United States International University

MIS 6060: DISTRIBUTED COMPUTING & INTERNET TECHNOLOGY

Lab Exercise 5: Implementing the Two Phase Commit (2PC) protocol

Objective

To show how messages are passed between 'n' nodes (or terminals) and actions are committed or aborted, with respect to the response of all other nodes. The final action is the decision gets communicated to all nodes.

Requirements

The NetBeans IDE will be used to demonstrate this exercise.

The distributed commit problem involves having an operation being performed by each member of process group, or none at all. In the case of reliable multi-casting, the operation is the delivery of a message. With distributed transaction the operation may be committing of transaction at a single site that takes part in transaction. Distributed commit is often established by means of a co-coordinator. In a simple scheme this co-coordinator tells all other processes that are also involves, called participants, whether or not to perform the operation in question. This scheme is referred to as a one phase commit protocol. It has the obvious drawback that if one of the participants cannot actually perform the operation, there is no way to tell the coordinator. In practice, more sophisticated schemes are needed, the most common one being the two phase commit protocol. The main drawback of this protocol is that it cannot efficiently handle the failure of the coordinator.

Two Phase Commit:

The original two phase commit (2PC) protocol is attributed to Gray (1978). Without loss of generality, consider a distributed transaction involving the participation of a number of processes each running on different machines. The 2PC protocol consists of following two phases, each consisting of two steps;

- The coordinator sends a VOTE_REQUEST message to all participants.
- When each participant receives the VOTE_REQUEST message, it returns either a VOTE_COMMIT message to the coordinator informing the coordinator that it is prepared to locally commit its part of the transaction, or otherwise a VOTE_ABORT message.

- The coordinator collects the entire vote from the participants. If the participants have voted to commit the transaction, then so will the coordinator. In that case, it sends GLOBAL_COMMIT message to all other participants. However, if any one participant votes to abort the transaction the coordinator aborts the transaction and multi-casts a GLOBAL_ABORT message.
- Each participant that votes for a commit waits for a final reaction from the coordinator. If the participant receives a GLOBAL_COMMIT message, it locally commits the transaction. Otherwise, upon receipt of a GLOBAL_ABORT message from the coordinator, the transaction is locally aborted.

Algorithm Used:

A) Actions by the coordinator:

```
Write START_2PC to local log;
Multicast VOTE_REQUEST to all participants;
While not all votes have been
collected {
        Wait for any incoming vote;
        If timeout
        {
                Write GLOBAL_ABORT to local log;
                Multicast GLOBAL_ABORT to all
                participants; Exit;
        }
        rec
        ord
        vot
        e;
}
If all participant sent VOTE_COMMIT and coordinator votes COMMIT
{
        Write GLOBAL_COMMIT to local log;
        Multicast GLOBAL_COMMIT to all participants;
}

e
l
s
e
{
        Write GLOBAL_ABORT to local log;
        Multicast GLOBAL_ABORT to all participants;
}
```

B) Actions by participants:

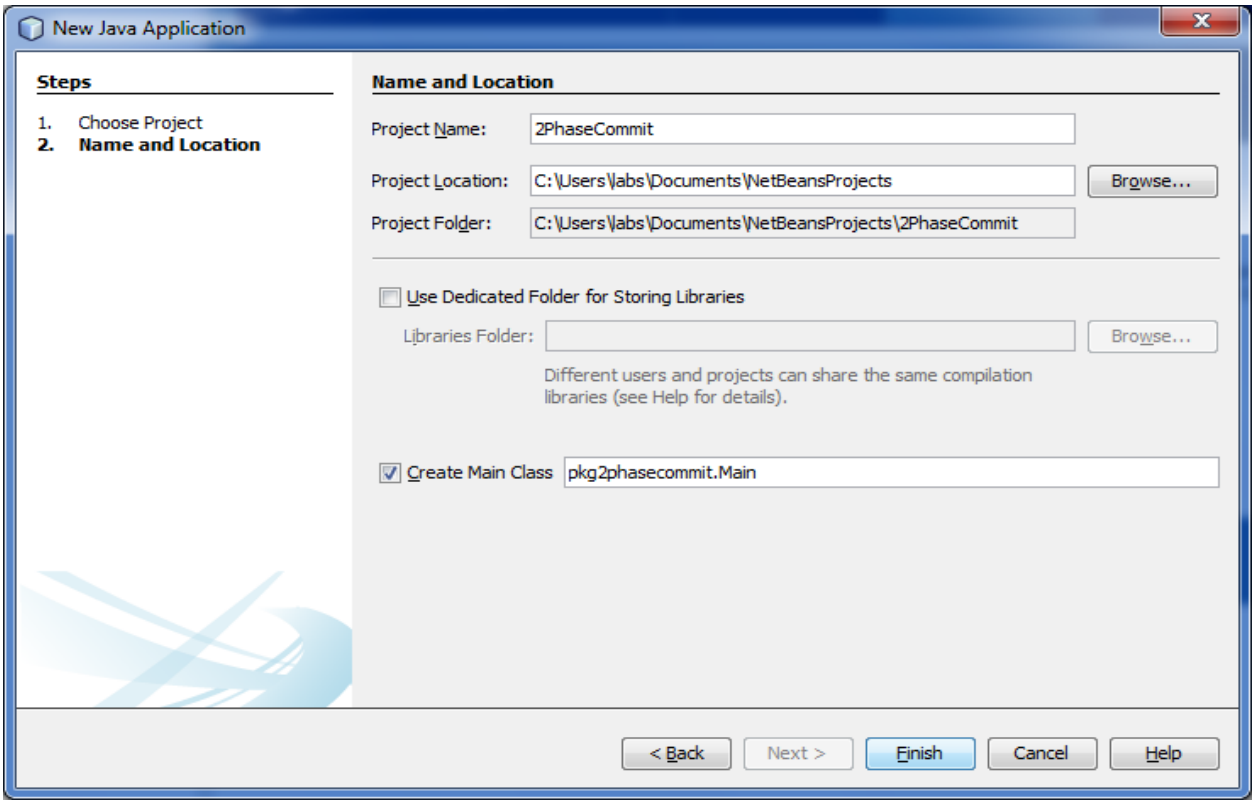
```
Write INIT to local log;
Wait for VOTE_REQUEST from coordinator;
If timeout
{ write VOTE_ABORT to local
        log;
        e
xit;
}
if participant vote COMMIT
{
        write VOTE_COMMIT to local
        log; send VOTE_COMMIT to
        coordinator; wait for
        DECISION from coordinator;
        if timeout
        { multicast DECISION_REQUEST to other
                participants; wait until DECISION is
                received;/*remain blocked*/ write DECISION
                to local log;
        }
        if DECISION= =
                GLOBAL_COMMIT write
                GLOBAL_COMMIT to local
                log;
        else if DECISION= =
                GLOBAL_ABORT write
                GLOBAL_ABORT to local
                log;
}

e
l
s
e
{ write VOTE_ABORT to local
        log;
        send VOTE_ABORT to coordinator;
}
```

C) Action for handling decision request:

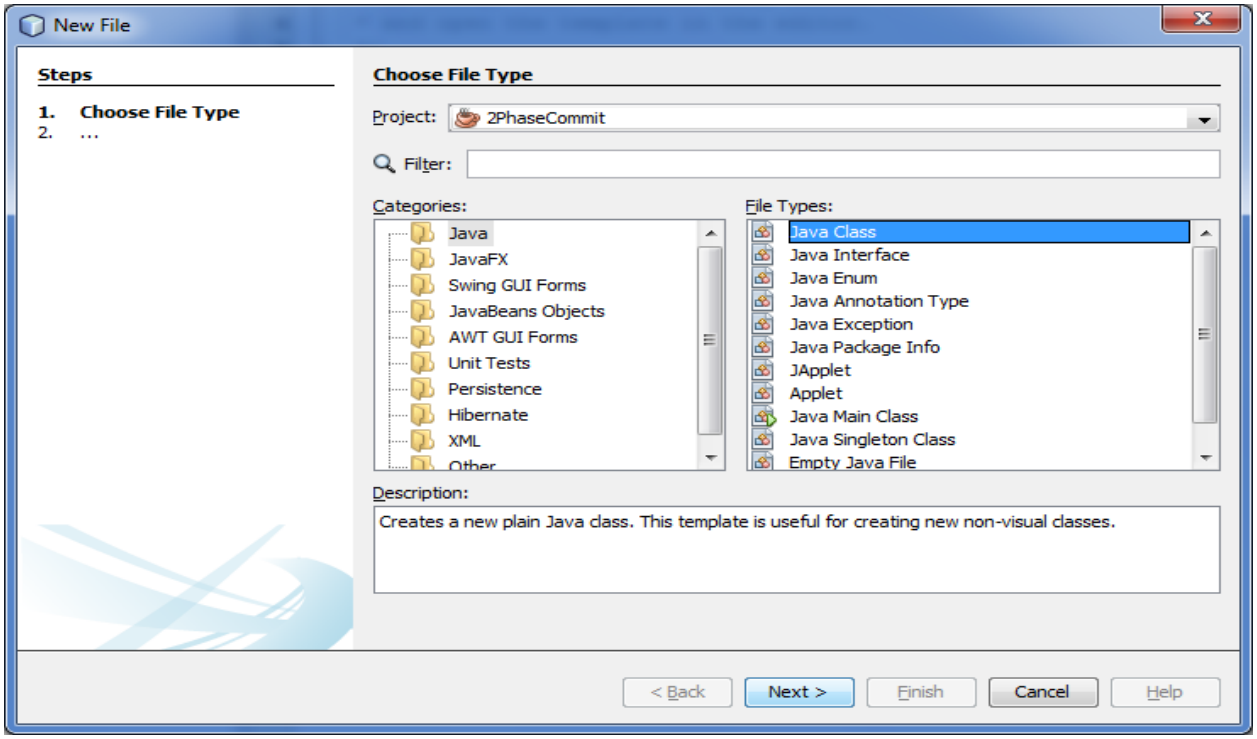
```
While true
{ wait until any incoming DECISION_REQUEST is received;/*remain
        blocked*/
        read most recently recorded STATES from the
        local log; if STATE= = GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting
        participants; else if STATE= =INIT or STATE=
        = GLOBAL_ABORT
        send GLOBAL_ABORT to requesting
        participants; else
        skip;/* participant remains blocked.*/
}
```

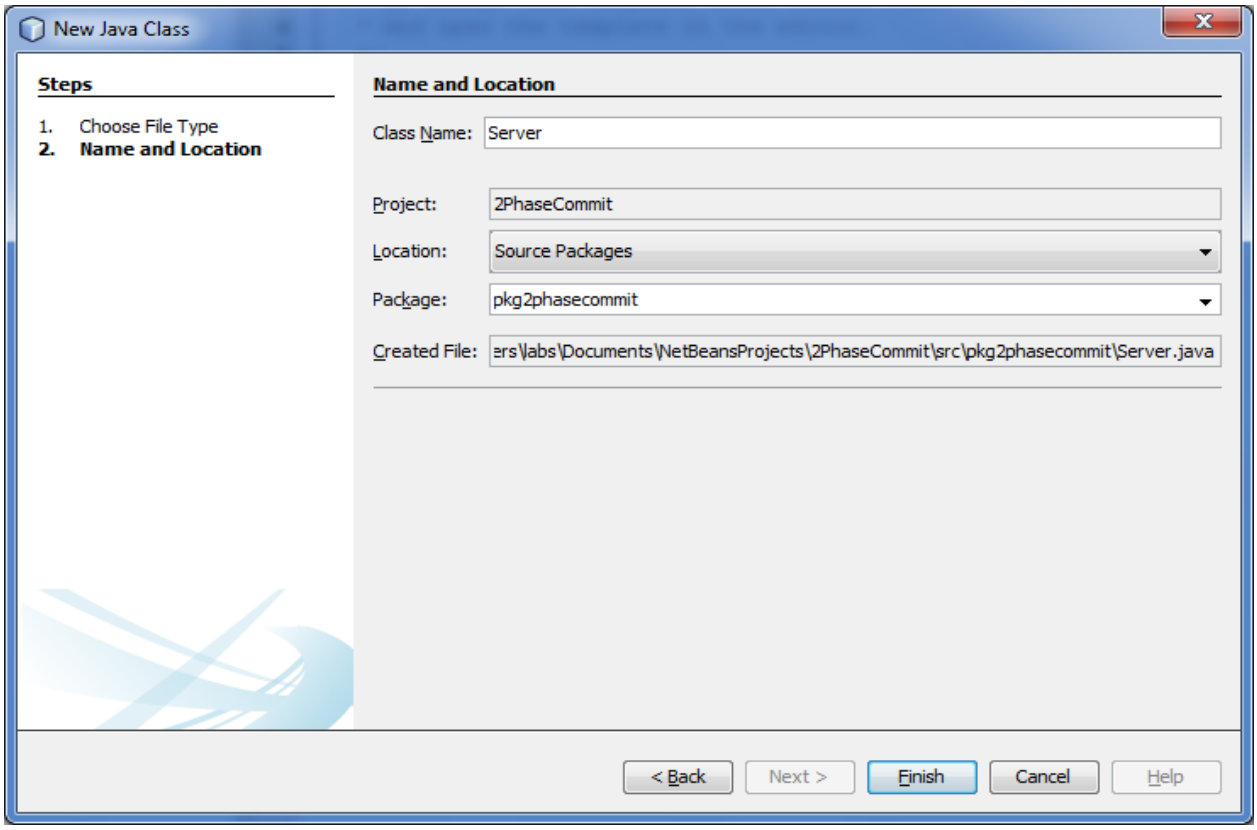
Step 1

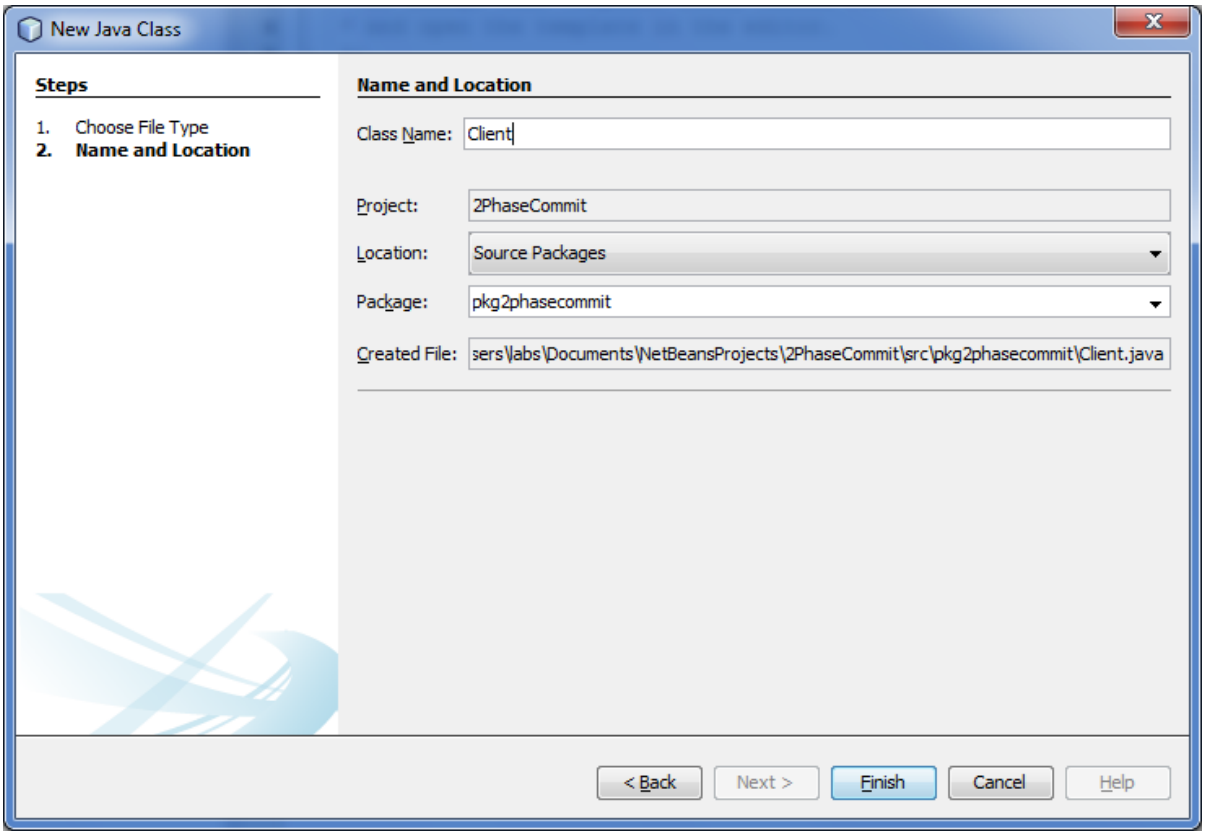
Create a new project and name it **2PhaseCommit**.



Create two new classes: ***Server.java*** class and ***Client.java*** class.







Step 2

Insert the following code in to the Server class.

Server.java code

```
import
java.io.*;
import
java.net.*;
import java.util.*;

public class Server
 {
        boolean closed=false,inputFromAll=false;
//like an array list. Bind it with objects of type clientThread
List<clientThread> t;
        List<String> data;
        Server()
        {
                t = new ArrayList<clientThread>();
                data= new ArrayList<String>();
```

```
	}
	public static void main(String args[])
	{
	Socket clientSocket = null;
 ServerSocket serverSocket = null;
	int port_number=1111;

Server ser=new Server();
		try
		{
		serverSocket = new ServerSocket(port_number);
	}
	catch (IOException e)
	{
			System.out.println(e);
		}
		while(!ser.closed)
			{
	try
			{
			clientSocket = serverSocket.accept();
				clientThread th=new clientThread(ser,clientSocket);
			(ser.t).add(th);
	System.out.println("\nNow Total clients are : "+(ser.t).size());
			(ser.data).add("NOT_SENT");
			th.start();
			}
				catch (IOException e)
				{
				}
		}
	try
	{
		serverSocket.close();
		}
		catch(Exception e1)
		{
		}
	}
}

 class clientThread extends Thread
 {
	DataInputStream is = null;
	String line;
```

```
        String destClient="";
        String name;
        PrintStream os = null;
        Socket clientSocket = null;
        String clientIdentity;
        Server ser;

        public clientThread(Server ser,Socket clientSocket)
        {
this.clientSocket=clientSocket;
this.ser=ser;
        }

        public void run()
        {
                try
                        {                       is = new
DataInputStream(clientSocket.getInputStream());                 os =
new PrintStream(clientSocket.getOutputStream());
os.println("Enter your name.");
                        name = is.readLine();
                        clientIdentity=name;
 os.println("Welcome "+name+" to this 2 Phase Application.\nYou will receive a vote
Request now...");
                        os.println("VOTE_REQUEST\nPlease enter COMMIT or ABORT
to proceed : ");

                        for(int i=0; i<(ser.t).size(); i++)
                        {
                                if((ser.t).get(i)!=this)
                                {
                                        ((ser.t).get(i)).os.println("---A new user
"+name+" entered the Appilcation---");
                                }
                        }
                            while (true)
                                {
                        line = is.readLine();
                                if(line.equalsIgnoreCase("ABORT"))
                        {
                       System.out.println("\nFrom "'+clientIdentity+"' :
ABORT\n\nSince aborted we will not wait for inputs from other clients.");
                        System.out.println("\nAborted....");
                                        for(int i=0; i<(ser.t).size(); i++)
                                        {
```

```
        ((ser.t).get(i)).os.println("GLOBAL_ABORT");
                                                        ((ser.t).get(i)).os.close();
                                                        ((ser.t).get(i)).is.close();
                                                }
                                                break;
                                        }

if(line.equalsIgnoreCase("COMMIT"))
        {
                System.out.println("\nFrom '"+clientIdentity+"' :
COMMIT");
                                                        if((ser.t).contains(this))
                                                {
                                                (ser.data).set((ser.t).indexOf(this), "COMMIT");
                        for(int j=0;j<(ser.data).size();j++)

{


if(!(((ser.data).get(j)).equalsIgnoreCase("NOT_SENT")))
                                        {
                                                                                        ser.inputFromAll=true;
                                                continue;
                                                        }
                                                                else
                                                        {
                                                                        ser.inputFromAll=false;

        System.out.println("\nWaiting for inputs from other clients.");
                                                        break;
                                                        }
                                        }
                if(ser.inputFromAll)
                                                        {

System.out.println("\n\nCommited....");
                for(int i=0; i<(ser.t).size(); i++)
                                                                {

        ((ser.t).get(i)).os.println("GLOBAL_COMMIT");
                                ((ser.t).get(i)).os.close();
                                ((ser.t).get(i)).is.close();
```

```
}
break;
                                        }
                                        }//if t.contains
                        }//commit
                        }//while
                                ser.closed=true;
                                clientSocket.close();
                }
                catch(IOException e)
                {
                };
        }
}
```

Step 3

Insert the following code into the Client class.

***Client.java* code**

```
import java.io.*;
import java.net.*;

public class Client implements Runnable
{
        static Socket clientSocket =
null;           static PrintStream os =
null;           static DataInputStream
is = null;      static BufferedReader
inputLine = null;       static boolean
closed = false;

        public static void main(String[] args)
        {               int
port_number=1111;
String host="localhost";
try
                {
                                clientSocket = new Socket(host, port_number);
                                inputLine = new BufferedReader(new
InputStreamReader(System.in));
                os = new PrintStream(clientSocket.getOutputStream());
                is = new DataInputStream(clientSocket.getInputStream());
```

```
                }
        catch (Exception
e)
                {
          System.out.println("Exception occurred : "+e.getMessage());
                }
                if (clientSocket != null && os != null && is != null)
        {
try
{
                        new Thread(new Client()).start();
                        while (!closed)
                {
                                os.println(inputLine.readLine());
                }
                        os.close();
                is.close();
                clientSocket.close();
                }
                catch (IOException e)
                {
                                System.err.println("IOException:  " + e);
                }
                }
        }

        public void run()
        {
                String responseLine;
                try
                        {                       while
((responseLine = is.readLine()) != null)
                                {
                System.out.println("\n"+responseLine);
                if
(responseLine.equalsIgnoreCase("GLOBAL_COMMIT")==true ||
responseLine.equalsIgnoreCase("GLOBAL_ABORT")==true )
                                {
                                        break;
                                }
                                }
                        closed=true;
                }
                catch (IOException e)
                {
```

```
                    System.err.println("IOException:  " + e);
                }
            }
}
```

Step 4

Output:

Run the *Server.java* code first then run the *Client.java* code and insert your name as the first Client when prompted, followed by up to 5 other clients.

1) Screen capture the client output for a i) VOTE_COMMIT and a ii) VOTE_ABORT scenario, save them to a word document.

2) Screen capture the server output for a i) GLOBAL_COMMIT and a ii) GLOBAL ABORT scenario, save them to the same word document used above then upload the single word document to Blackboard for marking.