1. **Socket Programming:**

A) Client.java:

```
package com.socketprogramming;
import java.io.*;
import java.net.*;

public class Client {
   public static void main(String[] args) {
      try {
         // Step 1: Create a Socket object and connect to the server (localhost, port 5000)
         Socket socket = new Socket("localhost", 5000);
         System.out.println("Connected to server!");

         // Step 2: Create input and output streams for communication
         BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

         // Step 3: Send a message to the server
         out.println("Hello from Client!");

         // Step 4: Read the server's response
         String response = in.readLine();
         System.out.println("Server says: " + response);

         // Step 5: Close the connection
         socket.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

B) Server.java:

```
package com.socketprogramming;
import java.io.*;
import java.net.*;

public class Server {
   public static void main(String[] args) {
      try {
         // Step 1: Create a ServerSocket object and bind it to a port (e.g., 5000)
         ServerSocket serverSocket = new ServerSocket(5000);
         System.out.println("Server is waiting for client connection...");

         // Step 2: Wait for a client to connect
         Socket socket = serverSocket.accept();
         System.out.println("Client connected!");

         // Step 3: Create input and output streams for communication
          BufferedReader in = new BufferedReader(new
          InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

         // Step 4: Read data from the client
         String message = in.readLine();
         System.out.println("Client says: " + message);

         // Step 5: Send a response back to the client
         out.println("Hello from Server!");

         // Step 6: Close the connection
         socket.close();
         serverSocket.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

2. Remote Method Invocation (RMI):

A) AddClient.java:

```
package com.rmi;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class AddClient {
   public static void main(String[] args) {
      try {
         Registry registry = LocateRegistry.getRegistry("localhost", 1099);
         AddServerIntf server = (AddServerIntf) registry.lookup("AddServer");
         double result = server.add(5.5, 10.5);
         System.out.println("Result: " + result);
      } catch (Exception e) {
         e.printStackTrace();
      }}}
```

B) AddServer.java:

```
package com.rmi;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class AddServer {
   public static void main(String[] args) {
      try {
         AddServerImpl server = new AddServerImpl();
         Registry registry = LocateRegistry.createRegistry(1099);
         registry.rebind("AddServer", server);
         System.out.println("Server is ready.");
      } catch (Exception e) {
         e.printStackTrace();
      }}}
```

C) AddServerIntf.java:

```
package com.rmi;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface AddServerIntf extends Remote {
   double add(double d1, double d2) throws RemoteException;
}
```

3. Common Object Request Broker Architecture (CORBA):

A) Calculator.idl:

```
module CalculatorApp {
   interface Calculator {
      double add(in double x, in double y);
      double subtract(in double x, in double y);
      double multiply(in double x, in double y);
      double divide(in double x, in double y);
   };
};
```

B) CalculatorServer.java:

```
import CalculatorApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
class CalculatorImpl extends CalculatorPOA {
   public double add(double x, double y) {
      return x + y;
   }
   public double subtract(double x, double y) {
      return x - y;
   }
   public double multiply(double x, double y) {
      return x * y;
   }
   public double divide(double x, double y) {
      if (y == 0) throw new org.omg.CORBA.BAD_PARAM("Cannot divide by zero");
      return x / y;
   }
}
public class CalculatorServer {
   public static void main(String args[]) {
      try {
         ORB orb = ORB.init(args, null);
         POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
```

```
            rootpoa.the_POAManager().activate();
            CalculatorImpl calculatorImpl = new CalculatorImpl();
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(calculatorImpl);
            Calculator href = CalculatorHelper.narrow(ref);
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            String name = "Calculator";
            NameComponent path[] = ncRef.to_name(name);
            ncRef.rebind(path, href);
            System.out.println("CalculatorServer ready and waiting...");
            orb.run();
        } catch (Exception e) {
            System.err.println("ERROR: " + e);
            e.printStackTrace(System.out);
        }}}
```

C) CalculatorClient.java:

```
import CalculatorApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class CalculatorClient {
  public static void main(String args[]) {
    try {
      ORB orb = ORB.init(args, null);
      org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      String name = "Calculator";
      Calculator calculator = CalculatorHelper.narrow(ncRef.resolve_str(name));
      double x = 10.5;
      double y = 2.5;
      System.out.println(x + " + " + y + " = " + calculator.add(x, y));
      System.out.println(x + " - " + y + " = " + calculator.subtract(x, y));
      System.out.println(x + " * " + y + " = " + calculator.multiply(x, y));
      System.out.println(x + " / " + y + " = " + calculator.divide(x, y));
    } catch (Exception e) {
      System.out.println("ERROR : " + e);
      e.printStackTrace(System.out);
    }}}
```

4. Message Passing Interface (MPI):

```
import mpi.*;
public class MPI_Scatter_Gather_Demo
{
   public static void main(String[] args) throws Exception
   {
      MPI.Init(args);
      int rank = MPI.COMM_WORLD.Rank() ;
      int size = MPI.COMM_WORLD.Size() ;
      int unitsize=4;
      int root=0;
                  int send_buffer [] = null;

                  //if (rank == root)
         //       {
                        send_buffer = new int [unitsize * size];
         //       }
                  int recv_buffer[] = new int[unitsize];
                  MPI.COMM_WORLD.Scatter(
                  /*java.lang.Object sendbuf*/ send_buffer,
                  /*int sendoffset*/                    0,
                  /*int sendcount*/                     unitsize,
                  /*Datatype sendtype*/                 MPI.INT,
                  /*java.lang.Object recvbuf*/  recv_buffer,
                  /*int recvoffset*/                    0,
                  /*int recvcount*/                     unitsize,
                  /*Datatype recvtype*/                 MPI.INT,
                  /*int root*/                          root
                  );
                  if(rank == root)
                  {
                        for(int i=0; i < (unitsize * size); i++)
                        {
                              System.out.println(send_buffer[i] + " ");
                        }
                  }
      MPI.Finalize();
   }  }
```

5. Clock Synchronization (Berkeley Algorithm):

A) Main.java:

```
package com.berkeley;

public class Main {
public static void main(String[] args) {
  // Create nodes with initial clock values
  Node node1 = new Node(1, 10);
  Node node2 = new Node(2, 15);
  Node node3 = new Node(3, 20);

  // Create the master node
  MasterNode masterNode = new MasterNode();
  masterNode.addNode(node1);
  masterNode.addNode(node2);
  masterNode.addNode(node3);

  // Synchronize clocks
  System.out.println("Before synchronization:");
  System.out.println("Node 1: " + node1.getClock());
  System.out.println("Node 2: " + node2.getClock());
  System.out.println("Node 3: " + node3.getClock());

  masterNode.synchronizeClocks();

  System.out.println("After synchronization:");
  System.out.println("Node 1: " + node1.getClock());
  System.out.println("Node 2: " + node2.getClock());
  System.out.println("Node 3: " + node3.getClock());
      }
}
```

B) Node.java:

```
package com.berkeley;
public class Node {
    private int id;
    private int clock;

    public Node(int id, int clock) {
        this.id = id;
        this.clock = clock;
    }
    public int getId() {
        return id;
    }
    public int getClock() {
        return clock;
    }
    public void setClock(int clock) {
        this.clock = clock;
    }
    public void adjustClock(int adjustment) {
        this.clock += adjustment;
    }
}
```

C) MasterNode.java:

```
package com.berkeley;

import java.util.ArrayList;
import java.util.List;

public class MasterNode {
    private List<Node> nodes;

    public MasterNode() {
        nodes = new ArrayList<>();
    }

    public void addNode(Node node) {
        nodes.add(node);
    }

    public void synchronizeClocks() {
        // Step 1: Collect clock values from all nodes
        int sum = 0;
        for (Node node : nodes) {
            sum += node.getClock();
        }

        // Step 2: Calculate the average clock value
        int average = sum / nodes.size();

        // Step 3: Send the adjustment to each node
        for (Node node : nodes) {
            int adjustment = average - node.getClock();
            node.adjustClock(adjustment);
            System.out.println("Node " + node.getId() + " adjusted
by " + adjustment);
        }
    }
}
```

6. Mutual Exclusion (Token Ring Algorithm):

A) Main.java:

```
package com.tokenring;

public class Main {
   public static void main(String[] args) {
      // Create processes
      Process process1 = new Process(1);
      Process process2 = new Process(2);
      Process process3 = new Process(3);

      // Set up the ring
      process1.setNextProcess(process2);
      process2.setNextProcess(process3);
      process3.setNextProcess(process1);

      // Start with the token at Process 1
      process1.setToken(true);

      // Start the processes
      process1.start();
      process2.start();
      process3.start();
   }
}
```

B) Process.java:

```
package com.tokenring;

public class Process extends Thread {
private int id;
private Process nextProcess;
private boolean hasToken;
private boolean inCriticalSection;

public Process(int id) {
this.id = id;
this.hasToken = false;
```

```
this.inCriticalSection = false;
}
public void setNextProcess(Process nextProcess) {
this.nextProcess = nextProcess;
}public void setToken(boolean hasToken) {
this.hasToken = hasToken;
}
@Override
public void run() {
while (true) {
if (hasToken) {
enterCriticalSection();
passToken();
}
try {
Thread.sleep(1000); // Simulate processing time
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
private void enterCriticalSection() {
inCriticalSection = true;
System.out.println("Process " + id + " is in the critical section.");
try {
Thread.sleep(2000); // Simulate critical section work
} catch (InterruptedException e) {
e.printStackTrace();
}
inCriticalSection = false;
System.out.println("Process " + id + " exited the critical section.");
}
private void passToken() {
hasToken = false;
nextProcess.setToken(true);
System.out.println("Process " + id + " passed the token to Process " + nextProcess.id);
}
}
```

7. Election Algorithms (Bully and Ring):

A) Main.java:

```
package com.election;
public class Main {
public static void main(String[] args) {
// Create processes
RingProcess process1 = new RingProcess(1);
RingProcess process2 = new RingProcess(2);
RingProcess process3 = new RingProcess(3);
// Set up the ring
process1.setNextProcess(process2);
process2.setNextProcess(process3);
process3.setNextProcess(process1);
// Start the election from Process 1
process1.startElection();
}
}
```

B) Process.java:

```
package com.election;
import java.util.ArrayList;
import java.util.List;

public class Process {
   private int id;
   private boolean isCoordinator;
   private List<Process> processes;
   public Process(int id) {
      this.id = id;
      this.isCoordinator = false;
      this.processes = new ArrayList<>();
   }
   public void addProcess(Process process) {
      processes.add(process);
   }
   public void startElection() {
      System.out.println("Process " + id + " started an election.");
      for (Process process : processes) {
         if (process.id > this.id) {
```

```
                System.out.println("Process " + id + " sent election message to Process " +
process.id);
                if (process.receiveElection(this.id)) {
                    return; // Higher process responded, stop election
                }
            }
        }
        declareVictory();
    }
    public boolean receiveElection(int senderId) {
        if (this.id > senderId) {
            System.out.println("Process " + id + " responded to Process " + senderId);
            startElection();
            return true;
        }
        return false;
    }
    public void declareVictory() {
        this.isCoordinator = true;
        System.out.println("Process " + id + " is the new coordinator.");
        for (Process process : processes) {
            process.receiveCoordinator(this.id);
        }
    }
    public void receiveCoordinator(int coordinatorId) {
        this.isCoordinator = false;
        System.out.println("Process " + id + " acknowledged Process " + coordinatorId +
" as coordinator.");
    }
}
```

C) RingProcess.java:

```
package com.election; public class RingProcess extends Thread {
private int id;
private RingProcess nextProcess;
private boolean isCoordinator;
private int[] electionMessage; public RingProcess(int id) {
this.id = id;
this.isCoordinator = false;
```

```
this.electionMessage = new int[0];
}public void setNextProcess(RingProcess nextProcess) {
this.nextProcess = nextProcess;
}
public void startElection() {
System.out.println("Process " + id + " started an election.");
electionMessage = new int[] { id };
nextProcess.receiveElection(electionMessage);
}
public void receiveElection(int[] message) {
if (message.length == 0) {
declareVictory();
return;
}
int maxId = message[0];
for (int id : message) {
if (id > maxId) maxId = id;
}
if (maxId == this.id) {
declareVictory();
} else {
int[] newMessage = new int[message.length + 1];
System.arraycopy(message, 0, newMessage, 0, message.length);
newMessage[message.length] = this.id;
nextProcess.receiveElection(newMessage);
} }
public void declareVictory() {
this.isCoordinator = true;
System.out.println("Process " + id + " is the new coordinator.");
nextProcess.receiveCoordinator(this.id);
}
public void receiveCoordinator(int coordinatorId) {
this.isCoordinator = false;
System.out.println("Process " + id + " acknowledged Process " + coordinatorId + " as
coordinator.");
if (coordinatorId != this.id) {
nextProcess.receiveCoordinator(coordinatorId);
} } }
```

8. Web Services:

A) CalculatorService.java:

```
package com.webservice;
import javax.jws.WebMethod;
import javax.jws.WebService;
@WebService
public interface CalculatorService {
   @WebMethod
   double add(double x, double y);
   @WebMethod
   double subtract(double x, double y);
   @WebMethod
   double multiply(double x, double y);
   @WebMethod
   double divide(double x, double y);
}
```

B) CalculatorServiceImpl.java:

```
package com.webservice;
import javax.jws.WebService;
@WebService(endpointInterface = "com.webservice.CalculatorService")
public class CalculatorServiceImpl implements CalculatorService {
   public double add(double x, double y) {
      return x + y;
   }
   public double subtract(double x, double y) {
      return x - y;
   }
   public double multiply(double x, double y) {
      return x * y;
   }
   public double divide(double x, double y) {
      if (y == 0) throw new ArithmeticException("Division by zero");
      return x / y;
   }
}
```

C) CalculatorServicePublisher.java:

```
package com.webservice;
import javax.xml.ws.Endpoint;
public class CalculatorServicePublisher {
   public static void main(String[] args) {
      Endpoint.publish("http://localhost:8080/ws/calculator", new
CalculatorServiceImpl());
      System.out.println("Service is published at
http://localhost:8080/ws/calculator");
   }
}
```

D) CalculatorClient.java:

```
package com.webservice;

public class CalculatorClient {
   public static void main(String[] args) {
      CalculatorServiceImplService service = new CalculatorServiceImplService();
      CalculatorService calculator = service.getCalculatorServiceImplPort();

      System.out.println("5 + 3 = " + calculator.add(5, 3));
      System.out.println("5 - 3 = " + calculator.subtract(5, 3));
      System.out.println("5 * 3 = " + calculator.multiply(5, 3));
      System.out.println("5 / 3 = " + calculator.divide(5, 3));
   }
}
```

9. Mini Project:

A) SocketServer.java:

```
package com.miniproject.socket;
import java.io.*;
import java.net.*;
public class SocketServer {
   public static void main(String[] args) {
      try (ServerSocket serverSocket = new ServerSocket(5000)) {
         System.out.println("Socket Server is running on port 5000...");
         while (true) {
            Socket socket = serverSocket.accept();
            new ClientHandler(socket).start();
         }
      } catch (IOException e) {
         e.printStackTrace();
      } } }
class ClientHandler extends Thread {
   private Socket socket;
   public ClientHandler(Socket socket) {
      this.socket = socket;
   }
   public void run() {
      try (BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
          PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {
         String inputLine;
         while ((inputLine = in.readLine()) != null) {
            String[] tokens = inputLine.split(" ");
            double x = Double.parseDouble(tokens[0]);
            double y = Double.parseDouble(tokens[2]);
            String operation = tokens[1];
            double result = 0;
            switch (operation) {
               case "+":
                  result = x + y;
                  break;
               case "-":
                  result = x - y;
```

```
                    break;
                default:
                    out.println("Invalid operation");
                    return;
            }
            out.println("Result: " + result);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } } }
```

B) Calculator.java:

```
package com.miniproject.rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculator extends Remote {
    double multiply(double x, double y) throws RemoteException;
    double divide(double x, double y) throws RemoteException;
}
```

C) CalculatorImpl.java:

```
package com.miniproject.rmi;

import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class CalculatorImpl extends UnicastRemoteObject implements Calculator {
    protected CalculatorImpl() throws RemoteException {
        super();
    }

    public double multiply(double x, double y) throws RemoteException {
        return x * y;
    }

    public double divide(double x, double y) throws RemoteException {
        if (y == 0) throw new RemoteException("Division by zero");
        return x / y;
```

```
    }
}
```

D) RMIServer.java:

```
package com.miniproject.rmi;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class RMIServer {
    public static void main(String[] args) {
        try {
            CalculatorImpl calculator = new CalculatorImpl();
            Registry registry = LocateRegistry.createRegistry(1099);
            registry.rebind("Calculator", calculator);
            System.out.println("RMI Server is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

E) CalculatorService.java:

```
package com.miniproject.webservice;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public interface CalculatorService {
    @WebMethod
    double add(double x, double y);

    @WebMethod
    double subtract(double x, double y);
}
```

F) CalculatorServiceImpl.java:

```
package com.miniproject.webservice;
```

```
import javax.jws.WebService;

@WebService(endpointInterface = "com.miniproject.webservice.CalculatorService")
public class CalculatorServiceImpl implements CalculatorService {
  public double add(double x, double y) {
    return x + y;
  }

  public double subtract(double x, double y) {
    return x - y;
  }
}
```

G) CalculatorServicePublisher.java:

```
package com.miniproject.webservice;

import javax.xml.ws.Endpoint;

public class CalculatorServicePublisher {
  public static void main(String[] args) {
    Endpoint.publish("http://localhost:8080/ws/calculator", new
CalculatorServiceImpl());
    System.out.println("Web Service is published at
http://localhost:8080/ws/calculator");
  }
}
```

H) Client.java:

```
package com.miniproject;

import com.miniproject.rmi.Calculator;
import com.miniproject.rmi.CalculatorImpl;
import com.miniproject.webservice.CalculatorService;
import com.miniproject.webservice.CalculatorServiceImplService;

import java.io.*;
import java.net.Socket;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
```

```
public class Client {
   public static void main(String[] args) {
      try {
         // Socket Client
         Socket socket = new Socket("localhost", 5000);
         BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

         out.println("5 + 3");
         System.out.println("Socket Result: " + in.readLine());

         out.println("5 - 3");
         System.out.println("Socket Result: " + in.readLine());

         socket.close();

         // RMI Client
         Registry registry = LocateRegistry.getRegistry("localhost", 1099);
         Calculator calculator = (Calculator) registry.lookup("Calculator");
         System.out.println("RMI Result (5 * 3): " + calculator.multiply(5, 3));
         System.out.println("RMI Result (5 / 3): " + calculator.divide(5, 3));

         // Web Service Client
         CalculatorService calculatorService = new
CalculatorServiceImplService().getCalculatorServiceImplPort();
         System.out.println("Web Service Result (5 + 3): " + calculatorService.add(5,
3));
         System.out.println("Web Service Result (5 - 3): " +
calculatorService.subtract(5, 3));
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```