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. Socked is one of the endpoint of link between two programs. 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 class ServerController implements Runnable {

private static final int PORT = 6666;

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();

count++;

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

public class ServerCommunication implements Runnable {

private ObjectInputStream inFromClient;

private ObjectOutputStream outToClient;

public ServerCommunication(Socket clientSocket, DummyObserver observer)

throws IOException

{

outToClient = new ObjectOutputStream(clientSocket.getOutputStream());

inFromClient = new ObjectInputStream(clientSocket.getInputStream());

observer.addClient(outToClient);

}

public void run() {

try {

outToClient.writeObject(DataHandler.getInstance().getTrips());

outToClient.reset();

} catch (IOException e) {

e.printStackTrace();

}

while (true) {

try {

LocalDate[] dates = (LocalDate[]) inFromClient.readObject();

DataHandler.getInstance().getInDates(dates, outToClient);

inFromClient.reset();

} catch (IOException e) {

//

} catch (ClassNotFoundException e) {

//

}

}

}

}

public class ConnectionController {

private static final String HOST = "localhost";

private static final int PORT = 6666;

private ObjectOutputStream outToServer;

public ConnectionController() throws IOException {

try {

Socket socket = new Socket(HOST, PORT);

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

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

ClientReceiver reciever = new ClientReceiver(inFromServer);

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 class ClientReceiver implements Runnable {

private ObjectInputStream inFromServer;

public ClientReceiver(ObjectInputStream inFromServer) {

this.inFromServer = inFromServer;

}

public void run() {

while (true) {

try {

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

Main.controller.showList(trips);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

}

“ServerController” class is responsible for listening for clients connection request. That class is creating new “ServerCommunication” thread every time that new client is connected and then waiting for new connection again. This class implements “Runnable” because it’s task is infinite and we want to run it in another thread. “ServerCommunication” class is sending trips to client and receives dates. “ConnectionController” class is requesting server for connection and then creating “Reciever” thread. “ClientReciver” class is receiving trips that “ServerCommunication” send.