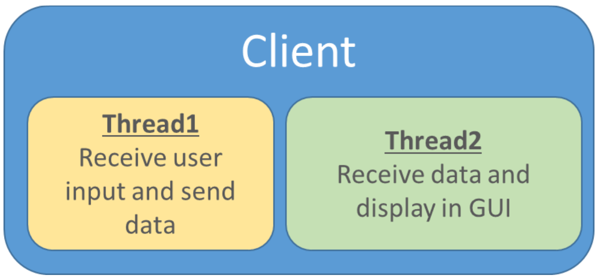
COMP3015
Data Communication and Networking
Multithreaded Client

Client with Multiple Threads

A client-side program is required to handle data transmission and user interaction. We need multiple threads so that the tasks can be handled concurrently.


Client
Thread1
Receive user input and send data
Thread2
Receive data and display in GUI

Example: GUI Echo Client

The GUI echo client has a text field for getting user inputs. It has another thread for receiving data sent by the server.

Establishing Connection & Creating New Thread for Data Receiving

```
public class GUIEchoClient extends JFrame {
   DataOutputStream out;
   ...

   public GUIEchoClient(String serverIP, int port) throws IOException {

      Socket socket = new Socket(serverIP, port);
      out = new DataOutputStream(socket.getOutputStream());
      /*
       * Run a thread behind to receive data
       */
      Thread t = new Thread(() -> {
         receiveData(socket);
      });
      t.start();
      ...
   }
   ...
}
```

When we create an instance of the **GUIEchoClient** class, we also establish a connection to the server and use a new thread to handle data receiving.

```
Thread t = new Thread(() -> {
   receiveData(socket);
});
t.start();
```

The code segment above can be replaced with the following line:

```
new Thread(()->receiveData(socket)).start();
```

User Interface Creation

On the other hand, we need to create a GUI for receiving user inputs and displaying data sent by the server.

```
public class GUIEchoClient extends JFrame {

   JTextArea textArea;
   JTextField textField;
   ...

   public GUIEchoClient(String serverIP, int port) throws IOException {
      ...

      Container container = this.getContentPane();
      container.setLayout(new BorderLayout());

      textArea = new JTextArea();
      textArea.setEditable(false);
      textArea.setBackground(Color.lightGray);

      JScrollPane sp = new JScrollPane(textArea);

      textField = new JTextField();
      ...

      container.add(sp, BorderLayout.CENTER);
      container.add(textField, BorderLayout.SOUTH);

      this.setTitle("Echo Client");
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.setSize(new Dimension(320, 240));
      this.setVisible(true);

   }
   ...
}
```

Adding Key Listener

For sending data, we can do it in the GUI thread directly. For example, when the user types a message in the text field and presses **ENTER**, the **sendMsg()** method will be invoked and it sends the message out through the output stream of the socket.

So, we need to add a key listener to the text field to capture the keystroke of the **ENTER** key.

```java
textField.addKeyListener(new KeyListener() {

   @Override
   public void keyTyped(KeyEvent e) {}

   @Override
   public void keyPressed(KeyEvent e) {}

   @Override
   public void keyReleased(KeyEvent e) {
      if (e.getKeyCode() == 10) {
         sendMsg(textField.getText());
         textField.setText("");
      }
   }
});
```

Receiving and Displaying Message

Receiving data is simple here. But, we need to pay attention to the GUI update. Note that the **receiveData()** method is executed in a thread, not the thread managing the GUI. We cannot directly run the **append()** method of the text area to update its content because the text area may be locked by the GUI thread for rendering (refresh the display).

Instead, we need **SwingUtilities.invokeLater()** method to help us to run the statement later when the text area is ready for the update.

```java
private void receiveData(Socket socket) {
   try {
      byte[] buffer = new byte[1024];
      DataInputStream in = new DataInputStream(socket.getInputStream());
      while (true) {
         int len = in.readInt();
         in.read(buffer, 0, len);

         SwingUtilities.invokeLater(() -> {
            textArea.append(new String(buffer, 0, len) + "\n");
         });
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

Sending Message

It is simple and just like what we discussed in TCP labs.

```
private void sendMsg(String msg) {
   try {
      out.writeInt(msg.length());
      out.write(msg.getBytes(), 0, msg.length());
   } catch (IOException e) {
      textArea.append("Unable to send message to the server!\n");
   }
}
```

Exploring

You can find the complete version of the client-side program and server-side program at the following URL:

https://www.comp.hkbu.edu.hk/~mandel/comp3015/lab6/

You can test the programs by following the procedure below:

1. Compile java files:

```
javac *.java
```

2. Run the server-side program:

```
java EchoServer2
```

3. Run the client-side program:

```
java GUIEchoClient 127.0.0.1 12345
```

(Change the IP address if you run the server-side program on another computer)