The client–server is a structure that partitions tasks or workloads between the providers of a resource or service (servers), and service requesters (clients). Often clients and servers communicate over a computer network. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function.

To archive client-server connection first of all we need to create socked with the same port number both on client and server. Client also needs information about location of the server. After that client and server have to do “hand shake”. To archive this we are calling method “accept” at server socket that method will listen to connections and block until connection will be made. When this is done we can use ObjectOutputStream and ObjectInputStream to send serializable objects. If we are expecting more than one client we should create a threat that will repeat whole procedure for every connections.

This is an example for presentation:

**– Client sending a Serializable object**

Socket s = new Socket(HOST, PORT);

ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());

out.writeObject(new Message("message"));

**– Server receiving a Serializable object**

ServerSocket welcomeSocket = new ServerSocket(PORT);

Socket s = welcomeSocket.accept();

ObjectInputStream in = new ObjectInputStream(s.getInputStream());

Message data = (Message)in.readObject();

public class ServerController implements Runnable {

private static final int PORT = 6666;

This is our implementation:

public ServerController() {

}

public void run() {

int count = 1;

try {

ServerSocket welcomeSocket = new ServerSocket(PORT);

System.out.println("Server started");

DummyObserver observer = new DummyObserver();

while (true) {

Socket connectionSocket = welcomeSocket.accept();

ServerCommunication c = new ServerCommunication(connectionSocket, observer);

new Thread(c, "Communication " + count).start();

System.out.println("Client connected");

count++;

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

public class ConnectionController implements Runnable{

private static final String HOST = "localhost";

private static final int PORT = 6666;

private ObjectInputStream inFromServer;

private ObjectOutputStream outToServer;

private Socket socket;

public ConnectionController() throws IOException

{

try

{

socket = new Socket(HOST, PORT);

outToServer = new ObjectOutputStream(socket.getOutputStream());

inFromServer = new ObjectInputStream(socket.getInputStream());

ClientReceiver reciever = new ClientReceiver(inFromServer, outToServer);

new Thread(reciever, "Reciever").start();

}

catch (Exception e)

{

e.printStackTrace();

}

}

public void sendDatesToServer(LocalDate from, LocalDate to) {

LocalDate[] dates = {from, to};

try {

outToServer.writeObject(dates);

} catch (IOException e) {

//

}

}

public void run()

{

}

}

public class ClientReceiver implements Runnable{

private ObjectInputStream inFromServer;

private ObjectOutputStream outToServer;

public ClientReceiver(ObjectInputStream inFromServer,ObjectOutputStream outToServer ) {

this.inFromServer=inFromServer;

}

public void run() {

while (Main.controller == null) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

while (true) {

try {

ProxyTripList trips = (ProxyTripList) inFromServer.readObject();

System.out.println(trips.getSize());

Main.controller.showList(trips);

} catch (IOException e) {

} catch (ClassNotFoundException e) {

//

}

}

}

}