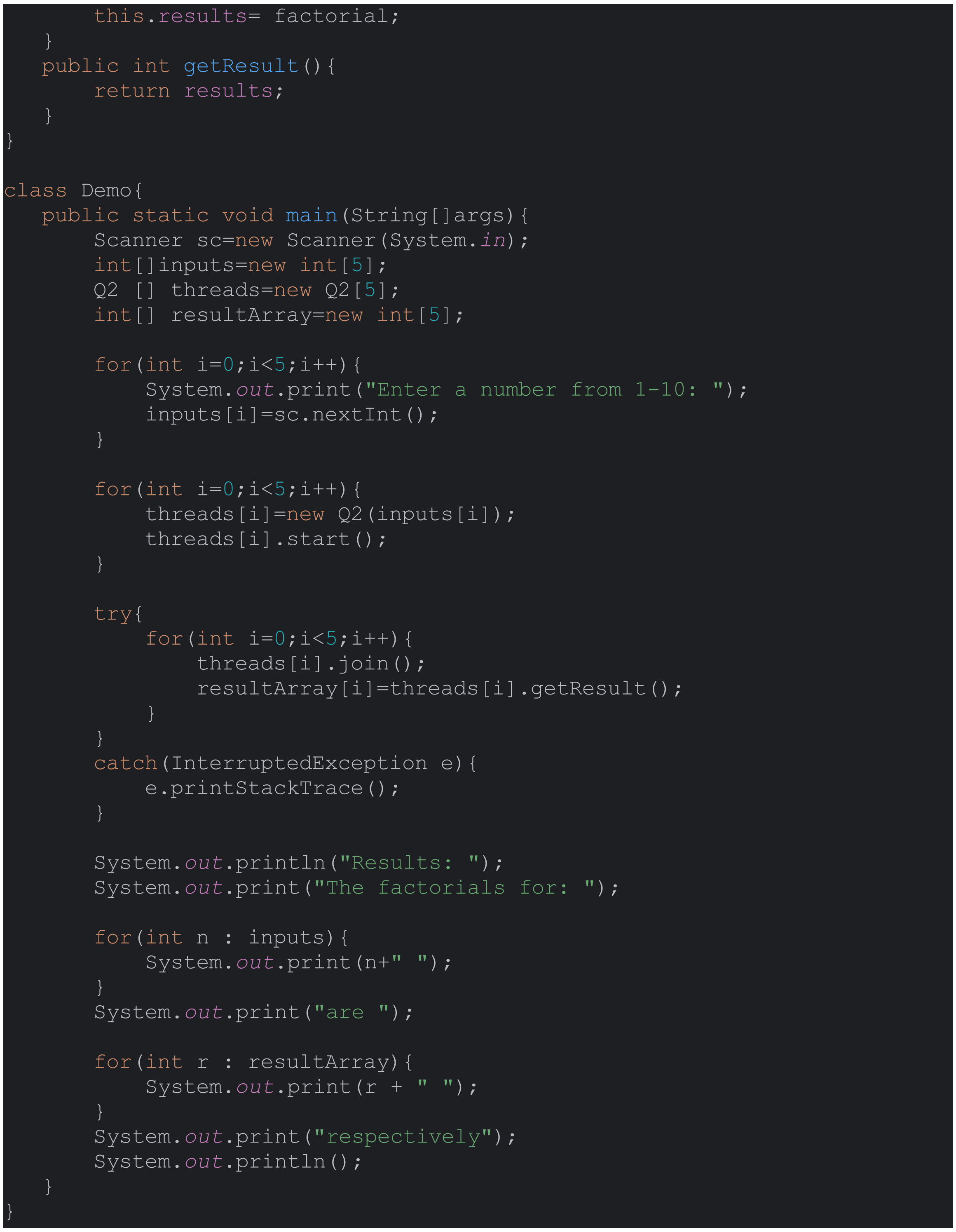
Create a Java program which calculates the factorial for 5 numbers (values ranging from 1 to 10) specified by user.
E.g.

```
Enter 5 numbers [1-10]: 10 3 6 7 4

The factorials for 10 3 6 7 4 are 3628800 6 720 5040 24
```

You must use multithreading to perform the processing, by creating a thread to calculate the factorial for each number given. The threads may store all the results into a single array. Then print out the results after all results has been calculated and stored inside the array (You may also use other data structure other than array if it is more suitable).

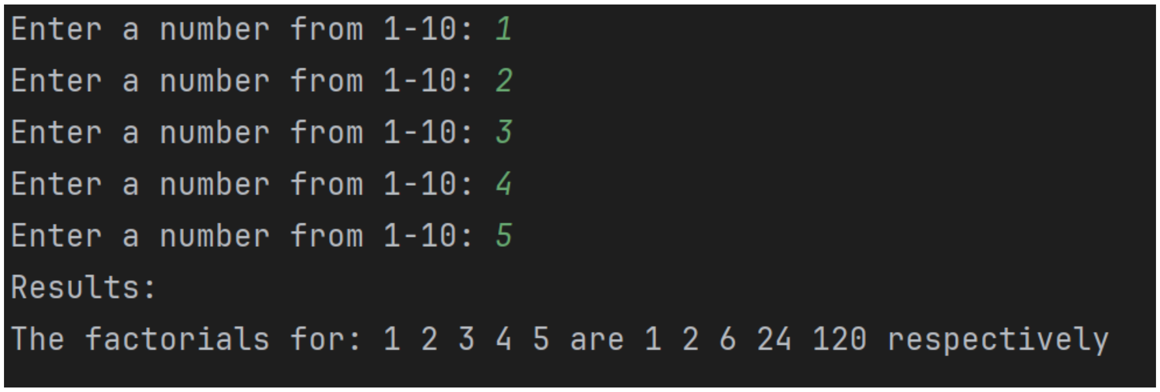
SOURCE CODE:

```
package com.example.os_a6;
import java.util.Scanner;

public class Q2 extends Thread {
   private int num;
   private int results;

   public Q2(int num){
       this.num=num;
   }

   @Override
   public void run(){
       int factorial=1;
       for (int j=1;j<=num;j++){
           factorial=factorial*j;
       }
```



```
        this.results= factorial;
    }
    public int getResult(){
        return results;
    }
}

class Demo{
    public static void main(String[]args){
        Scanner sc=new Scanner(System.in);
        int[]inputs=new int[5];
        Q2 [] threads=new Q2[5];
        int[] resultArray=new int[5];

        for(int i=0;i<5;i++){
            System.out.print("Enter a number from 1-10: ");
            inputs[i]=sc.nextInt();
        }

        for(int i=0;i<5;i++){
            threads[i]=new Q2(inputs[i]);
            threads[i].start();
        }

        try{
            for(int i=0;i<5;i++){
                threads[i].join();
                resultArray[i]=threads[i].getResult();
            }
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }

        System.out.println("Results: ");
        System.out.print("The factorials for: ");

        for(int n : inputs){
            System.out.print(n+" ");
        }
        System.out.print("are ");

        for(int r : resultArray){
            System.out.print(r + " ");
        }
        System.out.print("respectively");
        System.out.println();
    }
}
```

OUTPUT:



```
Enter a number from 1-10: 1
Enter a number from 1-10: 2
Enter a number from 1-10: 3
Enter a number from 1-10: 4
Enter a number from 1-10: 5
Results:
The factorials for: 1 2 3 4 5 are 1 2 6 24 120 respectively
```

3. Implement mutual exclusion with multithreading in Java

Build a Java program which simulates a communication between a server and clients. You must use a thread to represent each party (as server or client).

Create three threads. One thread will act as a server, which always ready to receive a message from a client (*ping*) and then replies the message to the same client (*pong*). The other two threads will become the clients, where each client will send a message to the server (*ping*) and waits for the reply (*pong*). Define 2 different classes to represent the server and the client.

To perform the communication, each thread will refer to a same variable/data structure for sending and receiving the message. Use mutual exclusion approach to ensure each client can send and receive the reply without interruption from any other client(s).

For this simulation, your objective is to ensure that each client must successfully send 10 *pings* (and thus the server must successfully perform 20 *pong*s) and then the program can terminate.

Example output:

```
      Client 1 ping 1
      Server pong 1 to client 1
      Client 2 ping 1
      Server pong 1 to client 2
      Client 2 ping 2
      Server pong 2 to client 2
      Client 1 ping 2
      Server pong 2 to client 1
...
      Client 1 ping 9
      Server pong 9 to client 1
      Client 2 ping 9
      Server pong 9 to client 2
      Client 2 ping 10
      Server pong 10 to client 2
      Client 1 ping 10
      Server pong 10 to client 1
      Simulation ends
```

SOURCE CODE:

```
package com.example.os_a6_q3;

class CommBuffer {
   private int clientId;
   private int messageId;

   private boolean hasMessage = false;

   public synchronized void sendPing(int clientId, int messageId) {
       while (hasMessage) {
           try {
               wait();
           }
           catch (InterruptedException e) {
               Thread.currentThread().interrupt();
           }
       }

       this.clientId = clientId;
       this.messageId = messageId;
       this.hasMessage = true;
       System.out.println("Client " + clientId + " ping " + messageId);
       notifyAll();
       while (hasMessage) {
           try {
               wait();
           }
           catch (InterruptedException e) {
               Thread.currentThread().interrupt();
           }
       }
   }
   public synchronized void replyPong() {
       while (!hasMessage) {
           try {
               wait();
           }
           catch (InterruptedException e) {
               Thread.currentThread().interrupt();
           }
       }
       System.out.println("Server pong " + messageId + " to client " +
clientId);
       this.hasMessage = false;
       notifyAll();
   }
}

class Server implements Runnable {
   private CommBuffer buffer;
   private int totalInteractions;

   public Server(CommBuffer buffer, int totalInteractions) {
       this.buffer=buffer;
       this.totalInteractions = totalInteractions;
   }
```

```java
   @Override
   public void run() {
       for (int i = 0; i < totalInteractions; i++) {
           buffer.replyPong();
       }
   }
}

class Client implements Runnable {
   private int id;
   private CommBuffer buffer;

   public Client(int id, CommBuffer buffer) {
       this.id = id;
       this.buffer = buffer;
   }

   @Override
   public void run() {
       // Each client sends 10 pings
       for (int i = 1; i <= 10; i++) {
           buffer.sendPing(id, i);
       }
   }
}

public class PingPong {
   public static void main(String[] args) {
       CommBuffer buffer = new CommBuffer();
       Thread serverThread = new Thread(new Server(buffer, 20));
       Thread client1 = new Thread(new Client(1, buffer));
       Thread client2 = new Thread(new Client(2, buffer));
       serverThread.start();
       client1.start();
       client2.start();
       try {
           client1.join();
           client2.join();
           serverThread.join();
       }
       catch (InterruptedException e) {
           e.printStackTrace();
       }

       System.out.println("Simulation ends");
   }
}
```

OUTPUT:

```
Client 1 ping 1
Server pong 1 to client 1
Client 1 ping 2
Server pong 2 to client 1
Client 1 ping 3
Server pong 3 to client 1
Client 1 ping 4
Server pong 4 to client 1
Client 1 ping 5
Server pong 5 to client 1
Client 1 ping 6
Server pong 6 to client 1
Client 1 ping 7
Server pong 7 to client 1
Client 1 ping 8
Server pong 8 to client 1
Client 1 ping 9
Server pong 9 to client 1
Client 1 ping 10
Server pong 10 to client 1
Client 2 ping 1
Server pong 1 to client 2
Client 2 ping 2
Server pong 2 to client 2
Client 2 ping 3
Server pong 3 to client 2
Client 2 ping 4
Server pong 4 to client 2
Client 2 ping 5
Server pong 5 to client 2
Client 2 ping 6
Server pong 6 to client 2
Client 2 ping 7
Server pong 7 to client 2
Client 2 ping 8
Server pong 8 to client 2
Client 2 ping 9
Server pong 9 to client 2
Client 2 ping 10
Server pong 10 to client 2
Simulation ends
```