How to create Chat Client/Server solution in Java

With the help of this tutorial, we will create a Client/Server chat application using Java.

Like (5) (1)

Services
Add a comment
Add to favorites
Mark as viewed
Add a note

+Java

How to start a network session

In order to start a network session, a server and a client are required , and a server is set up to listen at a given port. The server waits and does nothing until a client attempts to connect

Listing 1 : The Server code (Server.java):

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

class Server {
   public static void main(String args[]) {
      String data = "Toobie ornaught toobie";
      try {
         ServerConnection srvr = new ServerConnection(
         Connection skt = srvr.accept();
         System.out.print("Server has connected!
");
         PrintWriter out = new PrintWriter(skt.getOut
         System.out.print("Sending string: '" + data
");
         out.print(data);
         out.close();
         skt.close();
         srvr.close();
      }
      catch(Exception e) {
         System.out.print("Whoops! It didn't work!
");
      }
   }
}
```

The key portions of this program are in the try{} block. The ServerConnection instantiation is what sets up the server to listen at the given port. The server is automatically set up at the computer on which it is run. The Connection instantiation on the next line uses the accept() method of ServerConnection. This method waits until a client attempts to connect to the server, and it returns an instance of the Connection class. This Connection instance (skt) is now the route through which we can communicate with the client. skt.getOutputStream() returns the output stream through which the server can talk to the client, and skt.getInputStream() returns the input stream through which the server can hear the client.

This example creates a PrintWriter instance using the output stream for easier output and sends the data (stored in data) to the client (out.print(data);). After everything is done, all the streams and Connections should be closed before the program is exited. Now, let's see the client code.

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

class Client {
   public static void main(String args[]) {
      try {
         Connection skt = new Connection("localhost",
         BufferedReader in = new BufferedReader(new
            InputStreamReader(skt.getInputStream()));
         System.out.print("Received string: '");

         while (!in.ready()) {}
         System.out.println(in.readLine()); // Read or

         System.out.print("'
");
         in.close();
      }
      catch(Exception e) {
         System.out.print("Whoops! It didn't work!
");
      }
   }
}
```

The base of the program is in the try{} block. A connection to the server is attempted through the instantiation of the Connection class. It attempts to contact the server at localhost through port 1234 - the same port where the server is listening. Once the Connection is at hand, it works exactly the same as the one obtained through the ServerConnection class in Server.java. This time, the input stream is obtained and a BufferedReader is instantiated using it. The data is read from this stream and displayed to the screen.

Introduction to Swing Programming

Swing is a rapid GUI development tool that is part of the standard Java development kit. It was primarily developed due to the shortcomings of the Abstract Windows Toolkit (AWT). For example, Swing's JButton class enhances the AWT Button class to allow not only text, but images on the button. In addition, all Swing components support assistive technologies.

The below lists few of the basic Swing components:

- JFrame
- JPanel
- JButton

If you know how to use these basic components, using the others is simple. Usually, while making a GUI-based application, you instantiate a JFrame and choose its layout. Then you put one or more JPanels in the JFrame if you want to. JPanels also have different layout options like JFrames do. After that, you add other components. The following is the code corresponding to what we just mentioned. It can also be found in SimpleGui1.java.

Listing 3: The code SimpleGui1.java

```
Import java.lang.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class SimpleGui1 {
   public static void main(String args[]) {
      // Let's make a button first
      JButton btn = new JButton("Click Me!");
      btn.setMnemonic(KeyEvent.VK_C); // Now you can h

      // Let's make the panel with a flow layout.
      // Flow layout allows the components to be
      // their preferred size.
      JPanel pane = new JPanel(new FlowLayout());
      pane.add(btn);  // Add the button to the pane

      // Now for the frame
      JFrame fr = new JFrame();
      fr.setContentPane(pane);  // Use our pane as the
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
      fr.setLocation(200, 200); // located at (200, 20
      fr.pack();                // Frame is ready. Pac
      fr.setVisible(true);      // Make it visible
   }
}
```

Event Handlers

In addition to what we have so far , functionality can be added triggered and their appropriate methods (event handlers) are invoked. To see how this works, let's modify the example above:

Listing 4: Modifying the code with Event Handlers

```
import java.lang.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class SimpleGui2 {
   private static JFrame fr = null;
   private static JButton btn = null;

   public static void main(String args[]) {
      // Let's make a button first
      btn = new JButton("Click Me!");
      btn.setMnemonic(KeyEvent.VK_C); // Now you can h
      btn.addActionListener(new ButtonListener()); //

      // Let's make the panel with a flow layout.
      // Flow layout allows the components to be
      // their preferred size.
      JPanel pane = new JPanel(new FlowLayout());
      pane.add(btn);  // Add the button to the pane

      // Now for the frame
      fr = new JFrame();
      fr.setContentPane(pane);  // Use our pane as the
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
      fr.setLocation(200, 200); // located at (200, 20
      fr.pack();                // Frame is ready. Pac
      fr.setVisible(true);      // Make it visible
   }

   // Button event handler
   static class ButtonListener implements ActionLister
      public void actionPerformed(ActionEvent e) {
         btn.setEnabled(false); // Disable the button
      }
   }
}
```

Now this brings up the same window that the first one did. However, if you click the button, it disables itself:

To see something a lot more complex, refer to the design for the Client/Server chat program: Gui.java. The GUI looks like this:

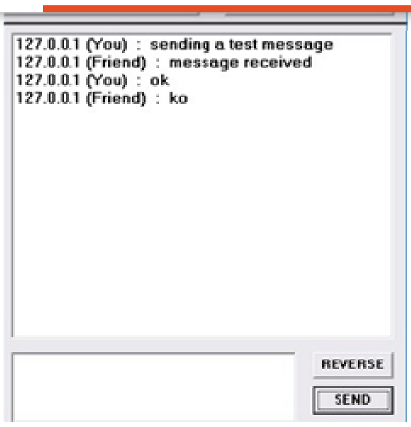
127.0.0.1 (You) : sending a test message
127.0.0.1 (Friend) : message received
127.0.0.1 (You) : ok
127.0.0.1 (Friend) : ko

Figure 1: Chat Image

Try clicking Connect and Disconnect buttons and see what happens.

Thread Issues

All AWT classes are thread safe whereas Swing's classes are not. A little knowledge about threads is necessary to completely understand the reason for the complexity of using threads. Consider an object that has been instantiated in one thread. Now this object needs to be modified by two or more threads. Data can easily be corrupted if two threads try to modify the same object at the same time. There is no such thing as "the same time" when it comes to threads since the operating system switches between the various threads to make them seem like they are running concurrently. However, if one thread is interrupted while it is modifying an object and a second thread is initiated so that it begins to modify that object, data can get corrupted.

There are a couple of ways to get around this problem in Java. Java allows a way to enforce that no thread modifies the same object that your thread is modifying through the use of the synchronize keyword.

Synchronizing an object before modifying it blocks any other threads attempting to access the synchronized object until your synchronize block is done. AWT's methods synchronize on their corresponding object instantiation, which prevents the objects (eg. buttons, panels, etc.) on the screen from being corrupted while they are being drawn. However, this can block your thread if it is trying to access the AWT object.

CHANNELS ▼ COURSES ▼ LOGIN

We learned how to create a Client/Server Chat Application using Java . See you next time.

Vinay Kumar

Software developer with more than 5 years of development on Java, HTML, CSS.

What did you think of this post?
Like (5) (1)

Add your comment

Robert See
Hello Vinay Kumar, Thanks for sharing. I do learn something from this article. I own a website and I develop it by myself. I need to add a jave chat client into my website, I mean integrated the into my site, to make the chat can share my site user info. And I download 123 flash chat, a chat software from this one(www.123flashchat.com) ; a same java server based product; I want to learn by using their free version it is java based and flash client focused. Can anyone share some experience?
[+1 year ago] Answer it

Mr.Bool Editor
You can follow this article and create your own chat client, what do you think ?
6/27/2014 2:54pm Answer it

With your help, we can keep providing free content and helping you to be a better professional
support us
2° we develop
1° you plant
3° and we grow together.
and start all over again, because learning never ends.

Know how to keep MrBool Online SUPPORT US

Post

Implementing Creational Patterns in Java

Post

How to Create Rich Web applications using Spring MVC?

Post

JSF Composite: Applying Reusability of code

Post

Restoring Security in Tomcat with BadInputFilter

Post

Introduction to Big Data in Java with Apache Spark

Post

Java EE: Working with Lifecycle and @Alternative Beans Annotations

Post

REST Server with Spring Data, Spring Boot and PostgreSQL

Show more