Advanced Object Oriented Programming
Paint Mini-Project #3
Sockets Calculator

Computer Science L2
Group G6:
Shoykyat Sharafyabi
Nadir Abdullayev
Kanan Mikayilov

24 April, 2020

Title:

The calculator mini project using Sockets.

Introduction:

When the program launches, firstly it creates a different thread for the server and launches it there. On the main thread, the GUI and the logic start to work. The calculator itself acts as a client since it sends expressions to the server to solve and waits for the response.

When any digit or operation is clicked, it gets added to the expression and the display updates to show the new expression. When AC is clicked, it simply cleans out the expression from the memory. When C is clicked, it removes the last character from the expression.

When = gets clicked, the calculator sends the expression to the server and when it gets the response, it saves it in the expression itself and shows it in the display.

When the calculator is initialized, it tries to connect to the server, if there's a problem, then it shows it on the display and asks you to click on red R button to reconnect.

Connection will always be with client and server, server has to wait for a client's request and the server at the same time has to accept and establish the connection.

So, there were 2 main packages and 1 main class created:

- CalcGUI
- Server

- Main class

Packages:

CalcGUI package consists of 3 classes:

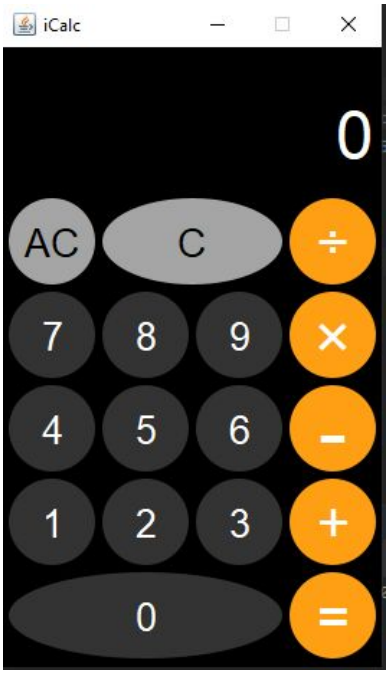
- Calc
- OpButton
- RoundButton

The main functional class which is responsible for GUI part of calculator project is Calc class. The other 2 classes serve to add some visual changes.

Calc class:

The calculator graphical user interface is a Jframe which consists of JPanel, JTextField and extends RoundButtons and OpButtons from the same CalcGUI package.
It has init() function which builds the gui of calculator (some basic stuff) in order to get the following result.

```
public class Calc implements ActionListener {
    JFrame f;
    JTextField display;
    JPanel p;
    RoundButton b1,b2,b3,b4,b5,b6,b7,b8,b9,b0,beq,bdel,bclr;
    OpButton bdiv, bmul, bsub, badd;

    static String expression = "0";

    private boolean connected = false;
    private Socket s;
    private PrintWriter pw;
    private BufferedReader br;

    public void init () {
        //----------------------1ROW---------------------------------
        display = new JTextField("0");
        display.setBounds( x: 0, y: 20, width: 260, height: 80);
        display.setBackground(Color.black);
        display.setForeground(Color.white);
        display.setHorizontalAlignment(JLabel.RIGHT);
        display.setBorder(null);
        display.setFont(new Font( name: "Helvetica", style: 0, size: 47));
        display.setCaretColor(Color.BLACK);
        display.setEditable(false);
```



Except of the functions which serves to build gui, there are more core functions, one of which is actionPerformed which is overriden from extended ActionListener class.

actionPerformed():

```
public void actionPerformed(ActionEvent e) {
    JButton source = (JButton) e.getSource();

    if (!connected) {
        if (source == bdel) {
            connectToServer();
        }
        return;
    }

    // Equals button
    if (source == beq) {
        askServerForAnswer();
    }

    // numbers or the dot
    if (source == b0 || source == b1 || source == b2 || source == b3
            || source == b4 || source == b5 || source == b6
            || source == b7 || source == b8 || source == b9)
        addToExpression(source.getText());
```

```
// operations
if (source == badd) {
    addToExpression( s: " + ");
} else if (source == bsub) {
    addToExpression( s: " - ");
} else if(source == bmul) {
    addToExpression( s: " * ");
} else if(source == bdiv) {
    addToExpression( s: " / ");
}

// AC button
if (source == bclr) {
    expression = "";

// C button
} else if (source == bdel) {
    if (!(expression.length() == 1 && expression.charAt(0) == '0'))
        expression = expression.substring(0, expression.length()-1);
    if (expression.length() == 0)
        expression = "0";
}

updateDisplay();
```

First it connects to server, then it listens to users actions:

- when user clicks the digits → it adds this digit to a string which will be sent to server at the end
- when user clicks to AC → it clears textfield
- when user clicks to C → it removes the last digit
- when user clicks to = → it calls askServerForAnswer() → which sends the expression to server for further evaluation and respond to textfield

The function which serves to connect the GUI calculator to server in order to get the answer of expression. This kind of connections should be in try{..}catch(){} to catch the exceptions

```
private void connectToServer() {
    try {
        s = new Socket(InetAddress.getLocalHost(), Main.PORT);
        dout = new DataOutputStream(s.getOutputStream());
        din = new DataInputStream(s.getInputStream());
        noProblemConnecting();
    } catch (Exception e) {
        problemConnecting();
        e.printStackTrace();
    }
    updateDisplay();
}
```

Another core function is askServerForAnswer(), which takes the expression and sends it to connected by socket server and waits for respond and after undated textfield

```
private void askServerForAnswer() {
    System.out.printf("CLIENT: Server call to find %s\n", expression);

    // server call here
    try {
        dout.writeUTF(expression);
        dout.flush();
        expression = din.readUTF();
    } catch (Exception e) {
        problemConnecting();
        e.printStackTrace();
    }
    updateDisplay();
}
```

The server package:

This package is responsible for server side of this project and consists of 3 classes, with main class named server:

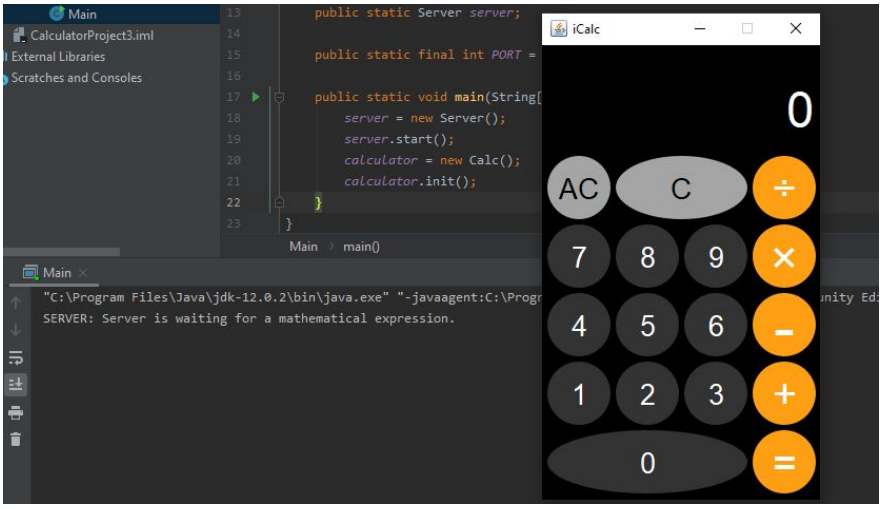
There are 2 classes which serve to take complicated mathematical expressions and solves them in right order.

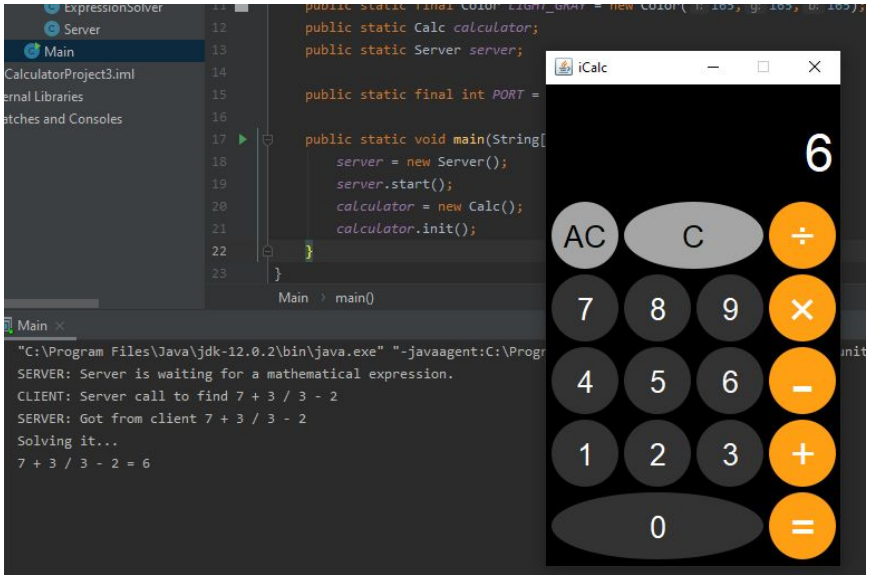
Server class:

Server class waits for clients and takes the expression, then it solves this expression before sending it to the client. Server takes the expressions until user won't exit from program The server can be only one, but the number of clients vary, there can be a lot of clients, each time user enters expression and clicks equal to calculate, new client is connecting to the server

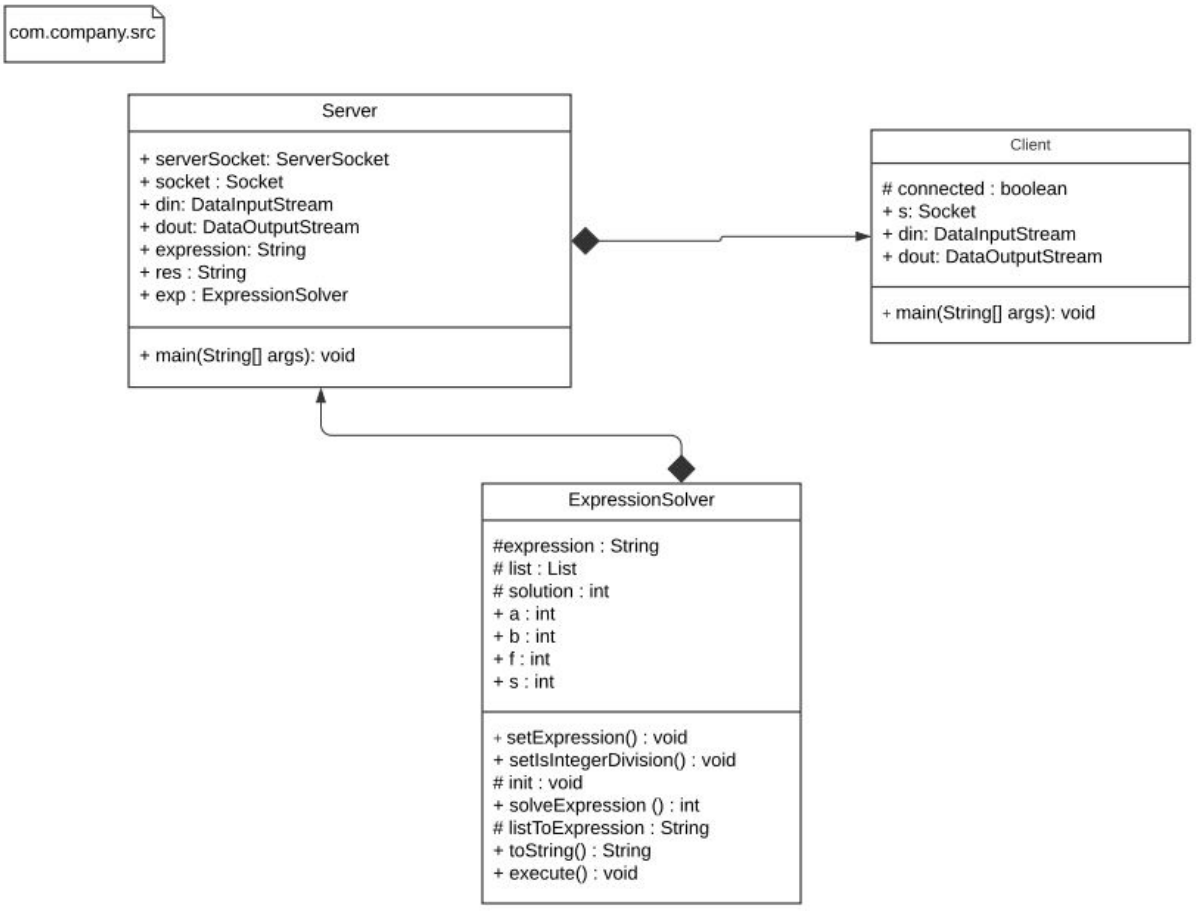
```
lic class Server extends Thread {
    public void run() {
        ServerSocket serverSocket;
        Socket socket;
        DataInputStream din;
        DataOutputStream dout;
        ExpressionSolver exp = new ExpressionSolver();
        String expression;
        String res;

        try {
            serverSocket = new ServerSocket(Main.PORT);
            socket = serverSocket.accept();
            din = new DataInputStream(socket.getInputStream());
            dout = new DataOutputStream(socket.getOutputStream());
            System.out.println("SERVER: Server is waiting for a mathematical expression.");
            while(true) {
                expression = din.readUTF();
                if (expression.equals("EXIT")) break;
                System.out.println("SERVER: Got from client " + expression + "\nSolving it...");
                exp.setExpression(expression);
                res = exp.solveExpression();
                System.out.println(exp);
                dout.writeUTF(res);
                dout.flush();
            }
            System.out.println("Server has finished its work.");
            din.close();
            dout.close();
            socket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Result:





UML:


com.company.src
Server
+ serverSocket: ServerSocket
+ socket : Socket
+ din: DataInputStream
+ dout: DataOutputStream
+ expression: String
+ res : String
+ exp : ExpressionSolver
+ main(String[] args): void
Client
connected : boolean
+ s: Socket
+ din: DataInputStream
+ dout: DataOutputStream
+ main(String[] args): void
ExpressionSolver
#expression : String
list : List
solution : int
+ a : int
+ b : int
+ f : int
+ s : int
+ setExpression() : void
+ setIsIntegerDivision() : void
# init : void
+ solveExpression () : int
# listToExpression : String
+ toString() : String
+ execute() : void