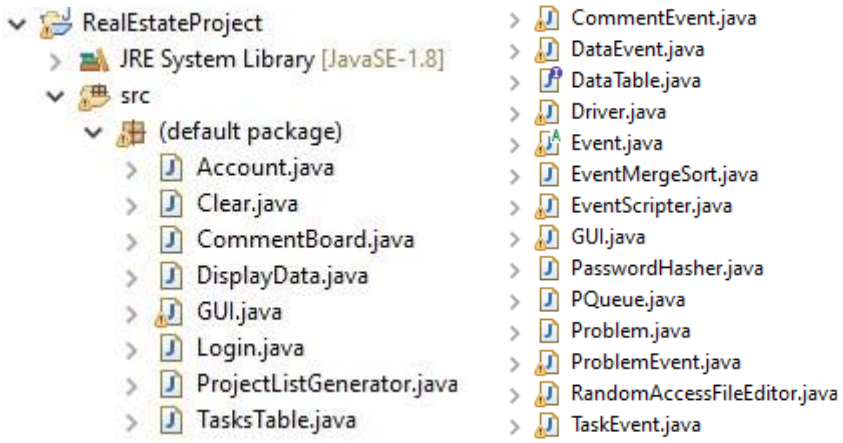
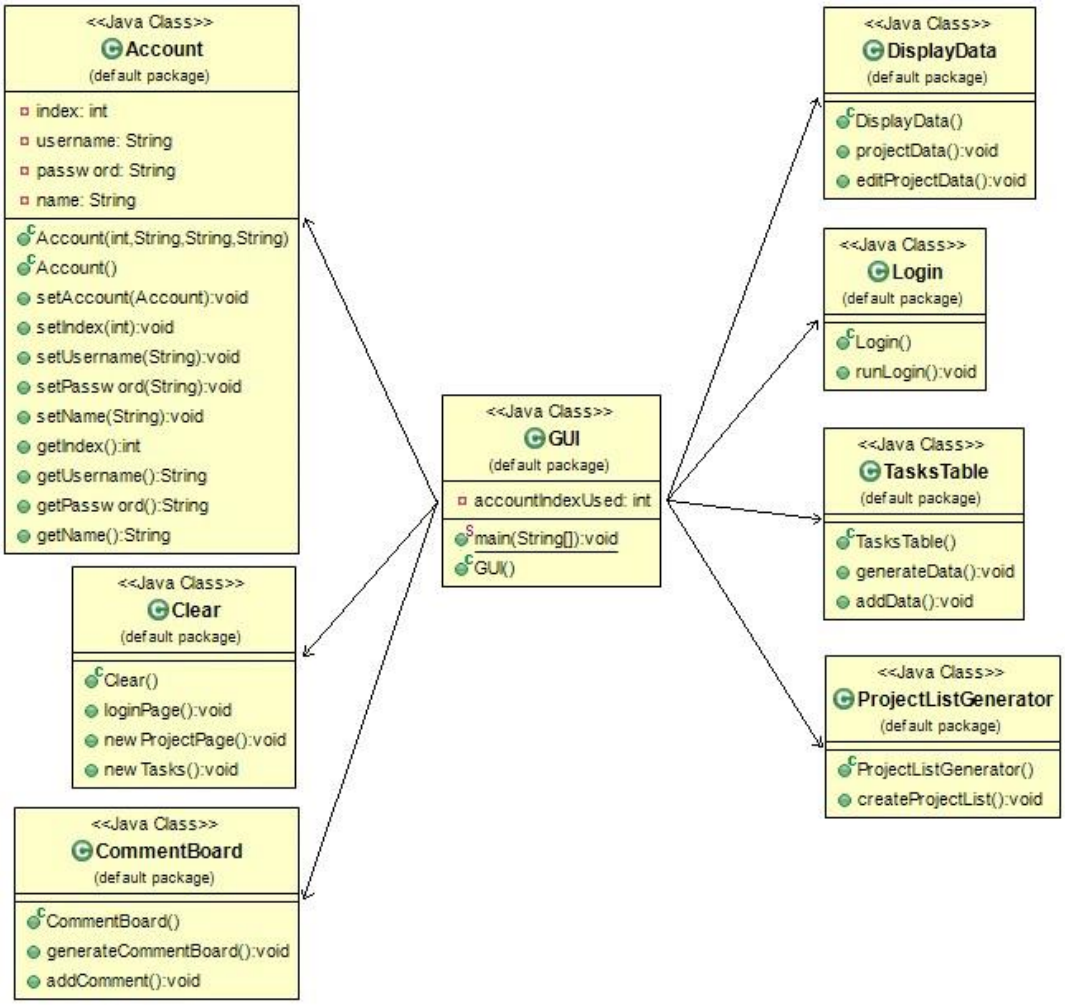
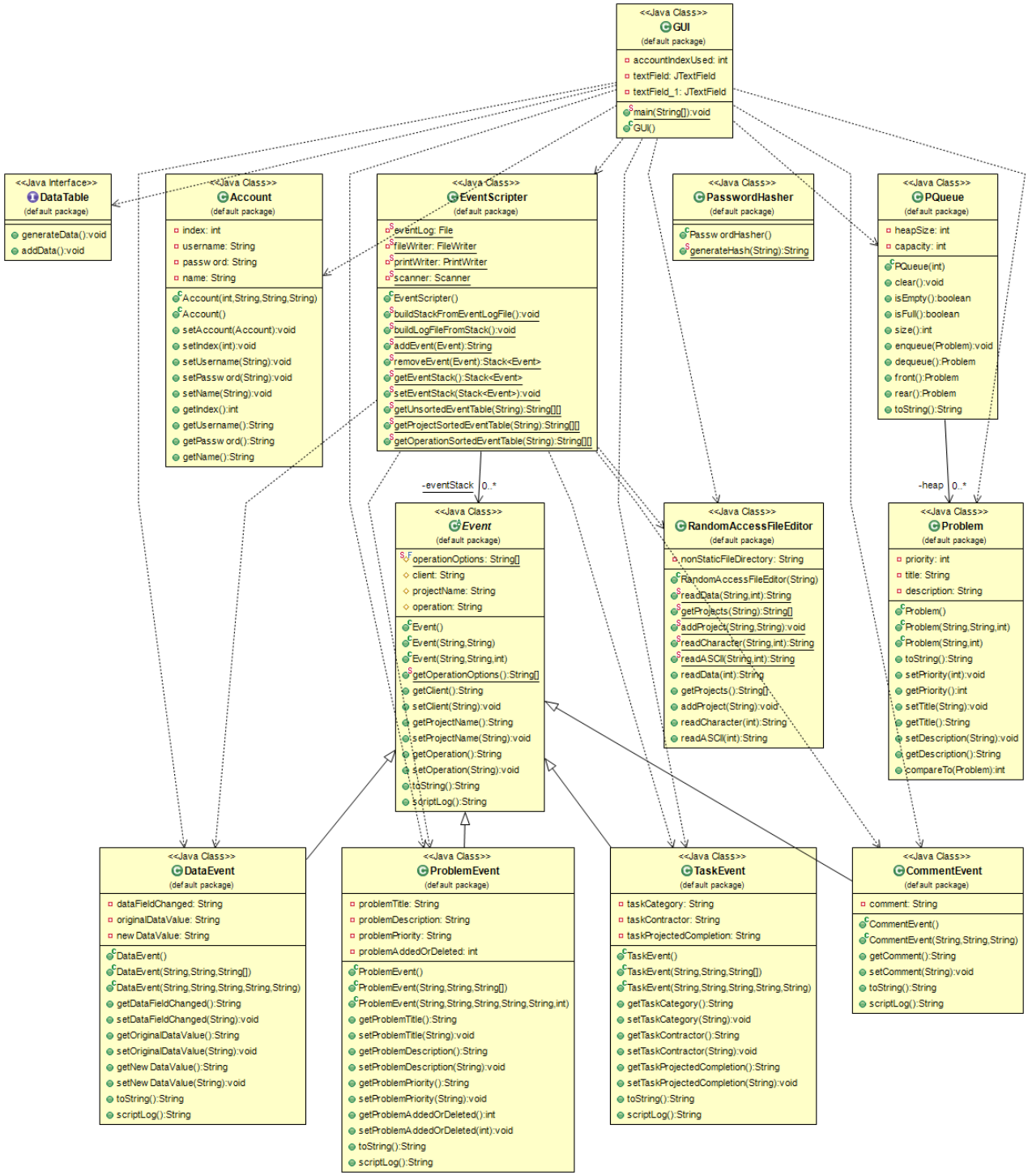
Criterion C: Development

All Classes



Clear, CommentBoard, DisplayData, Login, ProjectListGenerator, and TasksTable are nested classes inside the GUI class because their methods heavily involve GUI JComponents.



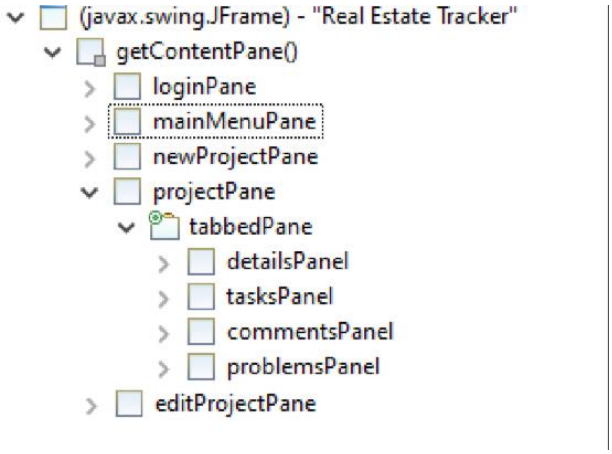

<<Java Class>>
GUI
(default package)
accountIndexUsed: int
textField: JTextField
textField_1: JTextField
main(String[]):void
GUI()
<<Java Interface>>
DataTable
(default package)
generateData():void
addData():void
<<Java Class>>
Account
(default package)
index: int
username: String
passw ord: String
name: String
Account(int,String,String,String)
Account()
setAccount(Account):void
setIndex(int):void
setUsername(String):void
setPassw ord(String):void
setName(String):void
getIndex():int
getUsername():String
getPassw ord():String
getName():String
<<Java Class>>
EventScripter
(default package)
eventLog: File
fileWriter: FileWriter
printWriter: PrintWriter
scanner: Scanner
EventScripter()
buildStackFromEventLogFile():void
buildLogFileFromStack():void
addEvent(Event):String
removeEvent(Event):Stack<Event>
getEventStack():Stack<Event>
setEventStack(Stack<Event>):void
getUnsortedEventTable(String):String[][]
getProjectSortedEventTable(String):String[][]
getOperationSortedEventTable(String):String[][]
<<Java Class>>
PasswordHasher
(default package)
Passw ordHasher()
generateHash(String):String
<<Java Class>>
PQueue
(default package)
heapSize: int
capacity: int
PQueue(int)
clear():void
isEmpty():boolean
isFull():boolean
size():int
enqueue(Problem):void
dequeue():Problem
front():Problem
rear():Problem
toString():String
-eventStack
0..*
-heap
0..*
<<Java Class>>
Event
(default package)
operationOptions: String[]
client: String
projectName: String
operation: String
Event()
Event(String,String)
Event(String,String,int)
getOperationOptions():String[]
getClient():String
setClient(String):void
getProjectName():String
setProjectName(String):void
getOperation():String
setOperation(String):void
toString():String
scriptLog():String
<<Java Class>>
RandomAccessFileEditor
(default package)
nonStaticFileDirectory: String
RandomAccessFileEditor(String)
readData(String,int):String
getProjects(String):String[]
addProject(String,String):void
readCharacter(String,int):String
readASCII(String,int):String
readData(int):String
getProjects():String[]
addProject(String):void
readCharacter(int):String
readASCII(int):String
<<Java Class>>
Problem
(default package)
priority: int
title: String
description: String
Problem()
Problem(String,String,int)
Problem(String,int)
toString():String
setPriority(int):void
getPriority():int
setTitle(String):void
getTitle():String
setDescription(String):void
getDescription():String
compareTo(Problem):int
<<Java Class>>
DataEvent
(default package)
dataFieldChanged: String
originalDataValue: String
new DataValue: String
DataEvent()
DataEvent(String,String,String[])
DataEvent(String,String,String,String,String)
getDataFieldChanged():String
setDataFieldChanged(String):void
getOriginalDataValue():String
setOriginalDataValue(String):void
getNew DataValue():String
setNew DataValue(String):void
toString():String
scriptLog():String
<<Java Class>>
ProblemEvent
(default package)
problemTitle: String
problemDescription: String
problemPriority: String
problemAddedOrDeleted: int
ProblemEvent()
ProblemEvent(String,String,String[])
ProblemEvent(String,String,String,String,String,int)
getProblemTitle():String
setProblemTitle(String):void
getProblemDescription():String
setProblemDescription(String):void
getProblemPriority():String
setProblemPriority(String):void
getProblemAddedOrDeleted():int
setProblemAddedOrDeleted(int):void
toString():String
scriptLog():String
<<Java Class>>
TaskEvent
(default package)
taskCategory: String
taskContractor: String
taskProjectedCompletion: String
TaskEvent()
TaskEvent(String,String,String[])
TaskEvent(String,String,String,String,String)
getTaskCategory():String
setTaskCategory(String):void
getTaskContractor():String
setTaskContractor(String):void
getTaskProjectedCompletion():String
setTaskProjectedCompletion(String):void
toString():String
scriptLog():String
<<Java Class>>
CommentEvent
(default package)
comment: String
CommentEvent()
CommentEvent(String,String,String)
getComment():String
setComment(String):void
toString():String
scriptLog():String

Libraries Imported

```
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.StringTokenizer;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.ScrollPaneConstants;
import javax.swing.SwingConstants;
import javax.swing.table.DefaultTableModel;
```

- NumberFormat is used to format monetary values into currencies
- util and io were used to allow for file reading and writing along with ArrayLists
- awt and swing are used for the program's GUI **Panel Structure**

The program uses the following structure for its JPanels



Launching the GUI

```
public class GUI extends JFrame {
    private int accountIndexUsed;

    public static void main(String[] args) throws IOException {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try { //using a try-catch allows the program to robustly attempt running the GUI
                    GUI realEstateTrackerFrame = new GUI();
                    realEstateTrackerFrame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public GUI() throws IOException {
        setTitle("Real Estate Tracker"); //GUI extends JFrame so these methods reference the GUI object
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocation(200,100);
        setSize(750,475);
        setResizable(false); //ensures all components are always visible  (requested by client)
        getContentPane().setLayout(new CardLayout(0, 0));
```

accountIndexUsed will record the index of the client which is currently logged in

Login Pane

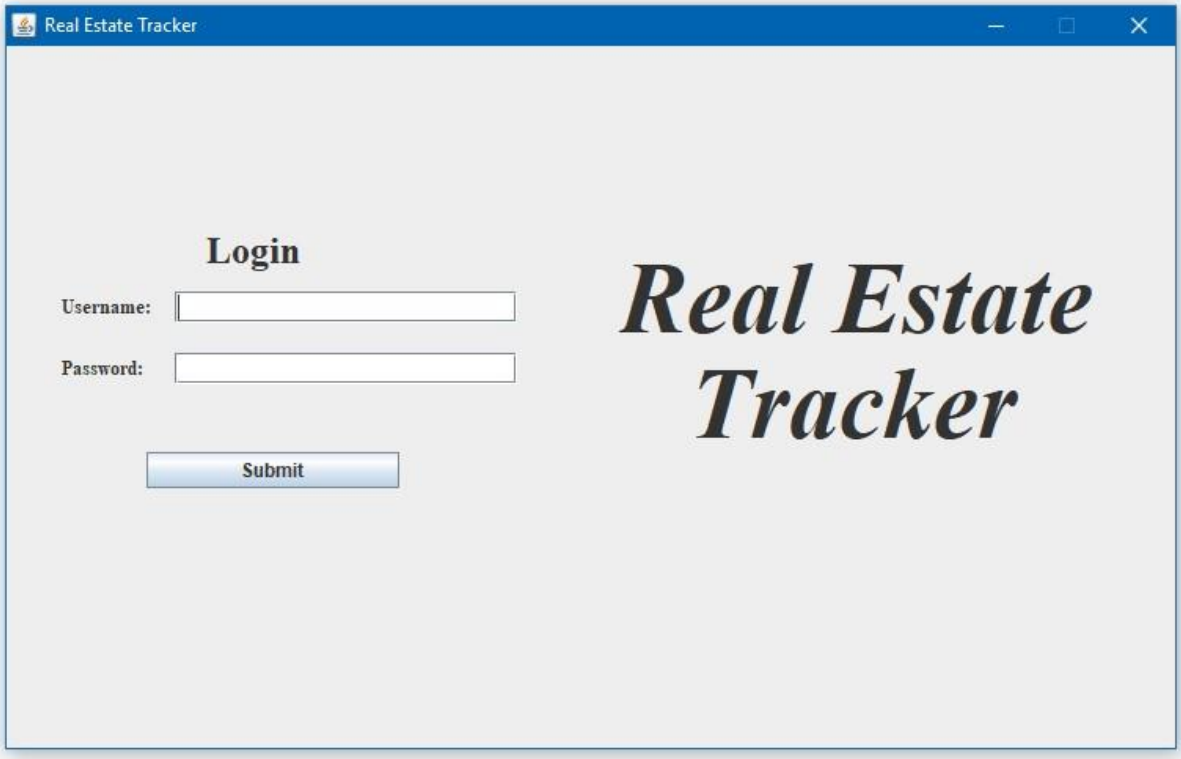


Figure 1: The login panel

Components used

```
JPanel loginPane = new JPanel();
loginPane.setForeground(Color.BLACK);
getContentPane().add(loginPane, "Login Pane");
loginPane.setLayout(null);

JTextField usernameField = new JTextField();
usernameField.setBounds(107, 156, 218, 20);
loginPane.add(usernameField);
usernameField.setColumns(10);

JPasswordField passwordField = new JPasswordField();
passwordField.setBounds(107, 195, 218, 20);
loginPane.add(passwordField);

JLabel lblPassword = new JLabel("Password:");
lblPassword.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblPassword.setBounds(35, 198, 62, 14);
loginPane.add(lblPassword);

JLabel lblUsername = new JLabel("Username:");
lblUsername.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblUsername.setBounds(35, 159, 62, 14);
loginPane.add(lblUsername);

JLabel lblIncorrectUandP = new JLabel("Incorrect username and password, please try again");
lblIncorrectUandP.setForeground(Color.RED);
lblIncorrectUandP.setFont(new Font("Times New Roman", Font.BOLD, 14));
lblIncorrectUandP.setBounds(27, 303, 358, 20);
loginPane.add(lblIncorrectUandP);
lblIncorrectUandP.setVisible(false);

JButton btnSubmit = new JButton("Submit");
btnSubmit.setBounds(89, 258, 161, 23);
loginPane.add(btnSubmit);


JLabel lblRealEstate = new JLabel("Real Estate");
lblRealEstate.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 64));
lblRealEstate.setBounds(390, 124, 310, 73);
loginPane.add(lblRealEstate);

JLabel labelTracker = new JLabel("Tracker");
labelTracker.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 64));
labelTracker.setBounds(434, 191, 298, 73);
loginPane.add(labelTracker);

JLabel lblLogin = new JLabel("Login");
lblLogin.setFont(new Font("Times New Roman", Font.BOLD, 24));
lblLogin.setBounds(127, 116, 109, 28);
loginPane.add(lblLogin);
```

Creating Account Objects

Once the program runs, the account whitelist is read, and the data is converted into account objects.

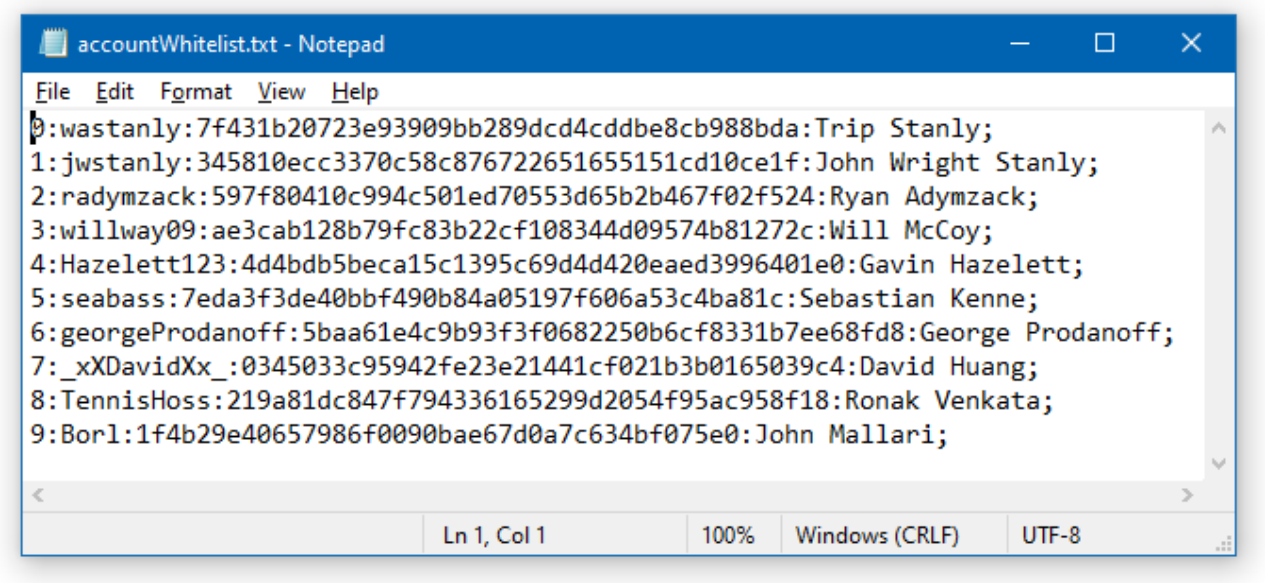


Figure 2: The account whitelist, where clients' account information is stored

The account whitelist stores 10 accounts, each with an index, username, hashed password, and fully printed name respectfully.

```
//ACCOUNT CREATOR
    //reads white list file and prints a list similar to the file's list
    FileReader accountWhitelist = new FileReader("accountWhitelist.txt");
    int charReader;
    int count=0;
    String[] accountStringList = new String[10];
    for(int x=0;x<10;x++) accountStringList[x]="";
    while ((charReader=accountWhitelist.read()) != -1) { //accountWhiteList is entirely read
        if((char)charReader != ';')
            accountStringList[count]+=(char)charReader; //non semi colons are read and added
        if((char)charReader == ';') { //semi colons indicate the account has been completely read
            count++;} //changes the accountStringList index
    }
    accountWhitelist.close();

    //creates the new Account classes
    Account[] account = new Account[10];
    for(int x=0;x<10;x++) {
        StringTokenizer st = new StringTokenizer(accountStringList[x],":"); //splits each account's line using the colon delimiter
        String[] items=new String[4];
        for(int xx=0;xx<=3;xx++) items[xx]=st.nextToken(); //adds each data point to the array
        account[x]=new Account(Integer.parseInt(items[0].trim()),items[1],items[2],items[3]); //passes each line into the account class
    }
```

accountWhitelist.txt is read.

```java
public class Account {

    //variables
    private int index;
    private String username;
    private String password;
    private String name;

    //constructors
    public Account(int i, String u, String p, String n) {
        index = i;
        username = u;
        password = p;
        name = n;
    }
    public Account() {
        this(0, "", "", "");
    }
    public void setAccount(Account a) {
        index = a.getIndex();
        username = a.getUsername();
        password = a.getPassword();
        name = a.getName();
    }

    //setters
    public void setIndex(int i){
        index = i;
    }
    public void setUsername(String u) {
        username = u;
    }
    public void setPassword(String p) {
        password = p;
    }
    public void setName(String n) {
        name = n;
    }

    //getters
    public int getIndex(){
        return index;
    }
    public String getUsername() {
        return username;
    }
    public String getPassword() {
        return password;
    }
    public String getName() {
        return name;
    }
}
```

Logging in the client

The Account and Login classes are used to log in, and can be triggered by the “submit” button or pressing enter in the PasswordField.

```
//creates the login system
class Login{
    public void runLogin() {
        for(int x=0;x<10;x++){
            if(account[x].getUsername().equals(usernameField.getText()) && account[x].getPassword().equals(
                    //generates the hash code for the inputed password
                    PasswordHasher.generateHash(new String(passwordField.getPassword()))))) {
                lblIncorrectUandP.setVisible(false);
                loginPane.setVisible(false);
                mainMenuPane.setVisible(true); //changes panels visible in the card layout
                accountIndexUsed = x; //records the index of the logged in client
                projectListMaker.createProjectList(); //creates the list of projects in the main menu
                lblWelcome.setText("Welcome " + account[accountIndexUsed].getName()); //builds the welcome JLabel
                }
            else lblIncorrectUandP.setVisible(true);
    }}
} Login login = new Login(); //creates an instance of the login class
```

Action listeners used:

```
//submit button
    btnSubmit.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0){
            login.runLogin();
        }});
//enter button
    passwordField.addKeyListener(new KeyAdapter() {
        public void keyPressed(KeyEvent evt){
            if(evt.getKeyCode()==KeyEvent.VK_ENTER) {
                login.runLogin();
            }
        }});
```

The runLogin() method compares the username and hashed password textfields to the array of account objects. Note if a match is found, accountIndexUsed stores the client’s index. If no matches are found, the alleged client is rejected (below).

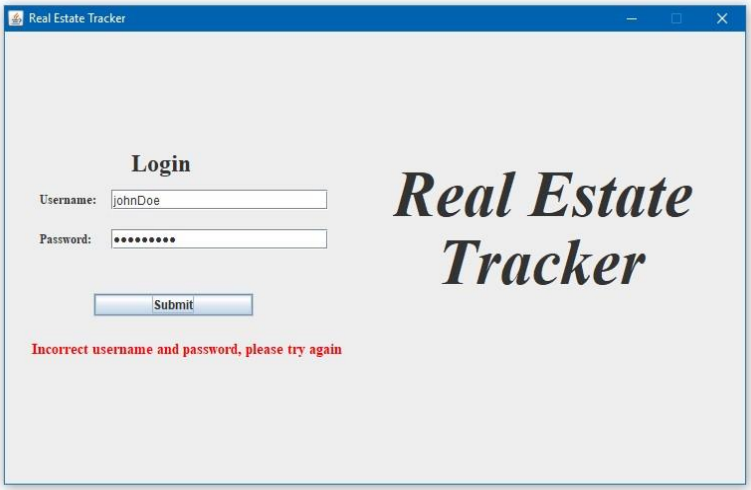


Figure 3: A failed login attempt

```java
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class PasswordHasher {

    public static String generateHash(String input) {

        //Creates string builder to add chars to
        StringBuilder newHash = new StringBuilder();

        try {
            //Creates instance of the Secure Hash Algorithm 1
            MessageDigest sha1 = MessageDigest.getInstance("SHA-1");

            //Computes bytes from the input string using the SHA-1 hash function
            byte[] hashedBytes = sha1.digest(input.getBytes());

            //Translates hashed bytes into append'able chars
            char[] possibleChars = {'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};
            for (int x=0;x<hashedBytes.length;++x) {
                byte b = hashedBytes[x];
                newHash.append(possibleChars[(b & 0xf0) >> 4]);
                newHash.append(possibleChars[b & 0x0f]);
            }
        }
        catch (NoSuchAlgorithmException e) {
            //catches possible exception
            System.out.println("ERROR IN GENERATING HASH FOR STRING: \" "+input+" \" ");
            e.printStackTrace();
        }

        return newHash.toString();
    }
}
```

Passwords are inputted into the SHA-1 cryptographic hash function along with chars and bitwise operators.

Main Menu

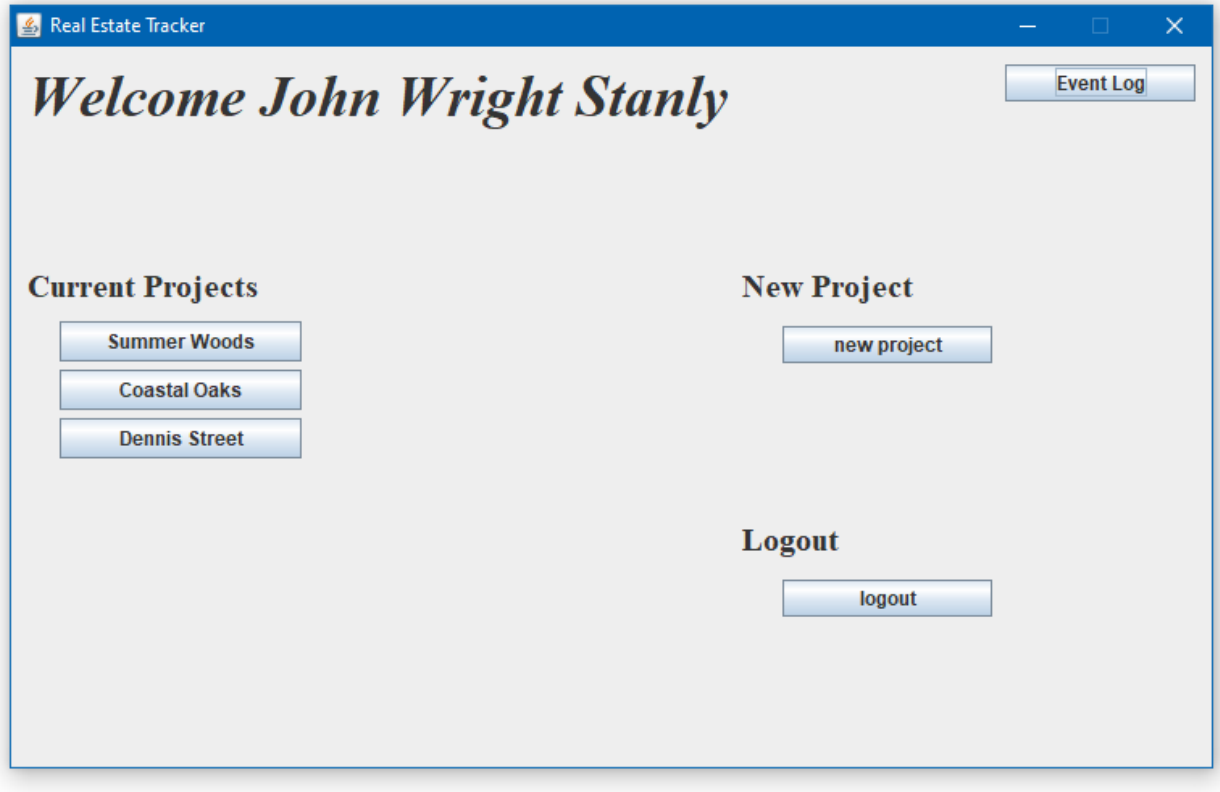


Figure 4: The main menu with the createProjectList() method outputting a single project button (Summer Woods)

Components Used

```
JPanel mainMenuPane = new JPanel();
getContentPane().add(mainMenuPane, "Main Menu Pane");
mainMenuPane.setLayout(null);

JLabel lblWelcome = new JLabel();
lblWelcome.setText("Welcome client");
lblWelcome.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
lblWelcome.setBounds(10, 0, 468, 62);
mainMenuPane.add(lblWelcome);

JLabel lblCurrentProjects = new JLabel("Current Projects");
lblCurrentProjects.setFont(new Font("Times New Roman", Font.BOLD, 20));
lblCurrentProjects.setBounds(10, 135, 189, 27);
mainMenuPane.add(lblCurrentProjects);

JLabel lblNewProject = new JLabel("New Project");
lblNewProject.setFont(new Font("Times New Roman", Font.BOLD, 20));
lblNewProject.setBounds(453, 135, 189, 27);
mainMenuPane.add(lblNewProject);

JLabel lblLogout = new JLabel("Logout");
lblLogout.setFont(new Font("Times New Roman", Font.BOLD, 20));
lblLogout.setBounds(453, 292, 189, 27);
mainMenuPane.add(lblLogout);

JButton btnLogout = new JButton("logout");
btnLogout.setBounds(478, 330, 130, 23);
mainMenuPane.add(btnLogout);

JButton btnNewProject = new JButton("new project");
btnNewProject.setBounds(478, 173, 130, 23);
mainMenuPane.add(btnNewProject);

JButton listOfProjectButtons[] = new JButton[50];
for(int x = 0; x < listOfProjectButtons.length; x++) {
    listOfProjectButtons[x] = new JButton("EMPTY");
    listOfProjectButtons[x].setBounds(30, (170+(30*x)), 150, 25);
    mainMenuPane.add(listOfProjectButtons[x]);
    listOfProjectButtons[x].setVisible(false);
}

JButton btnEventLog = new JButton("Event Log");
btnEventLog.setBounds(616, 11, 118, 23);
mainMenuPane.add(btnEventLog);
```

Project List Generation

```
class ProjectListGenerator{
    public void createProjectList() {
        for(int x=0;x<listOfProjectButtons.length;x++){ //resets all buttons
            listOfProjectButtons[x].setText("EMPTY");
            listOfProjectButtons[x].setBounds(30, (170+(30*x)), 150, 25);
            mainMenuPane.add(listOfProjectButtons[x]);
            listOfProjectButtons[x].setVisible(false);
        }

        String[] projectArray = null;
        try {
            projectArray = RandomAccessFileEditor.getProjects("ListOfProjectsRAF.txt");
            //Recursively reads the ListOfProjectsRAF file
        }
        catch (Exception e1) {
            e1.printStackTrace();
        }
        int count=0;
        for(int i=0;i<projectArray.length;i++) { //reads each line of the ListOfProjects.txt
            String projectFileName = projectArray[i];
            File accountFile = new File(projectFileName+"Accounts.txt");
            Scanner r2=null;
                try { //creates a scanner to search the accounts of each project in the ListOfProjects.txt
                    r2 = new Scanner(accountFile);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
            ArrayList<String> listOfAccounts = new ArrayList<String>();
            while(r2.hasNext()) listOfAccounts.add(r2.nextLine()); //makes a list of accounts per project
            r2.close();
            for(int x=0;x<listOfAccounts.size();x++) {
                if(listOfAccounts.get(x).equals(account[accountIndexUsed].getName())) {
                    //^checks if any of the names in the ArrayList match the logged in client
                    File dataFile = new File(projectFileName+"Data.txt");
                    Scanner r3=null;
                        try { //creates a scanner for the project's data file
                            r3 = new Scanner(dataFile);
                        } catch (FileNotFoundException e) {
                            e.printStackTrace();
                        }
                    String projectName = r3.nextLine(); //locates the project's name
                    r3.close();
                    listOfProjectButtons[count].setText(projectName);
                    listOfProjectButtons[count].setVisible(true); //creates a button for the respective project
                    count++;
                }else {
                    listOfProjectButtons[count].setVisible(false);
                }
            }
        }
    }
}  ProjectListGenerator projectListMaker = new ProjectListGenerator();
```

createProjectList() creates the JButtons for projects by using multiple loops and text files.

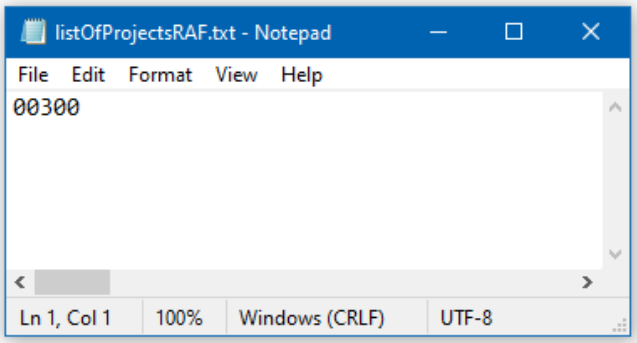


Figure 5: A sample List of Projects file (note the actual project names are much farther down the first line of the file)

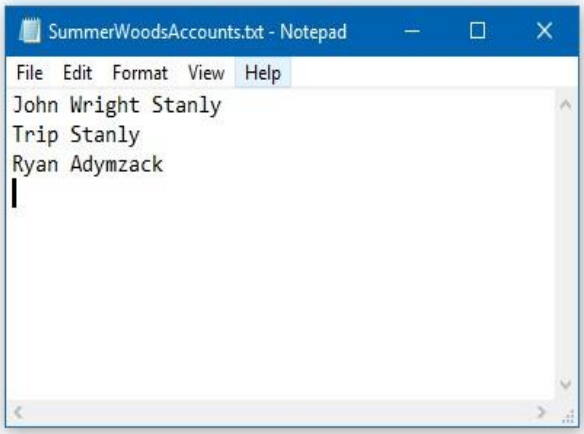


Figure 6: A sample project accounts file

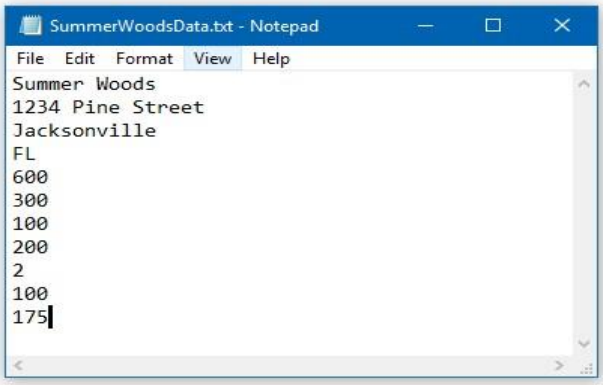


Figure 7: A sample project data file

```
public  class RandomAccessFileEditor {

    private String nonStaticFileDirectory;

    public RandomAccessFileEditor(String nonStaticFileDirectory) {
        this.nonStaticFileDirectory = nonStaticFileDirectory;
    }

    public static String readData(String filepath, int position) throws Exception{
        //makes RAF and seeks to correct position
        RandomAccessFile file = new RandomAccessFile(filepath, "rw");
        file.seek(position);

        //collects the first character to see if its a " " space for later testing
        int readInt = file.read();
        String output = "";

        //RECURSIVELY collects the other characters
        file.seek(position);
        if(readInt!=32) output += (char)file.read() + readData(filepath, position+1);

        return output;
    }

    public static String[] getProjects(String filepath) throws Exception {
        //makes the array list to add project names to
        ArrayList<String> projectsList = new ArrayList<String>();

        //finds total seek size
        int totalSeek = Integer.parseInt(readData(filepath, 0));

        //hunts through the document to add projects to the array
        for(int x=100;x<=totalSeek;x+=100) {
            projectsList.add(readData(filepath, x));
        }

        for(int x=0;x<projectsList.size();x++) System.out.println(projectsList.get(x));

        //converts array list to traditional array; returns array
        String[] projectArray = new String[projectsList.size()];
        for(int x=0;x<projectsList.size();x++) projectArray[x] = projectsList.get(x);

        return projectArray;
    }
```

ListOfProjectsRAF.txt is recursively read with readData(), and the results are placed into a String array with getProjects().

Logging Out

Pressing “log out” displays an option pane before taking the user back to the login pane:

```
//logout confirmation
btnLogout.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        int logoutConformation = JOptionPane.showConfirmDialog
                (null, "Are you sure?","Conformation", JOptionPane.YES_NO_OPTION);
        if(logoutConformation==0) { //runs if client selects "YES"
            loginPane.setVisible(true);
            mainMenuPane.setVisible(false); //takes client back to the login pane
            clear.loginPage(); //clears the login page
            clear.newProjectPage(); //clear the new project pane
        }
    }
});
```

Logging out also invokes the loginPage() method (below) and the newProjectPage() method (page 21) of the Clear class.

```
class Clear {
    public void loginPage() {
        passwordField.setText("");
        lblIncorrectUandP.setVisible(false);
    }
```

New Project Pane

Pressing “new project” on the main menu runs the following action listener:

```
//Takes client to the new project pane
btnNewProject.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        newProjectPane.setVisible(true);
        mainMenuPane.setVisible(false); //takes user to the new pane

        //renames check boxes to account names
        int accountIndexHasBeenMet=0;
        for(int x=0;x<account.length;x++) {
            if(x==accountIndexUsed) accountIndexHasBeenMet=1; //checks for the client's index
            if(x!=accountIndexUsed){ //skips over the client's index
                if(accountIndexHasBeenMet==0) chckbxAccount[x].setText(account[x].getName());
                if(accountIndexHasBeenMet==1) chckbxAccount[x-1].setText(account[x].getName());
                //^titles checkboxes in the correct index, based on the if the client's index has been met
            }
        }
    }
});
```

The action listener titles checkboxes correctly by skipping the client’s index through nested if statements.

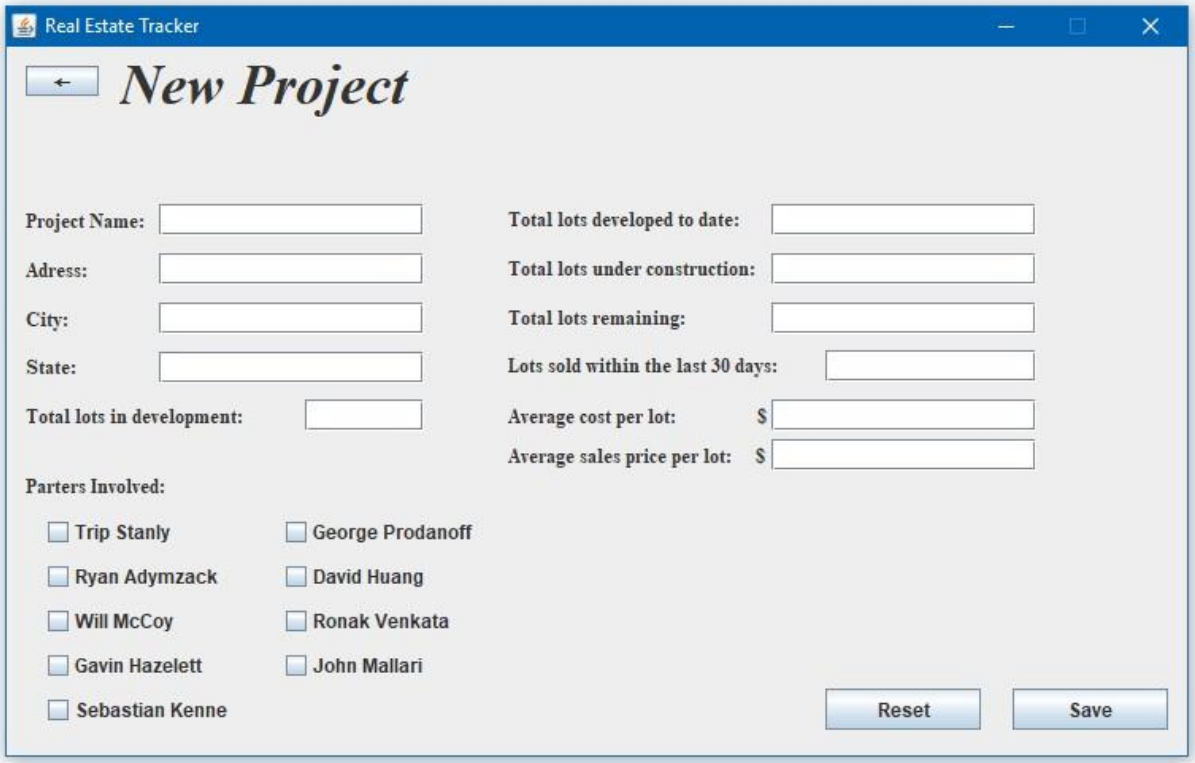


Figure 8: An empty new project pane. Notice the Partners Involved checkboxes avoid the account "John Wright Stanly", since "John Wright Stanly" is the currently logged in client

Components Used

```
JPanel newProjectPane = new JPanel();
getContentPane().add(newProjectPane, "New Project Pane");
newProjectPane.setLayout(null);

JLabel lblNewProjectTitle = new JLabel("New Project");
lblNewProjectTitle.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
lblNewProjectTitle.setBounds(71, 0, 235, 49);
newProjectPane.add(lblNewProjectTitle);

JButton btnBack = new JButton("\u2190");
btnBack.setToolTipText("Going to the main menu won't delete your data");
btnBack.setBounds(12, 12, 46, 19);
newProjectPane.add(btnBack);
btnBack.setFocusPainted(false);

JLabel lblProjectName = new JLabel("Project Name:");
lblProjectName.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblProjectName.setBounds(12, 102, 91, 16);
newProjectPane.add(lblProjectName);

JTextField projectNameTextfield = new JTextField();
projectNameTextfield.setBounds(96, 99, 167, 20);
newProjectPane.add(projectNameTextfield);
projectNameTextfield.setColumns(10);

JLabel lblAdress = new JLabel("Adress: ");
lblAdress.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblAdress.setBounds(12, 133, 91, 16);
newProjectPane.add(lblAdress);

JTextField adressTextfield = new JTextField();
adressTextfield.setColumns(10);
adressTextfield.setBounds(96, 130, 167, 20);
newProjectPane.add(adressTextfield);

JLabel lblCity = new JLabel("City: ");
lblCity.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblCity.setBounds(12, 164, 91, 16);
newProjectPane.add(lblCity);
```

```
JLabel lblState = new JLabel("State: ");
lblState.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblState.setBounds(12, 194, 91, 16);
newProjectPane.add(lblState);

JTextField stateTextfield = new JTextField();
stateTextfield.setColumns(10);
stateTextfield.setBounds(96, 192, 167, 20);
newProjectPane.add(stateTextfield);

JLabel lblTotalLotsInDev = new JLabel("Total lots in development: ");
lblTotalLotsInDev.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblTotalLotsInDev.setBounds(12, 225, 147, 16);
newProjectPane.add(lblTotalLotsInDev);

JTextField totalLotsInDevTextfield = new JTextField();
totalLotsInDevTextfield.setColumns(10);
totalLotsInDevTextfield.setBounds(188, 222, 75, 20);
newProjectPane.add(totalLotsInDevTextfield);

JLabel lblPartersInvolved = new JLabel("Parters Involved: ");
lblPartersInvolved.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblPartersInvolved.setBounds(12, 269, 104, 16);
newProjectPane.add(lblPartersInvolved);

JCheckBox chckbxAccount[] = new JCheckBox[9];
for(int x = 0; x < chckbxAccount.length; x++) {
    chckbxAccount[x] = new JCheckBox();
    if(x<5) chckbxAccount[x].setBounds(22,293+28*x,132,24);
    else chckbxAccount[x].setBounds(172, 293+28*(x-5), 132, 24);
    newProjectPane.add(chckbxAccount[x]);
}

JLabel lblTotalDevelopedTo = new JLabel("Total lots developed to date: ");
lblTotalDevelopedTo.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblTotalDevelopedTo.setBounds(316, 101, 167, 16);
newProjectPane.add(lblTotalDevelopedTo);
```

```
JTextField lblTotalDevelopedTextfield = new JTextField();
lblTotalDevelopedTextfield.setColumns(10);
lblTotalDevelopedTextfield.setBounds(482, 99, 167, 20);
newProjectPane.add(lblTotalDevelopedTextfield);

JLabel TotalLotsUnderConstruction = new JLabel("Total lots under construction:  ");
TotalLotsUnderConstruction.setFont(new Font("Times New Roman", Font.BOLD, 12));
TotalLotsUnderConstruction.setBounds(316, 132, 167, 16);
newProjectPane.add(TotalLotsUnderConstruction);

JTextField totalLotsUnderConstructionTextfield = new JTextField();
totalLotsUnderConstructionTextfield.setColumns(10);
totalLotsUnderConstructionTextfield.setBounds(482, 130, 167, 20);
newProjectPane.add(totalLotsUnderConstructionTextfield);

JLabel lblTotalLotsRemaining = new JLabel("Total lots remaining:   ");
lblTotalLotsRemaining.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblTotalLotsRemaining.setBounds(316, 163, 167, 16);
newProjectPane.add(lblTotalLotsRemaining);

JTextField totalLotsRemainingTextfield = new JTextField();
totalLotsRemainingTextfield.setColumns(10);
totalLotsRemainingTextfield.setBounds(482, 161, 167, 20);
newProjectPane.add(totalLotsRemainingTextfield);

JLabel lblLotsSoldWithin = new JLabel("Lots sold within the last 30 days: ");
lblLotsSoldWithin.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblLotsSoldWithin.setBounds(316, 193, 195, 16);
newProjectPane.add(lblLotsSoldWithin);

JTextField lotsSoldWithinTextfield = new JTextField();
lotsSoldWithinTextfield.setColumns(10);
lotsSoldWithinTextfield.setBounds(516, 191, 133, 20);
newProjectPane.add(lotsSoldWithinTextfield);

JLabel lblAverageCostPer = new JLabel("Average cost per lot:                $");
lblAverageCostPer.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblAverageCostPer.setBounds(316, 225, 195, 16);
newProjectPane.add(lblAverageCostPer);
```

```
JLabel lblAverageCostPer = new JLabel("Average cost per lot:                $");
lblAverageCostPer.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblAverageCostPer.setBounds(316, 225, 195, 16);
newProjectPane.add(lblAverageCostPer);

JTextField averageCostPerTextfield = new JTextField();
averageCostPerTextfield.setColumns(10);
averageCostPerTextfield.setBounds(482, 222, 167, 20);
newProjectPane.add(averageCostPerTextfield);

JLabel lblAveragePricePer = new JLabel("Average sales price per lot:     $");
lblAveragePricePer.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblAveragePricePer.setBounds(316, 250, 195, 16);
newProjectPane.add(lblAveragePricePer);

JTextField averagePricePerTextfield = new JTextField();
averagePricePerTextfield.setColumns(10);
averagePricePerTextfield.setBounds(482, 247, 167, 20);
newProjectPane.add(averagePricePerTextfield);

JButton btnSave = new JButton("Save");
btnSave.setBounds(634, 404, 98, 26);
newProjectPane.add(btnSave);

JButton btnReset = new JButton("Reset");
btnReset.setBounds(516, 404, 98, 26);
newProjectPane.add(btnReset);
```

Saving a Project

The following action listener runs when “save” is pressed:

```
//saving a new project
btnSave.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        boolean emptyTextFields = false;
        boolean noNumberTextFields = false; //sets all error booleans to false

        if(projectNameTextfield.getText().isEmpty() ||
            adressTextfield.getText().isEmpty() ||
            cityTextfield.getText().isEmpty() ||
            stateTextfield.getText().isEmpty() ||
            totalLotsInDevTextfield.getText().isEmpty() ||
            lblTotalDevelopedTextfield.getText().isEmpty() ||
            totalLotsUnderConstructionTextfield.getText().isEmpty() ||
            totalLotsRemainingTextfield.getText().isEmpty() ||
            lotsSoldWithinTextfield.getText().isEmpty() ||
            averageCostPerTextfield.getText().isEmpty() ||
            averagePricePerTextfield.getText().isEmpty()) {
            //^checks if any of the textfields are left empty
                emptyTextFields =  true; //triggers error boolean
                System.out.println("ERROR: Empty text fields");
        }

        if(emptyTextFields==true) { //runs if textfield(s) left empty
            JOptionPane.showMessageDialog(new JFrame(),
                    "All textboxes must be filled before submission",
                    "ERROR",
                    JOptionPane.WARNING_MESSAGE);
            //^displays an error option pane advising the client to fill all textfields
        }

        if(emptyTextFields==false) { //runs if all textfields contain data
            try {
                Integer.parseInt(totalLotsInDevTextfield.getText());
                Integer.parseInt(lblTotalDevelopedTextfield.getText());
                Integer.parseInt(totalLotsUnderConstructionTextfield.getText());
                Integer.parseInt(totalLotsRemainingTextfield.getText());
                Integer.parseInt(lotsSoldWithinTextfield.getText());
                Double.parseDouble(averageCostPerTextfield.getText());
                Double.parseDouble(averagePricePerTextfield.getText());
                //^checks if certain textfields are numbers
            }
            catch(NumberFormatException e1) {
                JOptionPane.showMessageDialog(new JFrame(),
                        "Please insert numbers where needed",
                        "ERROR",
                        JOptionPane.WARNING_MESSAGE);
            //^displays an error option pane advising the client to fill in numbers
                noNumberTextFields = true; //triggers erorr boolean
            }
        }
```

```
if(noNumberTextFields==false && emptyTextFields==false) { //runs if no error booleans
    //creating file's name
    StringTokenizer st = new StringTokenizer(projectNameTextfield.getText()," ");
    String fileName = "";
    while(st.hasMoreTokens()) fileName+=st.nextToken(); //builds the project name

    //data file
    File dataFile = new File(fileName+"Data.txt"); //creates the project's data file
    FileWriter dataWriter = null;
        try {dataWriter = new FileWriter(dataFile);}
        catch (IOException e1) {e1.printStackTrace();}
    PrintWriter dpw = new PrintWriter(dataWriter); //makes a File and PrintWriter
    dpw.println(projectNameTextfield.getText());
    dpw.println(adressTextfield.getText());
    dpw.println(cityTextfield.getText());
    dpw.println(stateTextfield.getText());
    dpw.println(totalLotsInDevTextfield.getText());
    dpw.println(lblTotalDevelopedTextfield.getText());
    dpw.println(totalLotsUnderConstructionTextfield.getText());
    dpw.println(totalLotsRemainingTextfield.getText());
    dpw.println(lotsSoldWithinTextfield.getText());
    dpw.println(averageCostPerTextfield.getText());
    dpw.println(averagePricePerTextfield.getText());
    //^writes the project's data into the new data file
    dpw.close();

    //account file
    File accountFile = new File(fileName+"Accounts.txt"); //creates the project's account file
    FileWriter accountWriter = null;
        try {accountWriter = new FileWriter(accountFile);}
        catch (IOException e1) {e1.printStackTrace();}
    PrintWriter apw = new PrintWriter(accountWriter); //makes a File and PrintWriter
    apw.println(account[accountIndexUsed].getName()); //automatically adds the client to the file
    for(int x=0;x<chckbxAccount.length;x++)
        if(chckbxAccount[x].isSelected())
            apw.println(chckbxAccount[x].getText());
    //^if checkbox is selected, then the account is added to the project account file
    apw.close();

    //writes new project to the list of projects file
    try{
        RandomAccessFileEditor.addProject("listOfProjectsRAF.txt", fileName);
    }
    catch (Exception e2) {
        e2.printStackTrace();
    }
```

```
            //creates task, comment, and problem files
            File tasksFile = new File(fileName+"Tasks.csv"); //creates the project's tasks file
                FileWriter tasksFileWriter = null;
                    try {tasksFileWriter = new FileWriter(tasksFile, true);}
                    catch (IOException e1) {e1.printStackTrace();}
                PrintWriter tasksPW = new PrintWriter(tasksFileWriter); //makes a File and PrintWriter
                tasksPW.print("Task,Contractor,Projected Completion");
                //^prints headers despite using a tableModel (used for exports; requested by the client)
                tasksPW.close();
            File commentsFile = new File(fileName+"Comments.txt"); //creates the project's comment file
                FileWriter commentsFileWriter = null;
                    try {commentsFileWriter = new FileWriter(commentsFile, true);}
                    catch (IOException e1) {e1.printStackTrace();}
                PrintWriter commentsPW = new PrintWriter(commentsFileWriter); //makes a File and PrintWriter
                commentsPW.print(""); //ensures project comments file is ready for future comments
                commentsPW.close();
            File problemsFile = new File(fileName+"Problems.csv"); //creates the project's problems file
                FileWriter problemsFileWriter = null;
                    try {problemsFileWriter = new FileWriter(problemsFile, true);}
                    catch (IOException e1) {e1.printStackTrace();}
                PrintWriter problemsPW = new PrintWriter(problemsFileWriter); //makes a File and PrintWriter
                problemsPW.print("Title,Description,Priority");
                //^ensures project problems file is ready for future problems
                problemsPW.close();

        //GUI processes
            mainMenuPane.setVisible(true);
            newProjectPane.setVisible(false); //returns user to the main menu
            clear.newProjectPage(); //clears the new project pane
            projectListMaker.createProjectList();
            //^reruns the project list algorithm so that the new project appears as a button
        }

    }
});
```

The action listener checks for errors, saves to files (Figures 6 and 7 respectfully), adds headers to the new tasks file (Figure 13), creates the comments file (Figure 15) and problems file (Figure 16), and generates the button for the project on the main menu (Figure 4).

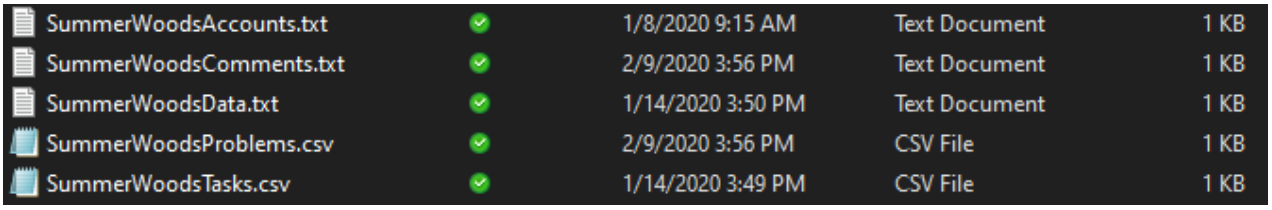


Figure 9: Sample Accounts, Comments, Data, and Tasks files for the Summer Woods project

Additionally, addProject() from RandomAccessFileEditor adds the project to the list of projects (Figure 5) with the procedure below.

```
public static void addProject(String filepath, String projectTitle) throws Exception {

    RandomAccessFile file = new RandomAccessFile(filepath, "rw");

    //read the size of the RAF
    int totalSeek = Integer.parseInt(readData(filepath, 0));
    int newTotal = totalSeek+100;
    String newTotalString = null;
    if(totalSeek>=000 && totalSeek<1000) newTotalString = "00"+newTotal;
    else if(totalSeek>=1000 && totalSeek<10000) newTotalString = "0"+newTotal;
    else if(totalSeek>=10000 && totalSeek<100000) newTotalString = ""+newTotal;

    //go back to the beginning and replace the new total size
    file.seek(0);
    file.write(newTotalString.getBytes("UTF-8"));

    //go to the end of the file and add the new project
    file.seek(newTotal);
    String projectTitleToAdd = projectTitle+" ";
    file.write(projectTitleToAdd.getBytes("UTF-8"));

    file.close();

    System.out.println(projectTitle + " added to file");
}
```

The “reset” button also can be pressed to flush out any data values in the textfields by invoking the newProjectPage() method of the Clear class:

```
//resets the new project pane
btnReset.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        Object[] warningOptions={"Return","Reset new project"};
        int resetWarning = JOptionPane.showOptionDialog //generates a JOptionPane warning the client
                (null, "WARNING\nAre you sure you want to reset your new project?",
                        "Warning",JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,
                        null, warningOptions, warningOptions[0]);
        if(resetWarning==1) { //runs if "reset" was nevertheless selected
            clear.newProjectPage(); //resets the new project page
        }
}});
```

```
public void newProjectPage() {
    projectNameTextfield.setText("");
    adressTextfield.setText("");
    cityTextfield.setText("");
    stateTextfield.setText("");
    totalLotsInDevTextfield.setText("");
    lblTotalDevelopedTextfield.setText("");
    totalLotsUnderConstructionTextfield.setText("");
    totalLotsRemainingTextfield.setText("");
    lotsSoldWithinTextfield.setText("");
    averageCostPerTextfield.setText("");
    averagePricePerTextfield.setText("");
    //^clears all the text fields
    for(JCheckBox x : chckbxAccount) x.setSelected(false);
    //^clears all the partner involved checkboxes
}
```

The “back” button returns users to the main menu:

```
btnBack.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        newProjectPane.setVisible(false);
        mainMenuPane.setVisible(true);
        setTitle("Real Estate Tracker");
        //^removes specific project from the window title
        projectListMaker.createProjectList();
        //^regenerates the client's list of projects
    }
});
```

Project Pane

Each project button has an action listener taking the client to the project pane:

```
//takes client to the respective project page
for(int x=0;x<listOfProjectButtons.length;x++){
    listOfProjectButtons[x].addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            mainMenuPane.setVisible(false);
            projectPane.setVisible(true); //takes user to the respective project
            lblProjectTitle.setText(e.getActionCommand()); //writes the project title label
            tabbedPane.setSelectedIndex(0);
            displayData.projectData(); //displays the project's data
            taskTable.generateData(); //generates the task table
            commentBoard.generateCommentBoard(); //generates the comment board
            setTitle(e.getActionCommand()+" - Real Estate Tracker"); //renames window title
        }
    });
}
```

Components Used

```
JPanel projectPane = new JPanel();
getContentPane().add(projectPane, "Project Pane");
projectPane.setLayout(null);

JLabel lblProjectTitle = new JLabel("Project X");
lblProjectTitle.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
lblProjectTitle.setBounds(71, 0, 496, 49);
projectPane.add(lblProjectTitle);

JButton btnBack1 = new JButton("\u2190");
btnBack1.setBounds(12, 12, 46, 19);
projectPane.add(btnBack1);
btnBack1.setFocusPainted(false);

JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
tabbedPane.setBounds(0, 60, 744, 386);
projectPane.add(tabbedPane);
```

The client can return to the main menu by pressing the back button, which runs the following action listener:

```
btnBack1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        projectPane.setVisible(false);
        mainMenuPane.setVisible(true);
        setTitle("Real Estate Tracker");
        //^removes specific project from the window title
        projectListMaker.createProjectList();
        //^regenerates the client's list of projects
    }
});
```

Each project pane is divided into four tabs: “details”, “tasks”, “comments”, and “problems”.

Project Pane (Details)

Figure 10: A project pane tabbed at details

Components Used

```
JPanel detailsPanel = new JPanel();
tabbedPane.addTab("Details", null, detailsPanel, null);
detailsPanel.setLayout(null);

JLabel projectAddressLabel = new JLabel("Address: ");
projectAddressLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectAddressLabel.setBounds(25, 20, 91, 16);
detailsPanel.add(projectAddressLabel);

JLabel projectCityLabel = new JLabel("City: ");
projectCityLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectCityLabel.setBounds(25, 40, 91, 16);
detailsPanel.add(projectCityLabel);

JLabel projectStateLabel = new JLabel("State: ");
projectStateLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectStateLabel.setBounds(25, 60, 91, 16);
detailsPanel.add(projectStateLabel);

JLabel projectTotalLotsInDevelopmentLabel = new JLabel("Total Lots in Development: ");
projectTotalLotsInDevelopmentLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectTotalLotsInDevelopmentLabel.setBounds(25, 100, 159, 16);
detailsPanel.add(projectTotalLotsInDevelopmentLabel);

JLabel projectTotalLotsDevelopedToDateLabel = new JLabel("Total Lots Developed To Date: ");
projectTotalLotsDevelopedToDateLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectTotalLotsDevelopedToDateLabel.setBounds(25, 120, 178, 16);
detailsPanel.add(projectTotalLotsDevelopedToDateLabel);

JLabel projectTotalLotsUnderConstrustionLabel = new JLabel("Total Lots Under Construction: ");
projectTotalLotsUnderConstrustionLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectTotalLotsUnderConstrustionLabel.setBounds(25, 140, 178, 16);
detailsPanel.add(projectTotalLotsUnderConstrustionLabel);

JLabel projectTotalLotsRemainingLabel = new JLabel("Total Lots Remaining: ");
projectTotalLotsRemainingLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectTotalLotsRemainingLabel.setBounds(25, 160, 130, 16);
detailsPanel.add(projectTotalLotsRemainingLabel);
```

```
JLabel projectLotsSoldWithinLast30DaysLabel = new JLabel("Lots Sold Within the Last 30 Days: ");
projectLotsSoldWithinLast30DaysLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectLotsSoldWithinLast30DaysLabel.setBounds(25, 200, 197, 16);
detailsPanel.add(projectLotsSoldWithinLast30DaysLabel);

JLabel projectSelloutLabel = new JLabel("Project Sellout: ");
projectSelloutLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectSelloutLabel.setBounds(25, 220, 106, 16);
detailsPanel.add(projectSelloutLabel);

JLabel projectAvgCostPerLotLabel = new JLabel("Average Cost Per Lot: ");
projectAvgCostPerLotLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectAvgCostPerLotLabel.setBounds(25, 260, 144, 16);
detailsPanel.add(projectAvgCostPerLotLabel);

JLabel projectAvgPricePerLotLabel = new JLabel("Average Sales Price Per Lot: ");
projectAvgPricePerLotLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectAvgPricePerLotLabel.setBounds(25, 280, 178, 16);
detailsPanel.add(projectAvgPricePerLotLabel);

JLabel projectTotalRemainingRevenueLabel = new JLabel("Total Remaining Revenue: ");
projectTotalRemainingRevenueLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectTotalRemainingRevenueLabel.setBounds(25, 300, 178, 16);
detailsPanel.add(projectTotalRemainingRevenueLabel);

JLabel projectProjectedRemainingProfitLabel = new JLabel("Projected Remaining Profit: ");
projectProjectedRemainingProfitLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
projectProjectedRemainingProfitLabel.setBounds(25, 320, 197, 16);
detailsPanel.add(projectProjectedRemainingProfitLabel);

JButton btnEdit = new JButton("Edit");
btnEdit.setBounds(622, 317, 89, 23);
detailsPanel.add(btnEdit);

JLabel lblPartnersInvolved = new JLabel("Partners Involved");
lblPartnersInvolved.setBounds(475, 13, 178, 23);
lblPartnersInvolved.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 22));
detailsPanel.add(lblPartnersInvolved);
```

```
JLabel[] projectData = new JLabel[16];
for(int x=0;x<projectData.length;x++) {
    projectData[x] = new JLabel("This is a test message");
    detailsPanel.add(projectData[x]);
    projectData[x].setFont(new Font("Times New Roman", Font.BOLD, 12));}
projectData[0].setBounds(231, 20, 200, 16);
projectData[1].setBounds(231, 40, 91, 16);
projectData[2].setBounds(231, 60, 91, 16);
projectData[3].setBounds(231, 100, 91, 16);
projectData[4].setBounds(231, 120, 91, 16);
projectData[5].setBounds(231, 140, 91, 16);
projectData[6].setBounds(231, 160, 91, 16);
projectData[7].setBounds(231, 200, 91, 16);
projectData[8].setBounds(231, 220, 91, 16);
projectData[9].setBounds(231, 260, 91, 16);
projectData[10].setBounds(231, 280, 91, 16);
projectData[11].setBounds(231, 300, 91, 16);
projectData[12].setBounds(231, 320, 91, 16);
projectData[13].setBounds(270, 120, 91, 16);
projectData[14].setBounds(270, 140, 91, 16);
projectData[15].setBounds(270, 160, 91, 16);

JLabel[] partnersInvoled = new JLabel[10];
for(int x=0;x<10;x++) {
    partnersInvoled[x] = new JLabel("Partner Involed");
    partnersInvoled[x].setFont(new Font("Times New Roman", Font.BOLD, 12));
    partnersInvoled[x].setBounds(485, 40+(x*20), 200, 15);
}
```

As the action listener shows, the projectData() method from the DisplayData class runs upon opening a project from the main menu:

```
class DisplayData{
    public void projectData() {
        //READ DATA
        NumberFormat cf = NumberFormat.getCurrencyInstance();
        //^creates a currency number formatter for financial data values
        String input = lblProjectTitle.getText();
        StringTokenizer st = new StringTokenizer(input);
        String nameBuilder="";
        while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
        //^creates the project's data name through tokenizing the title label
        File projectDataFile = new File(nameBuilder+"Data.txt");
        Scanner reader=null;
            try {reader = new Scanner(projectDataFile);}
            catch (FileNotFoundException e) {e.printStackTrace();}
        reader.nextLine();
        //^creates a scanner for the project's data file

        for(int x=0;x<=7;x++) projectData[x].setText(reader.nextLine());
        //^the first seven data points are directly converted into labels

        double totalRemaining = (Integer.parseInt(projectData[5].getText()) +
                                 Integer.parseInt(projectData[6].getText()));
        //^calculates a total remaining value, used in future calculations

        int selloutInteger = (int) Math.ceil(totalRemaining /
                                  Integer.parseInt(projectData[7].getText()));
        //^calculates the maximum number of months needed to completely sell out
        String selloutString =""+selloutInteger;
        projectData[8].setText(selloutString+" months");
        //^converts calculated number into a string and into a label

        double[] costsUnformatted = new double[2];
        for(int x=9;x<=10;x++){ //reads specifically 2 lines
            double value = Double.parseDouble(reader.nextLine());
            //^reads two lines
            costsUnformatted[x-9]=value;
            projectData[x].setText(""+cf.format(value));
            //^converts the data into currencies and puts them into labels
        }
```

```
        double revenue = totalRemaining * costsUnformatted[1];
        projectData[11].setText(""+cf.format(revenue));
        //^calculates total revenue and places it in a label

        double profit = costsUnformatted[1] - costsUnformatted[0];
        double totalProfit = profit * totalRemaining;
        projectData[12].setText(""+cf.format(totalProfit));
        //^subtracts costs from revenue to calculate profit; places in label

        double percentDeveloped = (Double.parseDouble(projectData[4].getText()) /
                                   Double.parseDouble(projectData[3].getText()))*100;
        percentDeveloped = Math.round(percentDeveloped*100)/100;
        projectData[13].setText("("+(int)percentDeveloped+"%)");
        //^calculates the percentage of properties that are in development

        double percentUnderConstruction = (Double.parseDouble(projectData[5].getText()) /
                                           Double.parseDouble(projectData[3].getText()))*100;
        percentUnderConstruction = Math.round(percentUnderConstruction*100)/100;
        projectData[14].setText("("+(int)percentUnderConstruction+"%)");
        //^calculates the percentage of properties that are under construction

        double percentRemaining = (Double.parseDouble(projectData[6].getText()) /
                                   Double.parseDouble(projectData[3].getText()))*100;
        percentRemaining = Math.round(percentRemaining*100)/100;
        projectData[15].setText("("+(int)percentRemaining+"%)");
        //^calculates the percentage of properties that are remaining

        reader.close();


        //FIND PARTNERS INVOLVED
        File projectAccountFile = new File(nameBuilder+"Accounts.txt");
        Scanner r1=null;
            try {r1 = new Scanner(projectAccountFile);}
            catch (FileNotFoundException e) {e.printStackTrace();}
        //^creates a new scanner in the project's account file
        ArrayList<String> accountNames = new ArrayList<String>();
        while(r1.hasNextLine()) accountNames.add(r1.nextLine());
        //^makes an ArrayList containing all the accounts involved in the project
        int JLabelCounter = 0;
        for(int x=0;x<accountNames.size();x++) {
            partnersInvoled[x].setText("- "+accountNames.get(x));
            partnersInvoled[x].setVisible(true);
            detailsPanel.add(partnersInvoled[x]);
            JLabelCounter++;
            //^makes partnersInvoled labels only visible for the needed partners
        }
        for(int x = JLabelCounter;x<10;x++){
            partnersInvoled[x].setVisible(false);
            //^ensures the rest are reset and non-visible
        }
        r1.close();
    }
```

The projectData() method transfers data form the data file to its according JLabels, in addition to calculating new inferred values from the original data set.

Edit Project Pane

Clicking “edit” on the details pane in the project pane runs this:

```
btnEdit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        displayData.editProjectData();
    }
});
```

Components Used

```
JPanel editProjectPane = new JPanel();
getContentPane().add(editProjectPane, "Edit Project Pane");
editProjectPane.setLayout(null);

JLabel lblEditProject = new JLabel("Edit Project");
lblEditProject.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
lblEditProject.setBounds(71, 0, 235, 49);
editProjectPane.add(lblEditProject);

JButton editProjectPaneEditButton = new JButton("\u2190");
editProjectPaneEditButton.setToolTipText("Going to the project menu WILL remove your revisions");
editProjectPaneEditButton.setFocusPainted(false);
editProjectPaneEditButton.setBounds(12, 12, 46, 19);
editProjectPane.add(editProjectPaneEditButton);

JLabel editProjectPaneProjectName = new JLabel("Project Name:");
editProjectPaneProjectName.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneProjectName.setBounds(12, 102, 91, 16);
editProjectPane.add(editProjectPaneProjectName);

JTextField editProjectPaneProjectNameTextfield = new JTextField();
editProjectPaneProjectNameTextfield.setColumns(10);
editProjectPaneProjectNameTextfield.setBounds(96, 99, 167, 20);
editProjectPane.add(editProjectPaneProjectNameTextfield);
editProjectPaneProjectNameTextfield.setEditable(false);

JLabel editProjectPaneAdress = new JLabel("Adress: ");
editProjectPaneAdress.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneAdress.setBounds(12, 133, 91, 16);
editProjectPane.add(editProjectPaneAdress);

JTextField editProjectPaneAdressTextfield = new JTextField();
editProjectPaneAdressTextfield.setColumns(10);
editProjectPaneAdressTextfield.setBounds(96, 130, 167, 20);
editProjectPane.add(editProjectPaneAdressTextfield);
editProjectPaneAdressTextfield.setEditable(false);

JLabel editProjectPaneCity = new JLabel("City: ");
editProjectPaneCity.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneCity.setBounds(12, 164, 91, 16);
editProjectPane.add(editProjectPaneCity);
```

```
JTextField editProjectPaneCityTextfield = new JTextField();
editProjectPaneCityTextfield.setColumns(10);
editProjectPaneCityTextfield.setBounds(96, 161, 167, 20);
editProjectPane.add(editProjectPaneCityTextfield);
editProjectPaneCityTextfield.setEditable(false);

JLabel editProjectPaneState = new JLabel("State: ");
editProjectPaneState.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneState.setBounds(12, 194, 91, 16);
editProjectPane.add(editProjectPaneState);

JTextField editProjectPaneStateTextfield = new JTextField();
editProjectPaneStateTextfield.setColumns(10);
editProjectPaneStateTextfield.setBounds(96, 192, 167, 20);
editProjectPane.add(editProjectPaneStateTextfield);
editProjectPaneStateTextfield.setEditable(false);

JLabel editProjectPaneTLID = new JLabel("Total lots in development: ");
editProjectPaneTLID.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneTLID.setBounds(12, 225, 147, 16);
editProjectPane.add(editProjectPaneTLID);

JTextField  editProjectPaneTLIDTextfield = new JTextField();
//editProjectPaneTLIDTextfield.setHorizontalAlignment(JTextField.RIGHT);
editProjectPaneTLIDTextfield.setColumns(10);
editProjectPaneTLIDTextfield.setBounds(188, 222, 75, 20);
editProjectPane.add(editProjectPaneTLIDTextfield);

JLabel editProjectPaneTLDTD = new JLabel("Total lots developed to date: ");
editProjectPaneTLDTD.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneTLDTD.setBounds(316, 101, 167, 16);
editProjectPane.add(editProjectPaneTLDTD);

JTextField editProjectPaneTLDTDTextfield = new JTextField();
editProjectPaneTLDTDTextfield.setColumns(10);
editProjectPaneTLDTDTextfield.setBounds(482, 99, 167, 20);
editProjectPane.add(editProjectPaneTLDTDTextfield);

JLabel editProjectPaneTLUC = new JLabel("Total lots under construction:  ");
editProjectPaneTLUC.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneTLUC.setBounds(316, 132, 167, 16);
editProjectPane.add(editProjectPaneTLUC);
```

```
JTextField editProjectPaneTLUCTextfield = new JTextField();
editProjectPaneTLUCTextfield.setColumns(10);
editProjectPaneTLUCTextfield.setBounds(482, 130, 167, 20);
editProjectPane.add(editProjectPaneTLUCTextfield);

JLabel editProjectPaneTLR = new JLabel("Total lots remaining:   ");
editProjectPaneTLR.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneTLR.setBounds(316, 163, 167, 16);
editProjectPane.add(editProjectPaneTLR);

JTextField editProjectPaneTLRTextfield = new JTextField();
editProjectPaneTLRTextfield.setColumns(10);
editProjectPaneTLRTextfield.setBounds(482, 161, 167, 20);
editProjectPane.add(editProjectPaneTLRTextfield);

JLabel editProjectPaneLSWTLTD = new JLabel("Lots sold within the last 30 days: ");
editProjectPaneLSWTLTD.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneLSWTLTD.setBounds(316, 193, 195, 16);
editProjectPane.add(editProjectPaneLSWTLTD);

JTextField editProjectPaneLSWTLTDTextfield = new JTextField();
editProjectPaneLSWTLTDTextfield.setColumns(10);
editProjectPaneLSWTLTDTextfield.setBounds(516, 191, 133, 20);
editProjectPane.add(editProjectPaneLSWTLTDTextfield);

JLabel editProjectPaneACPL = new JLabel("Average cost per lot:                $");
editProjectPaneACPL.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneACPL.setBounds(316, 225, 195, 16);
editProjectPane.add(editProjectPaneACPL);

JTextField editProjectPaneACPLTextfield = new JTextField();
editProjectPaneACPLTextfield.setColumns(10);
editProjectPaneACPLTextfield.setBounds(482, 222, 167, 20);
editProjectPane.add(editProjectPaneACPLTextfield);

JLabel editProjectPaneAPPL = new JLabel("Average sales price per lot:     $");
editProjectPaneAPPL.setFont(new Font("Times New Roman", Font.BOLD, 12));
editProjectPaneAPPL.setBounds(316, 250, 195, 16);
editProjectPane.add(editProjectPaneAPPL);

JTextField editProjectPaneAPPLFextfield = new JTextField();
editProjectPaneAPPLFextfield.setColumns(10);
editProjectPaneAPPLFextfield.setBounds(482, 247, 167, 20);
editProjectPane.add(editProjectPaneAPPLFextfield);

JButton editProjectPaneSaveButton = new JButton("Save");
editProjectPaneSaveButton.setBounds(634, 404, 98, 26);
editProjectPane.add(editProjectPaneSaveButton);
```

DisplayData has a second method, editProjectData(), used for editing projects:

```
    public void editProjectData(){
        StringTokenizer st = new StringTokenizer(lblProjectTitle.getText());
        String nameBuilder="";
        while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
        //^builds the project's name based on the project's title label
        File projectDataFile = new File(nameBuilder+"Data.txt");
        Scanner r1=null;
            try {r1 = new Scanner(projectDataFile);}
            catch (FileNotFoundException e) {e.printStackTrace();}
        //^makes a scanner for the project's data file
        editProjectPaneProjectNameTextfield.setText(r1.nextLine());
        editProjectPaneAdressTextfield.setText(r1.nextLine());
        editProjectPaneCityTextfield.setText(r1.nextLine());
        editProjectPaneStateTextfield.setText(r1.nextLine());
        editProjectPaneTLIDTextfield.setText(r1.nextLine());
        editProjectPaneTLDTDTextfield.setText(r1.nextLine());
        editProjectPaneTLUCTextfield.setText(r1.nextLine());
        editProjectPaneTLRTextfield.setText(r1.nextLine());
        editProjectPaneLSWTLTDTextfield.setText(r1.nextLine());
        editProjectPaneACPLTextfield.setText(r1.nextLine());
        editProjectPaneAPPLFextfield.setText(r1.nextLine());
        //^transfers each line in the file to the according textfield
        r1.close();

        editProjectPane.setVisible(true);
        projectPane.setVisible(false);
        //^sets the edit project pane visible, after the textfields have been filled
    }
} DisplayData displayData = new DisplayData();
```

A scanner is used to transfer the project's data into textfields so the user can edit them.

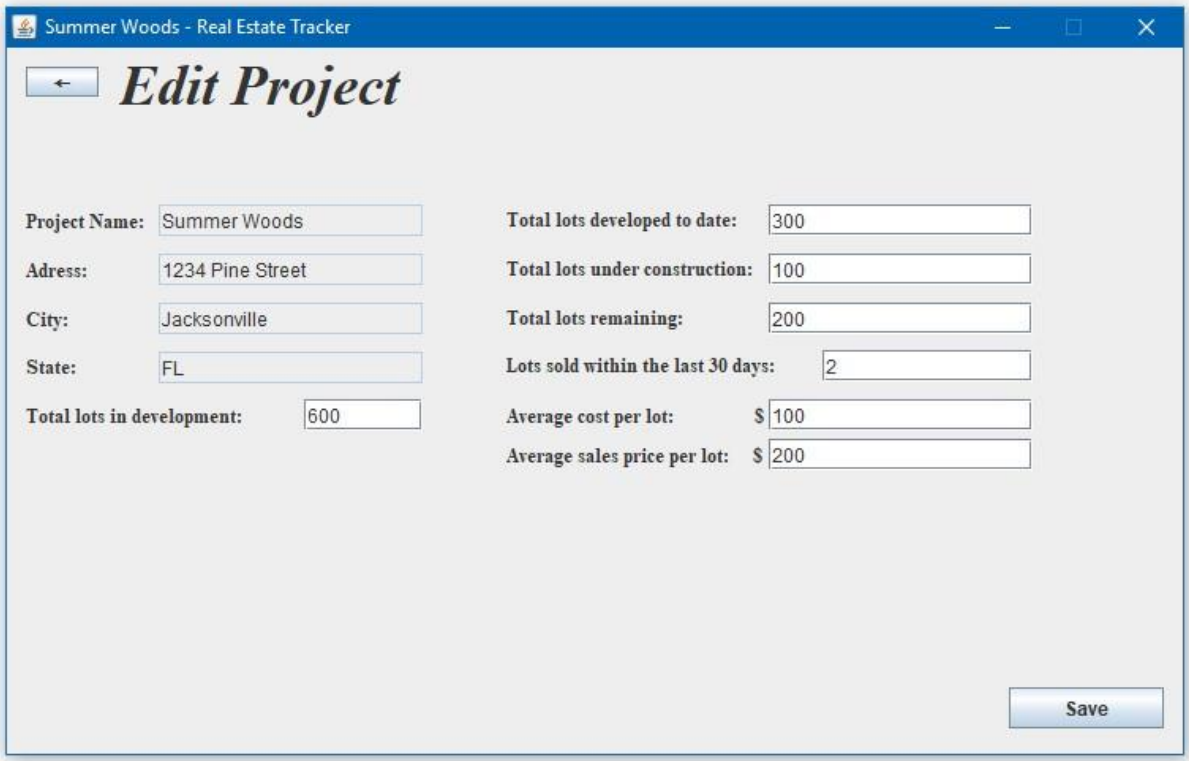


Figure 11: The edit project pane for the Summer Woods project. Notice how the editProjectPane() method fills the textfields with the project's existing data

When "Save" is clicked, the following action listener edits the project data file:

```
//save edits to project data
editProjectPaneSaveButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        boolean emptyTextFields = false;
        boolean noNumberTextFields = false; //sets all error booleans to false

        if(editProjectPaneProjectNameTextfield.getText().isEmpty() ||
                editProjectPaneAdressTextfield.getText().isEmpty() ||
                editProjectPaneCityTextfield.getText().isEmpty() ||
                editProjectPaneStateTextfield.getText().isEmpty() ||
                editProjectPaneTLIDTextfield.getText().isEmpty() ||
                editProjectPaneTLDTDTextfield.getText().isEmpty() ||
                editProjectPaneTLUCTextfield.getText().isEmpty() ||
                editProjectPaneTLRTextfield.getText().isEmpty() ||
                editProjectPaneLSWTLTDTextfield.getText().isEmpty() ||
                editProjectPaneACPLTextfield.getText().isEmpty() ||
                editProjectPaneAPPLFextfield.getText().isEmpty()) {
                //^checks if any textfields are left empty
                    emptyTextFields =  true;
                    System.out.println("ERROR: Empty text fields");
        }

        if(emptyTextFields==true) {
            JOptionPane.showMessageDialog(new JFrame(),
                    "All textboxes must be filled before revision",
                    "ERROR",
                    JOptionPane.WARNING_MESSAGE);
            //^runs a warning option pane, if textfield(s) are empty
        }

        if(emptyTextFields==false) { //runs if all textfields are full
            try {
                Integer.parseInt(editProjectPaneTLIDTextfield.getText());
                Integer.parseInt(editProjectPaneTLDTDTextfield.getText());
                Integer.parseInt(editProjectPaneTLUCTextfield.getText());
                Integer.parseInt(editProjectPaneTLRTextfield.getText());
                Integer.parseInt(editProjectPaneLSWTLTDTextfield.getText());
                Double.parseDouble(editProjectPaneACPLTextfield.getText());
                Double.parseDouble(editProjectPaneAPPLFextfield.getText());
                //^check if certain textfields contain numbers
            }
            catch(NumberFormatException e1) {
            //^runs if certain textfields do not contain numbers
                JOptionPane.showMessageDialog(new JFrame(),
                        "Please insert numbers where needed",
                        "ERROR",
                        JOptionPane.WARNING_MESSAGE);
                noNumberTextFields = true;
                //^runs a warning option pane, if textfield(s) dont have numbers
            }
        }
```

```
if(noNumberTextFields==false && emptyTextFields==false) {
//^runs if there have been no errors found
    StringTokenizer st = new StringTokenizer(lblProjectTitle.getText());
    String nameBuilder="";
    while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
    //^builds the project's name from the project title label
    File projectDataFile = new File(nameBuilder+"Data.txt");
    FileWriter dataWriter = null;
        try {dataWriter = new FileWriter(projectDataFile);}
        catch (IOException e1) {e1.printStackTrace();}
    PrintWriter dpw = new PrintWriter(dataWriter);
    //^makes a File and PrintWriter for the project's data file

    String newProjectPaneProjectNameTextfield = editProjectPaneProjectNameTextfield.getText();
        if(!newProjectPaneProjectNameTextfield.equals(checker.originalProjectPaneProjectNameTextfield)) {
            DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    "Project Name",checker.originalProjectPaneACPLTextfield,
                    newProjectPaneProjectNameTextfield);
            EventScripter.addEvent(temp);
        }
    String newProjectPaneAdressTextfield = editProjectPaneAdressTextfield.getText();
    if(!newProjectPaneAdressTextfield.equals(checker.originalProjectPaneAdressTextfield)) {
        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                "Project Address",checker.originalProjectPaneACPLTextfield,
                newProjectPaneAdressTextfield);
        EventScripter.addEvent(temp);
    }
    String newProjectPaneCityTextfield = editProjectPaneCityTextfield.getText();
    if(!newProjectPaneCityTextfield.equals(checker.originalProjectPaneCityTextfield)) {
        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                "Project City",checker.originalProjectPaneCityTextfield,
                newProjectPaneCityTextfield);
        EventScripter.addEvent(temp);
    }
    String newProjectPaneStateTextfield = editProjectPaneStateTextfield.getText();
    if(!newProjectPaneStateTextfield.equals(checker.originalProjectPaneStateTextfield)) {
        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                "Project State",checker.originalProjectPaneStateTextfield,
                newProjectPaneStateTextfield);
        EventScripter.addEvent(temp);
    }
    String newProjectPaneTLIDTextfield = editProjectPaneTLIDTextfield.getText();
    if(!newProjectPaneTLIDTextfield.equals(checker.originalProjectPaneTLIDTextfield)) {
        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                "Total Lots in Development",checker.originalProjectPaneTLIDTextfield,
                newProjectPaneTLIDTextfield);
        EventScripter.addEvent(temp);
    }
```

```
        String newProjectPaneTLDTDTextfield = editProjectPaneTLDTDTextfield.getText();
        if(!newProjectPaneTLDTDTextfield.equals(checker.originalProjectPaneTLDTDTextfield)) {
            DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    "Total Lots Developed to Date",checker.originalProjectPaneTLDTDTextfield,
                    newProjectPaneTLDTDTextfield);
            EventScripter.addEvent(temp);
        }
        String newProjectPaneTLUCTextfield = editProjectPaneTLUCTextfield.getText();
        if(!newProjectPaneTLUCTextfield.equals(checker.originalProjectPaneTLUCTextfield)) {
            DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    "Total Lots Under Construction",checker.originalProjectPaneTLUCTextfield,
                    newProjectPaneTLUCTextfield);
            EventScripter.addEvent(temp);
        }
        String newProjectPaneTLRTextfield = editProjectPaneTLRTextfield.getText();
        if(!newProjectPaneTLRTextfield.equals(checker.originalProjectPaneTLRTextfield)) {
            DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    "Total Lots Remaining",checker.originalProjectPaneTLRTextfield,
                    newProjectPaneTLRTextfield);
            EventScripter.addEvent(temp);
        }
        String newProjectPaneLSWTLTDTextfield = editProjectPaneLSWTLTDTextfield.getText();
        if(!newProjectPaneLSWTLTDTextfield.equals(checker.originalProjectPaneLSWTLTDTextfield)) {
            DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    "Total Lots Sold Within the Last 30 Days",checker.originalProjectPaneLSWTLTDTextfield,
                    newProjectPaneLSWTLTDTextfield);
            EventScripter.addEvent(temp);
        }
        String newProjectPaneACPLTextfield = editProjectPaneACPLTextfield.getText();
        if(!newProjectPaneACPLTextfield.equals(checker.originalProjectPaneACPLTextfield)) {
            DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    "Average Cost Per Lot",checker.originalProjectPaneACPLTextfield,
                    newProjectPaneACPLTextfield);
            EventScripter.addEvent(temp);
        }
        String newProjectPaneAPPLFextfield = editProjectPaneAPPLFextfield.getText();
        if(!newProjectPaneAPPLFextfield.equals(checker.originalProjectPaneAPPLFextfield)) {
            DataEvent temp = new DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    "Average Cost Per Lot",checker.originalProjectPaneAPPLFextfield,
                    newProjectPaneAPPLFextfield);
            EventScripter.addEvent(temp);
        }

            dpw.println(editProjectPaneProjectNameTextfield.getText());
            dpw.println(editProjectPaneAdressTextfield.getText());
            dpw.println(editProjectPaneCityTextfield.getText());
            dpw.println(editProjectPaneStateTextfield.getText());
            dpw.println(editProjectPaneTLIDTextfield.getText());
            dpw.println(editProjectPaneTLDTDTextfield.getText());
            dpw.println(editProjectPaneTLUCTextfield.getText());
            dpw.println(editProjectPaneTLRTextfield.getText());
            dpw.println(editProjectPaneLSWTLTDTextfield.getText());
            dpw.println(editProjectPaneACPLTextfield.getText());
            dpw.println(editProjectPaneAPPLFextfield.getText());
            //^prints the new data values, replacing the older ones
            dpw.close();

            editProjectPane.setVisible(false);
            projectPane.setVisible(true);
            //^takes the client back to the respective project's detail pane
            displayData.projectData();
            //^reruns the projectData() method to refresh the detail labels
        }
    }
});
```

Noticeably, DataEvents are created for each changed value.

ChangeChecker is also used to capture changes in values for the subsequent DataEvents. All instance variables are purposefully public.

```
class ChangeChecker{
    public String originalProjectPaneProjectNameTextfield;
    public String originalProjectPaneAdressTextfield;
    public String originalProjectPaneCityTextfield;
    public String originalProjectPaneStateTextfield;
    public String originalProjectPaneTLIDTextfield;
    public String originalProjectPaneTLDTDTextfield;
    public String originalProjectPaneTLUCTextfield;
    public String originalProjectPaneTLRTextfield;
    public String originalProjectPaneLSWTLTDTextfield;
    public String originalProjectPaneACPLTextfield;
    public String originalProjectPaneAPPLFextfield;

    public void captureOriginalDataValues() {
        originalProjectPaneProjectNameTextfield = editProjectPaneProjectNameTextfield.getText();
        originalProjectPaneAdressTextfield = editProjectPaneAdressTextfield.getText();
        originalProjectPaneCityTextfield = editProjectPaneCityTextfield.getText();
        originalProjectPaneStateTextfield = editProjectPaneStateTextfield.getText();
        originalProjectPaneTLIDTextfield = editProjectPaneTLIDTextfield.getText();
        originalProjectPaneTLDTDTextfield = editProjectPaneTLDTDTextfield.getText();
        originalProjectPaneTLUCTextfield = editProjectPaneTLUCTextfield.getText();
        originalProjectPaneTLRTextfield = editProjectPaneTLRTextfield.getText();
        originalProjectPaneLSWTLTDTextfield = editProjectPaneLSWTLTDTextfield.getText();
        originalProjectPaneACPLTextfield = editProjectPaneACPLTextfield.getText();
        originalProjectPaneAPPLFextfield = editProjectPaneAPPLFextfield.getText();
    }
} ChangeChecker checker = new ChangeChecker();
```

Additionally, pressing the “back” button returns the user to the project menu:

```
editProjectPaneEditButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        editProjectPane.setVisible(false);
        projectPane.setVisible(true);
        //^returns to the project pane
        displayData.projectData();
        //^refreshes the project data
    }
});
```

Project Pane (Tasks)

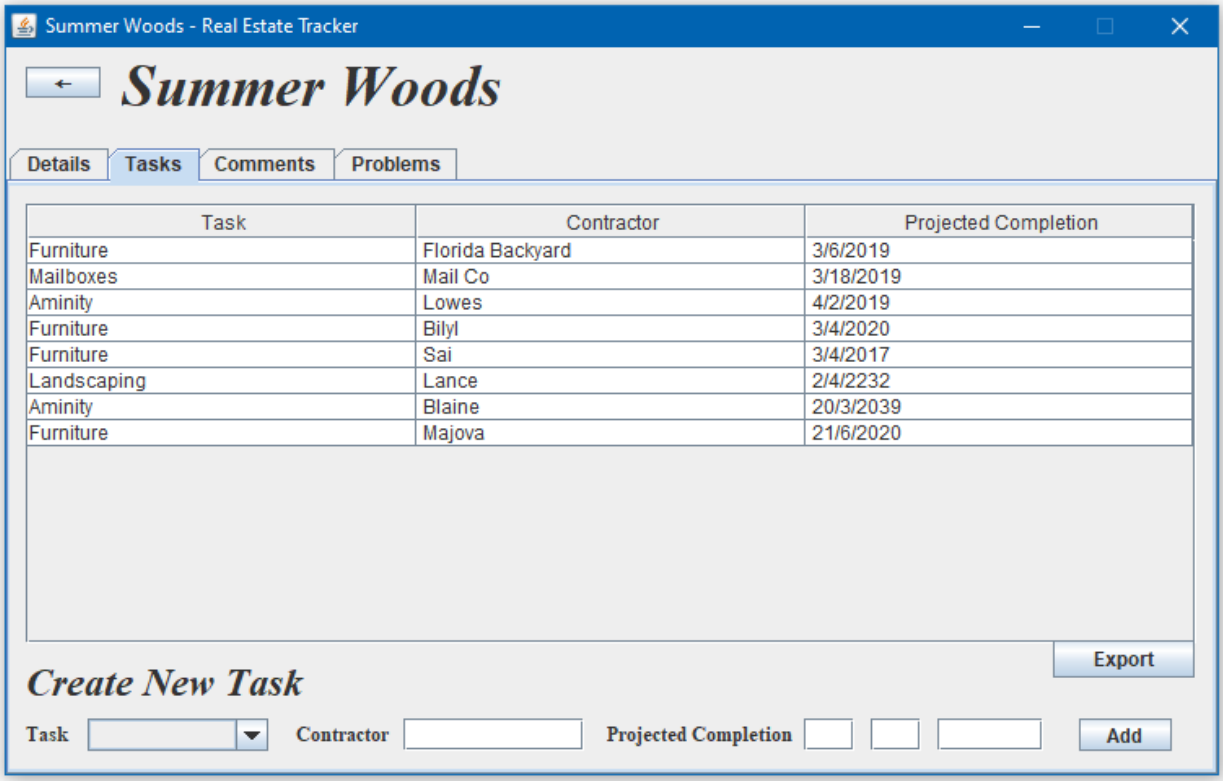


Figure 12: The tasks pane for the Summer Woods project

Components Used

```
JPanel tasksPanel = new JPanel();
tabbedPane.addTab("Tasks", null, tasksPanel, null);
tasksPanel.setLayout(null);

JLabel lblCreateNewTask = new JLabel("Create New Task");
lblCreateNewTask.setBounds(10, 291, 206, 28);
lblCreateNewTask.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 24));
tasksPanel.add(lblCreateNewTask);

JLabel lblTask = new JLabel("Task");
lblTask.setBounds(10, 330, 46, 14);
lblTask.setFont(new Font("Times New Roman", Font.BOLD, 12));
tasksPanel.add(lblTask);

JLabel lblContractor = new JLabel("Contractor");
lblContractor.setBounds(176, 330, 74, 14);
lblContractor.setFont(new Font("Times New Roman", Font.BOLD, 12));
tasksPanel.add(lblContractor);

JLabel lblProjectedCompletion = new JLabel("Projected Completion");
lblProjectedCompletion.setBounds(367, 330, 125, 14);
lblProjectedCompletion.setFont(new Font("Times New Roman", Font.BOLD, 12));
tasksPanel.add(lblProjectedCompletion);

JComboBox tasksComboBox = new JComboBox();
tasksComboBox.setModel(new DefaultComboBoxModel(new String[]
        {" ", "Aminity", "Furniture", "Landscaping", "Mailboxes"}));
tasksComboBox.setBounds(48, 327, 111, 20);
tasksPanel.add(tasksComboBox);

JTextField contractorTextField = new JTextField();
contractorTextField.setBounds(242, 327, 111, 20);
tasksPanel.add(contractorTextField);
contractorTextField.setColumns(10);

JTextField projCompMMTextField = new JTextField();
projCompMMTextField.setBounds(488, 327, 31, 20);
tasksPanel.add(projCompMMTextField);
projCompMMTextField.setColumns(10);
```

```
JTextField projCompDDTextField = new JTextField();
projCompDDTextField.setColumns(10);
projCompDDTextField.setBounds(529, 327, 31, 20);
tasksPanel.add(projCompDDTextField);

JTextField projCompYYYYTextField = new JTextField();
projCompYYYYTextField.setColumns(10);
projCompYYYYTextField.setBounds(570, 327, 65, 20);
tasksPanel.add(projCompYYYYTextField);

JButton btnExport = new JButton("Export");
btnExport.setBounds(641, 279, 87, 23);
tasksPanel.add(btnExport);

JButton btnAdd = new JButton("Add");
btnAdd.setBounds(657, 327, 57, 20);
tasksPanel.add(btnAdd);

String[] columnNames = {"Task","Contractor","Projected Completion"};
DefaultTableModel tasksTableModel = new DefaultTableModel(columnNames,0){
    //makes cells uneditable by overriding DefaultTableModel's isCellEditable() method
    public boolean isCellEditable(int rows, int columns) {return false;}
};

JTable tasksTable = new JTable(tasksTableModel);
tasksTable.setBounds(10, 11, 719, 270);
tasksTable.getTableHeader().setReorderingAllowed(false);
tasksPanel.add(tasksTable);

JScrollPane scrollpane = new JScrollPane(tasksTable);
scrollpane.setBounds(10, 11, 719, 270);
tasksPanel.add(scrollpane);
```

When the project pane is opened, the generateData() method from the TasksTable class runs. TasksTable extends the abstract class TableManager and implements the DataTable interface.

```
public interface DataTable {
    public void generateData();
    public void addData();
}
```

```
abstract class TableManager implements DataTable{
    public String getFileName(){
        StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
        String fileName="";
        while(st.hasMoreTokens()) fileName+=st.nextToken();
        return fileName;
    }
    public void generateData(){}
    public void addData() {}
}
```

```
//creates the table for the tasks page and adds new values to the table
class TasksTable extends TableManager implements DataTable{
    public void generateData() {

        //removes the preexisting table from another project
        tasksTableModel.setRowCount(0);

        File csvFile = new File(super.getFileName()+"Tasks.csv");
        Scanner reader=null;
            try {reader = new Scanner(csvFile);}
            catch (FileNotFoundException e) {e.printStackTrace();}

        //creates the data ArrayList
        ArrayList<String> dataList = new ArrayList<String>();

        //reads through the header values (they have already been set and only...
        //...exist in the text file for exporting reasons)
        reader.nextLine();

        //reads the actual task data
        while(reader.hasNextLine()) dataList.add(reader.nextLine());

        //converts from an ArrayList to a normal array
        String[] dataArray = new String[dataList.size()];
        for(int x=0;x<dataArray.length;x++) dataArray[x] = dataList.get(x);

        //adds the data to the JTable
        for(int x=0;x<dataArray.length;x++){
            String[] rowData = new String[3];
            int rowIndex = 0;
            StringTokenizer st2 = new StringTokenizer(dataArray[x], ",");
            while(st2.hasMoreTokens()) {
                rowData[rowIndex] = st2.nextToken();
                rowIndex++;}
            tasksTableModel.addRow(rowData); //uses defaultTableModel
        }
        reader.close();
    }
```

generateData() creates the task table.

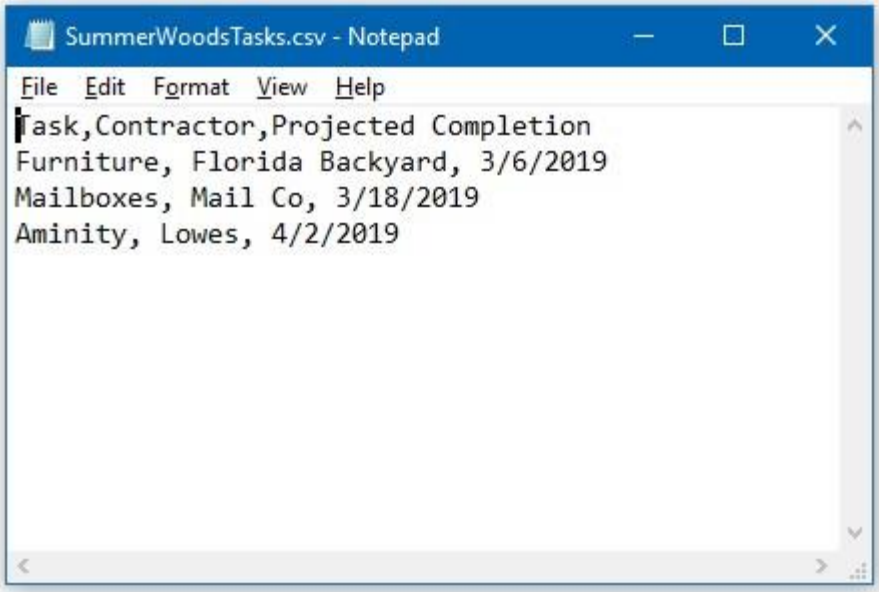


Figure 13: The tasks CSV file for the project Summer Woods

The client adds a task by clicking “add”, and exports the tasks file by clicking “export”. The following action listers run respectfully:

```
//adds the new task to the task table
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        String typeOfTask = tasksComboBox.getSelectedItem().toString();
        String contractor = contractorTextField.getText();
        String date = projCompMMTextField.getText() + "/" + projCompDDTextField.
                projCompYYYYTextField.getText();

        boolean emptyTextFields = false;
        boolean noNumberTextFields = false; //sets all error booleans to false

        if(projCompYYYYTextField.getText().isEmpty() ||
            projCompDDTextField.getText().isEmpty() ||
            projCompMMTextField.getText().isEmpty() ||
            contractorTextField.getText().isEmpty() ||
            tasksComboBox.getSelectedItem().equals(" ")) {
            //^runs if textfield(s) are left empty
                emptyTextFields =  true;
                System.out.println("ERROR: Empty text fields");
        }

        if(emptyTextFields==true) {
        //^runs if textfield(s) are left empty
            JOptionPane.showMessageDialog(new JFrame(),
                    "All textboxes must be filled before submission",
                    "ERROR",
                    JOptionPane.WARNING_MESSAGE);
            //^A warning option pane notifies the user of the blank textfields
        }

        if(emptyTextFields==false) {
        //^runs if all textfields are filled
            try {
                Integer.parseInt(projCompYYYYTextField.getText());
                Integer.parseInt(projCompDDTextField.getText());
                Integer.parseInt(projCompMMTextField.getText());
                //^tests if certain textfields are numbers
            }
```

```
            catch(NumberFormatException e1) {
            //^runs if those certian textfields are not numbers
                JOptionPane.showMessageDialog(new JFrame(),
                        "Please insert a valid date",
                        "ERROR",
                        JOptionPane.WARNING_MESSAGE);
                noNumberTextFields = true;
                //^runs a warning option pane notifying the client of the mistake
            }
        }
        if(noNumberTextFields==false && emptyTextFields==false) {
        //^runs if no errors have been found
            taskTable.addData(); //adds the new task
            clear.newTasks(); //clears the textfields on the tasks pane

            TaskEvent temp = new TaskEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                    typeOfTask, contractor, date);

            EventScripter.addEvent(temp);
        }
    }
});

//exports the task csv file by opening the file in notepad
btnExport.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        StringTokenizer st = new StringTokenizer(lblProjectTitle.getText());
        String nameBuilder="";
        while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
            try {Runtime.getRuntime().exec("notepad "+nameBuilder+"Tasks.csv");}
            catch (IOException e) {e.printStackTrace();}
    }
});
```

- "Add" checks for errors, and if none are found, then addData() from the TasksTable class runs.
- "Export" runs the project's task csv file so the client can export the data into other applications

```
    public void addData() {
        //gets the project's name
        StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
        String fileName="";
        while(st.hasMoreTokens()) fileName+=st.nextToken();

        //makes the file and filewriter and printwriter
        File csvFile = new File(fileName+"Tasks.csv");
        FileWriter fileWriter = null;
            try {fileWriter = new FileWriter(csvFile, true);}
            catch (IOException e1) {e1.printStackTrace();}
        PrintWriter printWriter = new PrintWriter(fileWriter);

        //writes the new task to the text file
        String task = (String) tasksComboBox.getSelectedItem();
        String contractor = contractorTextField.getText();
        String date = projCompMMTextField.getText() +"/"+
                      projCompDDTextField.getText() +"/"+
                      projCompYYYYTextField.getText();
        printWriter.print("\r\n");
        printWriter.print(task +", "+ contractor +", "+ date);
        printWriter.close();

        //adds the data to the JTable
        String[] rowData = {task, contractor, date};
        tasksTableModel.addRow(rowData);

    }
} DataTable taskTable = new TasksTable();
```

Project Pane (Comments)

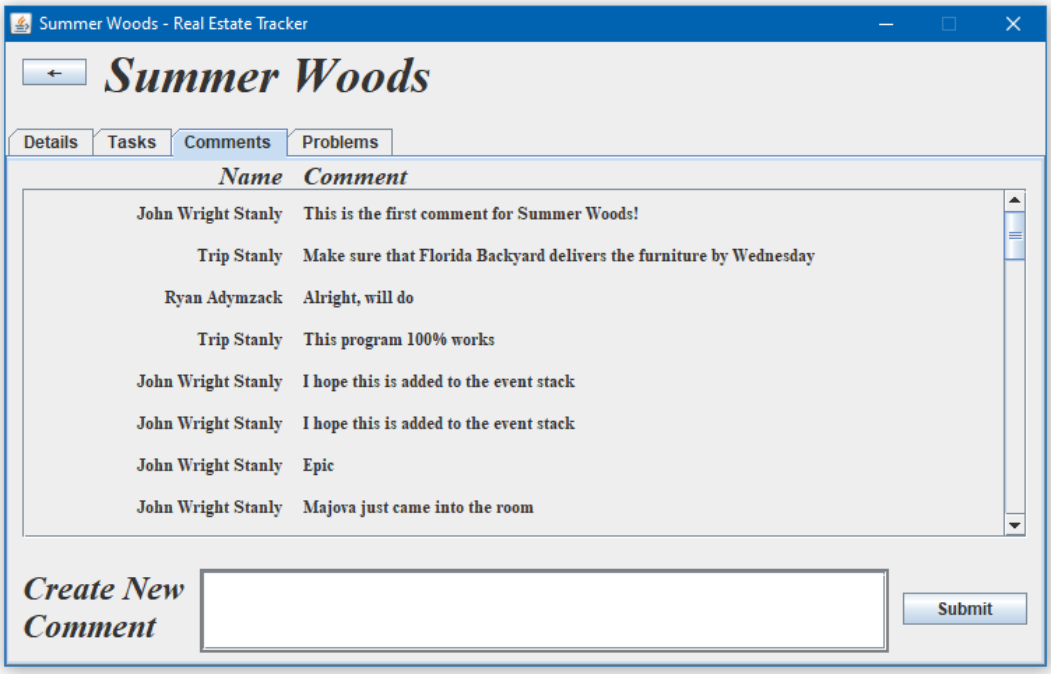


Figure 14: The comments pane for the project Summer Woods

Components Used

```
JPanel commentsPanel = new JPanel();
tabbedPane.addTab("Comments", null, commentsPanel, null);
commentsPanel.setLayout(null);

JLabel lblCreateNewComment = new JLabel("Create New");
lblCreateNewComment.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 24));
lblCreateNewComment.setBounds(10, 292, 128, 28);
commentsPanel.add(lblCreateNewComment);

JLabel lblCreateNewCommentpt2 = new JLabel("Comment");
lblCreateNewCommentpt2.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 24));
lblCreateNewCommentpt2.setBounds(10, 319, 128, 28);
commentsPanel.add(lblCreateNewCommentpt2);

JButton btnSubmit_1 = new JButton("Submit");
btnSubmit_1.setBounds(640, 309, 89, 23);
commentsPanel.add(btnSubmit_1);

JPanel commentBorderPanel = new JPanel();
commentBorderPanel.setBackground(Color.GRAY);
commentBorderPanel.setBounds(137, 292, 493, 60);
commentsPanel.add(commentBorderPanel);
commentBorderPanel.setLayout(null);

JTextArea commentTextArea = new JTextArea();
commentTextArea.setBounds(2, 2, 489, 56);
//commentBorderPanel.add(commentTextArea);
commentTextArea.setBackground(Color.WHITE);
commentTextArea.setLineWrap(true);
    JScrollPane commentTextareaScrollPane =
            new JScrollPane(commentTextArea,
                            ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
                            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
    commentTextareaScrollPane.setBounds(2, 2, 489, 56);
    commentBorderPanel.add(commentTextareaScrollPane);


JLabel lblName = new JLabel("Name");
lblName.setHorizontalAlignment(SwingConstants.TRAILING);
lblName.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 18));
lblName.setBounds(10, 1, 186, 20);
commentsPanel.add(lblName);
```

```
JLabel lblComment = new JLabel("Comment");
lblComment.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 18));
lblComment.setBounds(211, 1, 158, 20);
commentsPanel.add(lblComment);

int numberOfCommentsSupported = 50; //sets the max comments viewed at once

    JPanel commentBoardPanel = new JPanel();
    commentBoardPanel.setLayout(null);
    commentBoardPanel.setBounds(10, 20, 719, 20+(30*numberOfCommentsSupported));
    //^dynamically sets the size to the max comments supported
    commentBoardPanel.setPreferredSize(new Dimension(719,(20+(30*numberOfCommentsSupported))));

    JLabel[] nameLabelArray = new JLabel[numberOfCommentsSupported];
    JLabel[] commentLabelArray = new JLabel[numberOfCommentsSupported];
    //^arrays' physical size only supports the max amount of comments
        for(int x=0;x<numberOfCommentsSupported;x++) {
            nameLabelArray[x] = new JLabel("This is a name #"+x);
            nameLabelArray[x].setBounds(10, 10+(30*x), 175, 15);
            nameLabelArray[x].setFont(new Font("Times New Roman", Font.BOLD, 12));
            nameLabelArray[x].setHorizontalAlignment(SwingConstants.TRAILING);
            nameLabelArray[x].setVisible(false);
            commentBoardPanel.add(nameLabelArray[x]);
            //^creates array of name labels

            commentLabelArray[x] = new JLabel("This is a comment #"+x);
            commentLabelArray[x].setBounds(200, 10+(30*x), 500, 15);
            commentLabelArray[x].setFont(new Font("Times New Roman", Font.BOLD, 12));
            commentLabelArray[x].setVisible(false);
            commentBoardPanel.add(commentLabelArray[x]);
            //^creates array of comment labels
        }

    JScrollPane commentGridScrollPane = new JScrollPane(commentBoardPanel,
                            ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
    commentGridScrollPane.setBounds(10, 20, 719, 250);
    commentGridScrollPane.setPreferredSize(new Dimension(719,250));
    commentsPanel.add(commentGridScrollPane);
```

When a project is selected from the main menu, the generateCommentBoard() method from the CommentBoard class runs.

```
class CommentBoard{
    public void generateCommentBoard() {
        //creates array lists of the ID's of the accounts and the comments themselves
        ArrayList<Integer> indexArray = new ArrayList<Integer>();
        ArrayList<String> commentArray = new ArrayList<String>();

        //reads the text file
        StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
        String fileName="";
        while(st.hasMoreTokens()) fileName+=st.nextToken();
        File commentsFile = new File(fileName+"Comments.txt");
        Scanner reader=null;
            try {reader = new Scanner(commentsFile);}
            catch (FileNotFoundException e) {e.printStackTrace();}
        //^makes a scanner for the project's comments file
        while(reader.hasNextLine()) {
            //each line in the text file has an ID and a comment
            StringTokenizer st2 = new StringTokenizer(reader.nextLine(),";");
            //^account indexes and comments are split by the semi-colon delimiter
            indexArray.add(Integer.parseInt(st2.nextToken()));
            commentArray.add(st2.nextToken());
        }

        //erases previous comments from other opened projects
        for(int x=0;x<numberOfCommentsSupported;x++) {
            nameLabelArray[x].setText("");
            nameLabelArray[x].setVisible(false);
            commentLabelArray[x].setText("");
            commentLabelArray[x].setVisible(false);
        }

        //converts the information from the array lists into JLabels
        for(int x=0;x<indexArray.size();x++) {
            nameLabelArray[x].setText(""+account[indexArray.get(x)].getName());
            //^finds the respective name of the account index recorded
            nameLabelArray[x].setVisible(true);
            commentLabelArray[x].setText(""+commentArray.get(x));
            commentLabelArray[x].setToolTipText(""+commentArray.get(x));
            commentLabelArray[x].setVisible(true);}
    }
```

The method reads the comments file, stores the indexes and comments into ArrayLists, and transfers the ArrayList values into name and comment JLabels.

Figure 15: The comments text file for the project Summer Woods

When “Submit” is clicked, the following action listener runs:

```
//saves a new comment made
btnSubmit_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        boolean emptyTextFields = false; //sets error boolean to false

        if(commentTextArea.getText().isEmpty()) {
                emptyTextFields =  true;
                System.out.println("ERROR: Empty text fields");
        }
        if(emptyTextFields==true) { //runs if the comment area is left blank
            JOptionPane.showMessageDialog(new JFrame(),
                    "You must write a comment before you submit one!",
                    "ERROR",
                    JOptionPane.WARNING_MESSAGE);
            //^runs a warning option pane informing the client of the problem
        }
        if(emptyTextFields==false) { //runs if there are no errors
            commentBoard.addComment();
        }
    }
});
```

If no errors are found, the action listener invokes addComment(), another method found in the CommentBoard class.

```
public void addComment() {
    //adds the new comment to the comment text file
    StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
    String fileName="";
    while(st.hasMoreTokens()) fileName+=st.nextToken();
    //^builds the project's name based on the project's title label
    File commentsFile = new File(fileName+"Comments.txt");
    FileWriter dataWriter = null;
        try {dataWriter = new FileWriter(commentsFile, true);} //supports amending
        catch (IOException e1) {e1.printStackTrace();}
    PrintWriter dpw = new PrintWriter(dataWriter);
    //^creates a File and PrintWriter for the project's comments file
    dpw.print(""+account[accountIndexUsed].getIndex()+";"+commentTextArea.getText());
    //^prints the client's index and the comment itself, using the semi-colon delimiter
    dpw.println("");
    dpw.close();
    commentTextArea.setText("");
    //^resets the comment's textarea without invoking a Clear method

    generateCommentBoard(); //regenerates the comment board by invoking the method
}
```

addComment() records the client’s index and comment using a PrintWriter, and they are split using a semi colon (which a StringTokenizer in generateCommentBoard() uses as a delimiter).

Project Pane (Problems)

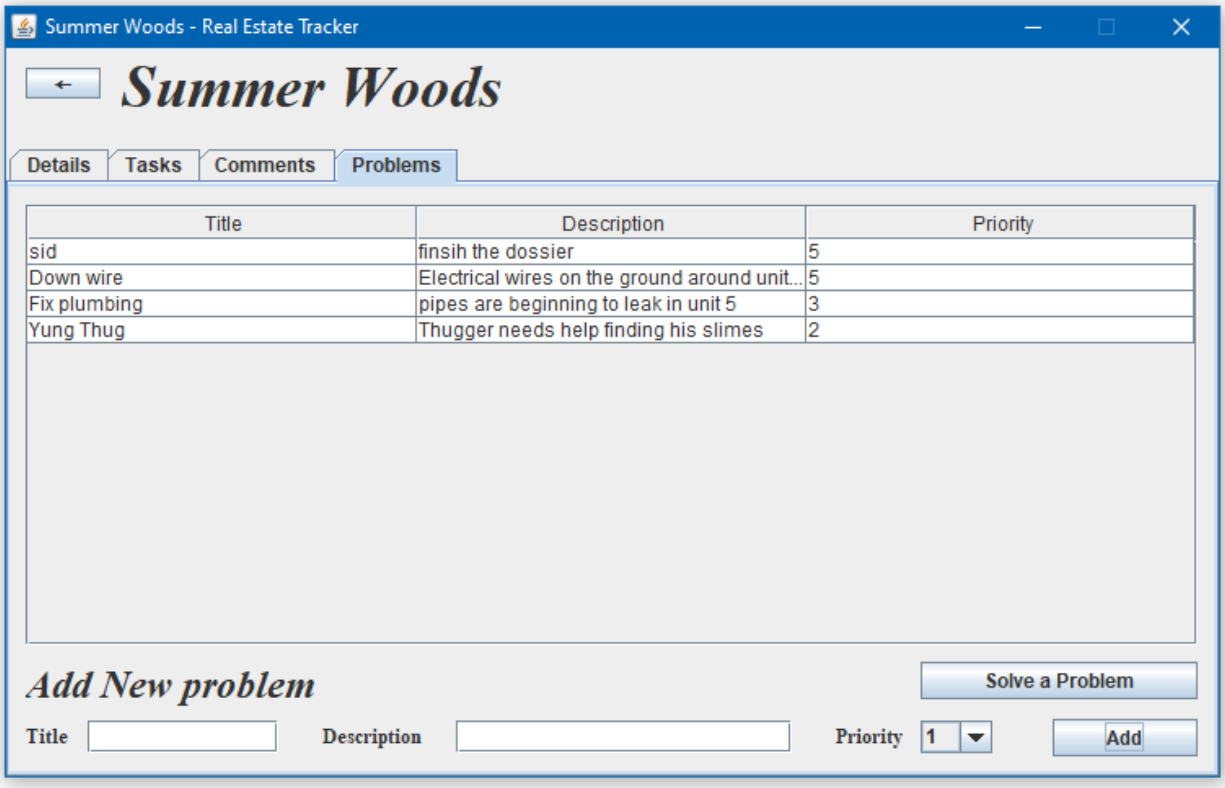


Figure 16: The project pane for the project Summer Woods

Components Used

```
JLabel lblCreateNewproblem = new JLabel("Add New problem");
lblCreateNewproblem.setBounds(10, 291, 206, 28);
lblCreateNewproblem.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 24));
problemsPanel.add(lblCreateNewproblem);

JLabel lblproblem = new JLabel("Title");
lblproblem.setBounds(10, 330, 46, 14);
lblproblem.setFont(new Font("Times New Roman", Font.BOLD, 12));
problemsPanel.add(lblproblem);

String[] problemColumnNames = {"Title","Description","Priority"};
DefaultTableModel problemsTableModel = new DefaultTableModel(problemColumnNames,0){
    //makes cells uneditable by ovveriding DefaultTableModel's isCellEditable() method
    public boolean isCellEditable(int rows, int columns) {return false;}
};

JTable problemsTable = new JTable(problemsTableModel);
problemsTable.setBounds(10, 11, 719, 270);
problemsTable.getTableHeader().setReorderingAllowed(false);
problemsPanel.add(problemsTable);

JScrollPane scrollpaneProblems = new JScrollPane(problemsTable);
scrollpaneProblems.setBounds(10, 11, 719, 270);
problemsPanel.add(scrollpaneProblems);

JTextField titleTextField = new JTextField();
titleTextField.setBounds(48, 327, 117, 20);
problemsPanel.add(titleTextField);
titleTextField.setColumns(10);

JLabel lblDescription = new JLabel("Description");
lblDescription.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblDescription.setBounds(192, 330, 72, 14);
problemsPanel.add(lblDescription);

JTextField descriptionTextField = new JTextField();
descriptionTextField.setBounds(274, 327, 206, 20);
problemsPanel.add(descriptionTextField);
descriptionTextField.setColumns(10);

JLabel lblPriority = new JLabel("Priority");
lblPriority.setFont(new Font("Times New Roman", Font.BOLD, 12));
lblPriority.setBounds(507, 330, 52, 14);
problemsPanel.add(lblPriority);

JComboBox priorityComboBox = new JComboBox();
priorityComboBox.setModel(new DefaultComboBoxModel(new String[] {"1", "2", "3", "4", "5"}));
priorityComboBox.setBounds(559, 327, 44, 20);
problemsPanel.add(priorityComboBox);

JButton btnAddProblem = new JButton("Add");
btnAddProblem.setBounds(640, 326, 89, 23);
problemsPanel.add(btnAddProblem);

JButton btnRecommendAProblem = new JButton("Solve a Problem");
btnRecommendAProblem.setBounds(559, 291, 170, 23);
problemsPanel.add(btnRecommendAProblem);
```

Individual problems are encapsulated into instances of the Problem class and implement a comparable interface.

```
public class Problem implements Comparable<Problem>{
    private int priority;
    private String title;
    private String description;

    public Problem(){
        priority = 1;
        title = "NA";
        description = "";
    }

    public Problem(String s, String s2, int n){
        priority = n;
        title = s;
        description = s2;
    }

    public Problem(String s, int n){
        priority = n;
        title = s;
        description = "";
    }

    public String toString(){
        return "TITLE: " + title + " ... PRIORITY: " + priority;
    }

    public void setPriority(int n){
        priority = n;}

    public int getPriority(){
        return priority;}

    public void setTitle(String n){
        title = n;}

    public String getTitle(){
        return title;}

    public void setDescription(String n){
        description = n;}

    public String getDescription(){
        return description;}


    public int compareTo(Problem anotherProblem){
        return this.getPriority() - anotherProblem.getPriority();
    }
}
```

Problem instances are then stored in a specially designed problem priority queue PQueue.

```
public class PQueue{

    private Problem[] heap;
    private int heapSize;
    private int capacity;

    public PQueue(int capacity){
        this.capacity = capacity + 1;
        heap = new Problem[this.capacity];
        heapSize = 0;
    }

    public void clear(){
        heap = new Problem[capacity];
        heapSize = 0;
    }

    public boolean isEmpty(){
        return heapSize == 0;
    }

    public boolean isFull(){
        return heapSize == capacity - 1;
    }

    public int size(){
        return heapSize;
    }

    public void enqueue(Problem passedProblem){
        heap[++heapSize] = passedProblem;
        int pos = heapSize;
        while(pos != 1 && passedProblem.getPriority() > heap[pos/2].getPriority()){
            heap[pos] = heap[pos/2];
            pos /=2;
        }
        heap[pos] = passedProblem;
    }
```

```
    public Problem dequeue(){
        int parent, child;
        Problem item, temp;
        if (isEmpty()){
            System.out.println("Problem heap is empty");
            return null;
        }

        item = heap[1];
        temp = heap[heapSize--];

        parent = 1;
        child = 2;
        while (child <= heapSize){
            if(child < heapSize && heap[child].getPriority() < heap[child + 1].getPriority()) {
                child++;
            }
            if(temp.getPriority() >= heap[child].getPriority()) {
                break;
            }

            heap[parent] = heap[child];
            parent = child;
            child *= 2;
        }
        heap[parent] = temp;

        return item;
    }

    public Problem front(){
        return heap[1];
    }

    public Problem rear(){
        return heap[heapSize];
    }

    public String toString(){
        String output = null;
        for(int x=0;x<heap.length;x++) {
            output+=heap[x].toString()+"\n";
        }
        return output;
    }
}
```

ProblemsTable first generates the queue from the CSV through the following method. ProblemsTable also impliments DataTable and extends TableManager.

```
//creates the table for the tasks page and adds new values to the table
class ProblemsTable extends TableManager implements DataTable{

    private PQueue problemQueue;

    public void generateProblemQueue() {
        //removes the preexisting table from another project
        problemsTableModel.setRowCount(0);

        //reads the tasks csv file
        File csvFile = new File(super.getFileName()+"Problems.csv");
        Scanner reader=null;
            try {reader = new Scanner(csvFile);}
            catch (FileNotFoundException e) {e.printStackTrace();}

        //creates the data ArrayList
        ArrayList<String> dataList = new ArrayList<String>();

        //reads through the header values (they have already been set and only...
        //...exist in the text file for exporting reasons)
        reader.nextLine();

        //reads the actual task data
        while(reader.hasNextLine()) dataList.add(reader.nextLine());

        //converts from an ArrayList to a normal array
        String[] dataArray = new String[dataList.size()];
        for(int x=0;x<dataArray.length;x++) dataArray[x] = dataList.get(x);

        //creates an array of Problem instances
        problemQueue = new PQueue(dataList.size()*3);

        //adds the data to the Problem Queue
        for(int x=0;x<dataArray.length;x++){
            String[] rowData = new String[3];
            int rowIndex = 0;
            StringTokenizer st2 = new StringTokenizer(dataArray[x], ",");
            String title = st2.nextToken();
            String description = st2.nextToken();
            int priority = Integer.parseInt(st2.nextToken());
            problemQueue.enqueue(new Problem(title, description, priority));
        }
        reader.close();
    }
```

The GUI's table is then uploaded through generateData();

```
public void generateData() {

    generateProblemQueue();

    //adds the data to the JTable
    while(!problemQueue.isEmpty()){
        Problem temp = problemQueue.dequeue();
        String[] rowData = new String[3];
        rowData[0] = temp.getTitle();
        rowData[1] = temp.getDescription();
        rowData[2] = ""+temp.getPriority();
        problemsTableModel.addRow(rowData); //uses defaultTableModel
    }
}
```

Problems are added through clicking the “Add” button, triggering the following action listener

```
//adds a new problem
btnAddProblem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        String title = titleTextField.getText();
        String description = descriptionTextField.getText();
        String priority = priorityComboBox.getSelectedItem().toString();

        boolean emptyTextFields = false;

        if(titleTextField.getText().isEmpty() || descriptionTextField.getText().isEmpty()) {
        //^runs if textfield(s) are left empty
            emptyTextFields =  true;
            System.out.println("ERROR: Empty text fields");
        }

        if(emptyTextFields==true) {
        //^runs if textfield(s) are left empty
            JOptionPane.showMessageDialog(new JFrame(),
                    "All textboxes must be filled before submission",
                    "ERROR",
                    JOptionPane.WARNING_MESSAGE);
            //^A warning option pane notifies the user of the blank textfields
        }

        if(emptyTextFields==false) {
        //^runs if no errors have been found
            problemTable.addData(); //adds the new task
            clear.newProblems(); //clears the textfields on the tasks pane

            ProblemEvent temp = new ProblemEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                                title, description, priority, 0);

            EventScripter.addEvent(temp);
        }

    }
});
```

ProblemsTable’s method addData() is called in the action listener above.

```
public void addData() {
    //makes the file and filewriter and printwriter
    File csvFile = new File(super.getFileName()+"Problems.csv");
    FileWriter fileWriter = null;
        try {fileWriter = new FileWriter(csvFile, true);}
        catch (IOException e1) {e1.printStackTrace();}
    PrintWriter printWriter = new PrintWriter(fileWriter);

    //writes the new task to the text file
    String title = (String) titleTextField.getText();
    String description = (String) descriptionTextField.getText();
    int priority = Integer.parseInt((String) priorityComboBox.getSelectedItem());
    printWriter.print("\r\n");
    printWriter.print(title +","+ description +","+ priority);
    printWriter.close();

    //adds the data to the JTable simply by adding to the row (not by actually enqueue/dequeue'ing the queue)
    //String[] rowData = {title, description, ""+priority};
    //problemsTableModel.addRow(rowData);

    //adds the data to the JTable by actually enqueue/dequeue'ing the queue
    generateData();
}
```

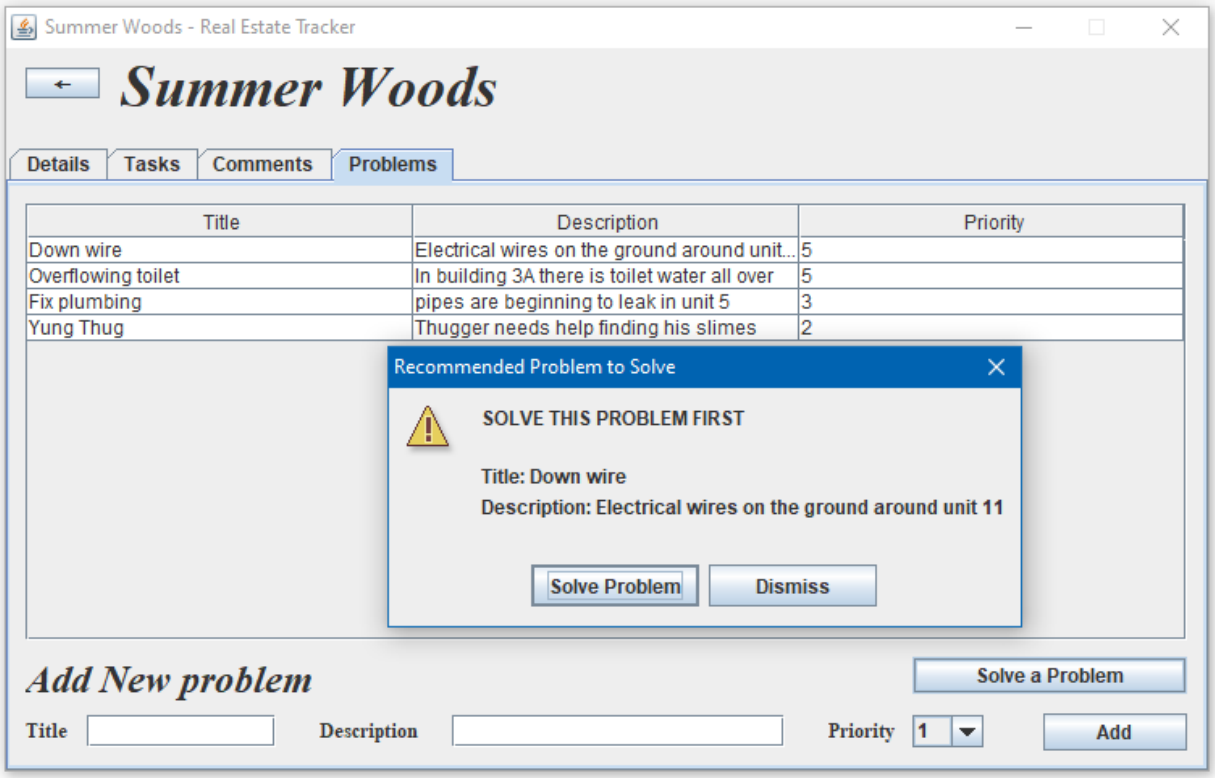


Figure 17: The JOptionPane dynamically prints the dequeued problem instance once the "Solve a Problem" button action listener is executed

Reversely, problems can be dequeued with the "Solve a Problem" button.

```
//Creates a recommended problem to solve
btnRecommendAProblem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //reads the problems csv file
        StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
        String fileName="";
        while(st.hasMoreTokens()) fileName+=st.nextToken();
        File csvFile = new File(fileName+"Problems.csv");
        Scanner reader=null;
            try {reader = new Scanner(csvFile);}
            catch (FileNotFoundException e1) {e1.printStackTrace();}

        //creates the data ArrayList
        ArrayList<String> dataList = new ArrayList<String>();

        //reads through the header values (they have already been set and only...
        //...exist in the text file for exporting reasons)
        reader.nextLine();

        //reads the actual task data
        while(reader.hasNextLine()) dataList.add(reader.nextLine());

        //converts from an ArrayList to a normal array
        String[] dataArray = new String[dataList.size()];
        for(int x=0;x<dataArray.length;x++) dataArray[x] = dataList.get(x);

        //creates an array of Problem instances
        PQueue tempProblemQueue = new PQueue(dataList.size()*3);
        //the capacity is three times larger than the list size for extra...
        //...robustness at the sake of computational speed
```

```
        //adds the data to the Problem Queue
        for(int x=0;x<dataArray.length;x++){
            String[] rowData = new String[3];
            int rowIndex = 0;
            StringTokenizer st2 = new StringTokenizer(dataArray[x], ",");
            String title = st2.nextToken();
            String description = st2.nextToken();
            int priority = Integer.parseInt(st2.nextToken());
            tempProblemQueue.enqueue(new Problem(title, description, priority));
        }
        reader.close();

        //checks for empty problem queue
        if(tempProblemQueue.isEmpty()) {
            JOptionPane.showMessageDialog(new JFrame(),
                    "There are no problems to solve",
                    "ERROR",
                    JOptionPane.WARNING_MESSAGE);
        }
        else {
            //recognizes the recommended problem
            Problem recommendation = tempProblemQueue.front();
            String output = "";
            output += "SOLVE THIS PROBLEM FIRST\n\n";
            output += "Title: "+recommendation.getTitle()+"\n";
            output += "Description: "+recommendation.getDescription()+"\n";
            output += " ";

            Object[] selectionOptions={"Solve Problem","Dismiss"};
            int resetWarning = JOptionPane.showOptionDialog //generates a JOptionPane warning the client
                     (null, output,
                            "Recommended Problem to Solve",JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,
                            null, selectionOptions, selectionOptions[0]);
            if(resetWarning==0) { //runs if "Solve Problem" was selected
                Problem tempProblem = problemTable.removeProblem();
                problemTable.generateData(); //removes the problem

                ProblemEvent temp = new ProblemEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
                        tempProblem.getTitle(), tempProblem.getDescription(), ""+tempProblem.getPriority(), 1);
                //problem event is instantiated with a "1" parameter at the end to signal this is a ~removed~ problem

                EventScripter.addEvent(temp);
            }
        }
    }
});
```

ProblemsTable's removeProble() method is evoked, which behaves by dequeuing the problem and rewriting the problem CSV

```
public Problem removeProblem(){
    //makes the altered queue
    generateProblemQueue();
    Problem removedProblem = problemQueue.dequeue();

    //makes the file
    File csvFile = new File(super.getFileName()+"Problems.csv");

    //deletes the file's contents
    PrintWriter writer = null;
        try {writer = new PrintWriter(csvFile);}
        catch (FileNotFoundException e) {e.printStackTrace();}
    writer.print("");

    //makes the file editor classes
    FileWriter fileWriter = null;
        try {fileWriter = new FileWriter(csvFile, true);}
        catch (IOException e1) {e1.printStackTrace();}
    PrintWriter printWriter = new PrintWriter(fileWriter);

    //sets up header
    printWriter.print("Title,Description,Priority");

    //prints the new queue
    while(!problemQueue.isEmpty()) {
        Problem temp = problemQueue.dequeue();
        printWriter.print("\r\n");
        printWriter.print(temp.getTitle()+","+temp.getDescription()+","+temp.getPriority());
    }
    printWriter.close();

    return removedProblem;
}
```

Event Log

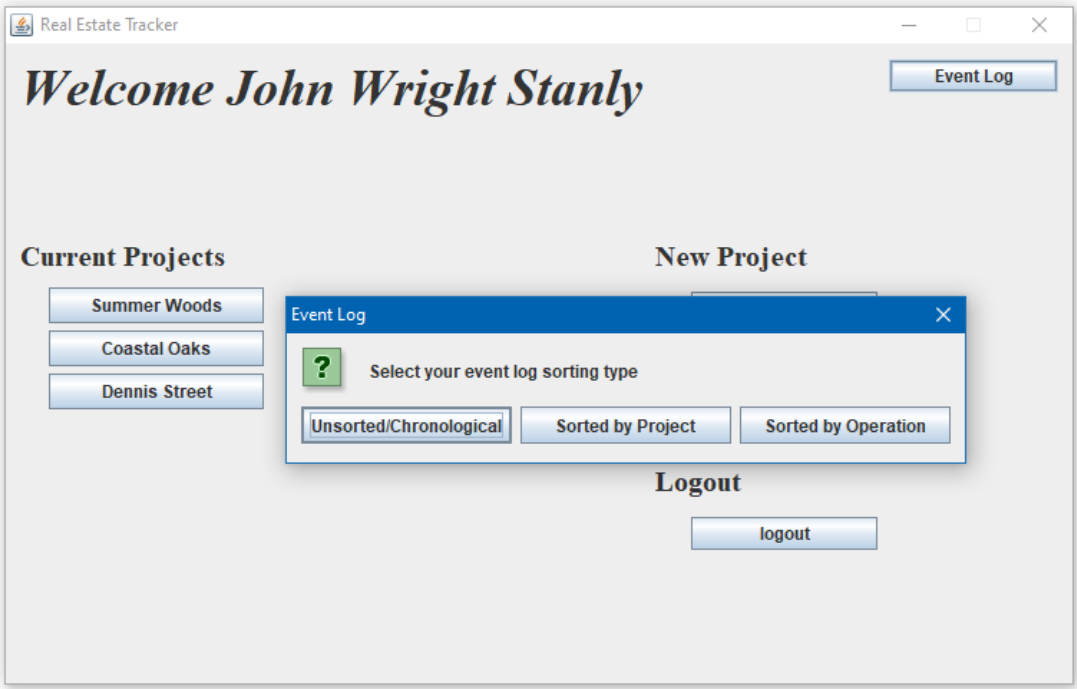


Figure 18: The JOptionPane prompting the client for their sorting choice

```java
import java.util.ArrayList;

public abstract class Event {

    protected static final String[] operationOptions = {"Undeclaired","Data","Task","Comment","Problem"};

    protected String client;
    protected String projectName;
    protected String operation;

    public Event() {
        client = "NA";
        projectName = "NA";
        operation = operationOptions[0];
    }

    public Event(String input, String input2) {
        client = input;
        projectName = input2;
        operation = operationOptions[0];
    }

    public Event(String input, String input2, int input3) {
        client = input;
        projectName = input2;
        operation = operationOptions[input3];
    }

    public static String[] getOperationOptions() {
        return operationOptions;
    }

    public String getClient() {
        return client;
    }

    public void setClient(String client) {
        this.client = client;
    }

    public String getProjectName() {
        return projectName;
    }

    public void setProjectName(String projectName) {
        this.projectName = projectName;
    }

    public String getOperation() {
        return operation;
    }


    public void setOperation(String operation) {
        this.operation = operation;
    }

    public String toString() {
        String output = client + " edited " + projectName + "'s " + operation + " page";
        return output;
    }

    public String scriptLog() {
        String output = client + "~" + projectName + "~" + "0" + "~";
        return output;
    }
}
```

The abstract Event class enables extension.

Data Event

```
public class DataEvent extends Event{

    private String dataFieldChanged;
    private String originalDataValue;
    private String newDataValue;

    public DataEvent() {
        super();
    }

    public DataEvent(String input1, String input2, String[] input4) {
        super(input1, input2);
        super.operation = super.operationOptions[1];

        assert input4.length >= 3;

        dataFieldChanged = input4[0];
        originalDataValue = input4[1];
        newDataValue = input4[2];
    }

    public DataEvent(String input1, String input2, String input3, String input4, String input5) {
        super(input1, input2);
        super.operation = super.operationOptions[1];

        dataFieldChanged = input3;
        originalDataValue = input4;
        newDataValue = input5;
    }

    public String getDataFieldChanged() {
        return dataFieldChanged;
    }

    public void setDataFieldChanged(String dataFieldChanged) {
        this.dataFieldChanged = dataFieldChanged;
    }

    public String getOriginalDataValue() {
        return originalDataValue;
    }

    public void setOriginalDataValue(String originalDataValue) {
        this.originalDataValue = originalDataValue;
    }

    public String getNewDataValue() {
        return newDataValue;
    }

    public void setNewDataValue(String newDataValue) {
        this.newDataValue = newDataValue;
    }
```

```
    public String toString() {
        String output = client + " edited " + projectName + "'s " + operation + " page: " +
                        dataFieldChanged + " was changed from " + originalDataValue + " to " + newDataValue;
        return output;
    }

    public String scriptLog() {
        String output = client + "~" + projectName + "~" + "1" + "~" + dataFieldChanged +
                        "~" + originalDataValue + "~" + newDataValue + "~";
        return output;
    }
}
```

Task Event

```
public class TaskEvent extends Event{

    private String taskCategory;
    private String taskContractor;
    private String taskProjectedCompletion;

    public TaskEvent() {
        super();
    }

    public TaskEvent(String input1, String input2, String[] input3) {
        super(input1, input2);
        super.operation = super.operationOptions[2];

        assert input3.length >= 3;

        taskCategory = input3[0];
        taskContractor = input3[1];
        taskProjectedCompletion = input3[2];
    }

    public TaskEvent(String input1, String input2, String input3, String input4, String input5) {
        super(input1, input2);
        super.operation = super.operationOptions[2];

        taskCategory = input3;
        taskContractor = input4;
        taskProjectedCompletion = input5;
    }

    public String getTaskCategory() {
        return taskCategory;
    }

    public void setTaskCategory(String taskCategory) {
        this.taskCategory = taskCategory;
    }

    public String getTaskContractor() {
        return taskContractor;
    }

    public void setTaskContractor(String taskContractor) {
        this.taskContractor = taskContractor;
    }

    public String getTaskProjectedCompletion() {
        return taskProjectedCompletion;
    }

    public void setTaskProjectedCompletion(String taskProjectedCompletion) {
        this.taskProjectedCompletion = taskProjectedCompletion;
    }
```

```
    public String toString() {
        String output = client + " edited " + projectName + "'s " + operation + " page: A " +
                        taskCategory + " task was contracted with " + taskContractor + " set for " + taskProjectedCompletion;
        return output;
    }

    public String scriptLog() {
        String output = client + "~" + projectName + "~" + "2" + "~" + taskCategory +
                        "~" + taskContractor + "~" + taskProjectedCompletion + "~";
        return output;
    }

}
```

Comment Event

```
public class CommentEvent extends Event{

    private String comment;

    public CommentEvent() {
        super();
    }

    public CommentEvent(String input1, String input2, String input3) {
        super(input1, input2);
        super.operation = super.operationOptions[3];
        comment = input3;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }

    public String toString() {
        String output = client + " edited " + projectName + "'s " + operation + " page: \"" +
                        comment + "\" was commented by " + client;
        return output;
    }

    public String scriptLog() {
        String output = client + "~" + projectName + "~" + "3" + "~" + comment + "~";
        return output;
    }
}
```

Problem Event

```java
public class ProblemEvent extends Event{

    private String problemTitle;
    private String problemDescription;
    private String problemPriority;
    private int problemAddedOrDeleted;
        //0=added (enqueued)
        //1=deleted (dequeued)

    public ProblemEvent() {
        super();
    }

    public ProblemEvent(String input1, String input2, String[] input3) {
        super(input1, input2);
        super.operation = super.operationOptions[4];

        assert input3.length >= 4;

        problemTitle = input3[0];
        problemDescription = input3[1];
        problemPriority = input3[2];
        int temp = Integer.parseInt(input3[3].trim());
        assert temp==0||temp==1;
        problemAddedOrDeleted = temp;
    }

    public ProblemEvent(String input1, String input2, String input3, String input4, String input5, int input6) {
        super(input1, input2);
        super.operation = super.operationOptions[4];

        problemTitle = input3;
        problemDescription = input4;
        problemPriority = input5;
        assert input6==0||input6==1;
        problemAddedOrDeleted = input6;
    }

    public String getProblemTitle() {
        return problemTitle;
    }

    public void setProblemTitle(String problemTitle) {
        this.problemTitle = problemTitle;
    }

    public String getProblemDescription() {
        return problemDescription;
    }

    public void setProblemDescription(String problemDescription) {
        this.problemDescription = problemDescription;
    }
```

```
    public String getProblemPriority() {
        return problemPriority;
    }

    public void setProblemPriority(String problemPriority) {
        this.problemPriority = problemPriority;
    }

    public int getProblemAddedOrDeleted() {
        return problemAddedOrDeleted;
    }

    public void setProblemAddedOrDeleted(int problemAddedOrDeleted) {
        assert problemAddedOrDeleted==0||problemAddedOrDeleted==1;
        this.problemAddedOrDeleted = problemAddedOrDeleted;
    }

    public String toString() {
        String output = client + " edited " + projectName + "'s " + operation + " page: ";
        if(problemAddedOrDeleted==0) {
            output += problemTitle + " (" + problemDescription + ") was added as a problem with priority " + problemPriority;
        }
        if(problemAddedOrDeleted==1) {
            output += problemTitle + " (" + problemDescription + ") was removed as a problem with priority " + problemPriority;
        }

        return output;
    }

    public String scriptLog() {
        String output = client + "~" + projectName + "~" + "4" + "~" + problemTitle +
                        "~" + problemDescription + "~" + problemPriority +"~" + problemAddedOrDeleted + "~";
        return output;
    }

}
```

Each concrete child class enables respective data points to be stored. Additionally, each class has a unique scriptLog() method.

Together, instances are stored in the EventStack found in the EventScript class.

```
import java.io.File;

public class EventScripter {
    private static Stack<Event> eventStack = new Stack<Event>();
    private static File eventLog = new File("eventLog.txt");
    private static FileWriter fileWriter;
    private static PrintWriter printWriter;
    private static Scanner scanner;
```

The addEvent() method (used throughout the program already) adds new events to the stack and also logs it to the eventLog.txt

```java
public static String addEvent(Event newEvent) {
    //adds to the Stack
    eventStack.push(newEvent);

    //adds to the event log file
    try {
        //creates the file writer and print writer
        fileWriter = new FileWriter(eventLog, true);
        printWriter = new PrintWriter(fileWriter);
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    //adds the file's scriptLog to the event log
    printWriter.print(newEvent.scriptLog());
    printWriter.print("\r\n");
    System.out.println(newEvent.toString());

    printWriter.close();

    return newEvent.toString();
}
```

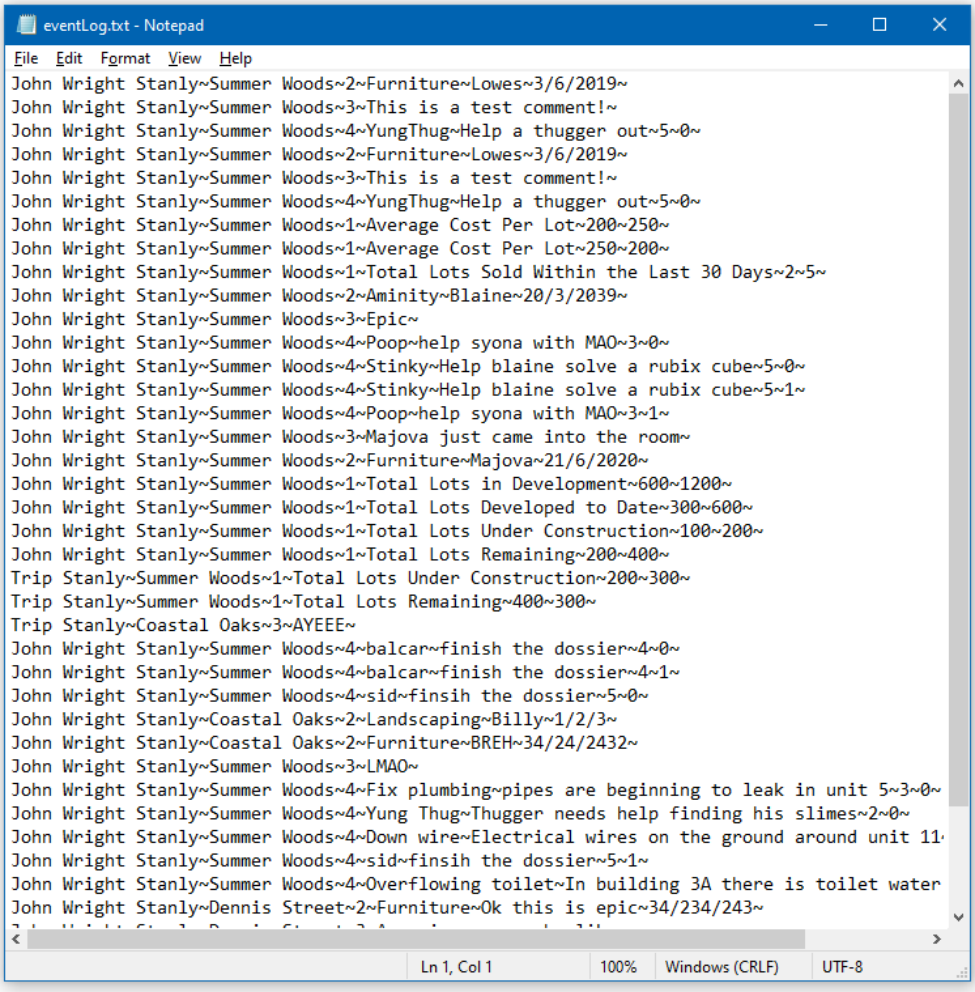


Figure 19: The eventLog.txt file. Notice " ~ " is used as a delimiter for the EventScripter class

Reversely, the buildStackFromEventLogFile() method creates the stack from the eventLog file.

```
public static void buildStackFromEventLogFile() {

    if(!eventStack.isEmpty()) eventStack.clear();

    try {
        scanner = new Scanner(eventLog);
    }
    catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    while(scanner.hasNextLine()) {
        //reads next line
        String[] tempTokens = scanner.nextLine().split("~");

        //creates an array with guaranteed size >5
        String[] tokens = new String[7];
        for(int x=0;x<tempTokens.length;x++) tokens[x] = tempTokens[x];

        //adds according to the event type
        switch(Integer.parseInt(tokens[2].trim())) {
        case 1:
            DataEvent tempData = new DataEvent(tokens[0],tokens[1],tokens[3],
                    tokens[4],tokens[5]);
            eventStack.push(tempData);
            //System.out.println("New data event added to stack --- " + tempData);
            break;
        case 2:
            TaskEvent tempTask = new TaskEvent(tokens[0],tokens[1],tokens[3],
                    tokens[4],tokens[5]);
            eventStack.push(tempTask);
            //System.out.println("New task event added to stack --- " + tempTask);
            break;
        case 3:
            CommentEvent tempComment = new CommentEvent(tokens[0],tokens[1],tokens[3]);
            eventStack.push(tempComment);
            //System.out.println("New comment event added to stack --- " + tempComment);
            break;
        case 4:
            ProblemEvent tempProblem = new ProblemEvent(tokens[0],tokens[1],tokens[3],
                    tokens[4],tokens[5],Integer.parseInt(tokens[6].trim()));
            eventStack.push(tempProblem);
            //System.out.println("New problem event added to stack --- " + tempProblem);
            break;
        }
    }
    scanner.close();
}
```

The buildLogFileFromStack() method is vice versa, constructing the file from the stack

```
public static void buildLogFileFromStack() {

    //clears the contents of the existing events log file
    FileWriter fwOb = null;
    PrintWriter pwOb = null;
    try {
        fwOb = new FileWriter(eventLog, false);
        pwOb = new PrintWriter(fwOb, false);
    }
    catch (IOException e1) {
        e1.printStackTrace();
    }
    pwOb.flush();
    pwOb.close();
    try {
        fwOb.close();
    }
    catch (IOException e1) {
        e1.printStackTrace();
    }

    //adds the events from the Stack
    try {
        //creates the file writer and print writer
        fileWriter = new FileWriter(eventLog, true);
        printWriter = new PrintWriter(fileWriter);
    }
    catch (IOException e) {
        e.printStackTrace();
    }

    //flips the ordering so the following lines of code don't write to the text field backwards
    Stack<Event> temp = new Stack<Event>();
    while(!eventStack.isEmpty()) {
        temp.push(eventStack.pop());
    }

    //adds the file's scriptLog to the event log
    System.out.println("DEBUG TEMP: "+temp);
    while(!temp.isEmpty()) {
        printWriter.print(temp.pop().scriptLog());
        printWriter.print("\r\n");
    }
    printWriter.close();

    //rebuilds the stack becuase in the process of creating the next text file, the stack was emptied
    buildStackFromEventLogFile();
}
```

The event stack can be viewed with the “Event Log” button (page 63). The JOptionPane and switch statement allows the client an ideal sorting algorithm.

```
//Allows the client to view the event script
btnEventLog.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        Object[] options = {"Unsorted/Chronological",
                            "Sorted by Project",
                            "Sorted by Operation"};
        int n = JOptionPane.showOptionDialog(new JFrame(),
            "Select your event log sorting type",
            "Event Log",
            JOptionPane.YES_NO_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,
            options,
            options[0]);

        JFrame eventFrame = new JFrame();
            //eventFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            eventFrame.setTitle("Event Log");
            eventFrame.setSize(500,500);
            eventFrame.setLocationRelativeTo(null);

        JPanel eventPanel = new JPanel();
        GridLayout layout = new GridLayout();
        eventPanel.setLayout(layout);

        String[][] data = null;
        switch(n) {
        case 0:
            data = EventScripter.getUnsortedEventTable(account[accountIndexUsed].getName());
            break;
        case 1:
            data = EventScripter.getProjectSortedEventTable(account[accountIndexUsed].getName());
            break;
        case 2:
            data = EventScripter.getOperationSortedEventTable(account[accountIndexUsed].getName());
            break;
        }
        String[] columnName = {"Project","Operation","Details"};
        JTable eventTable = new JTable(data, columnName);
        JScrollPane scrollpane = new JScrollPane(eventTable);
        eventPanel.add(scrollpane);


        eventFrame.add(eventPanel);
        eventFrame.setVisible(true);
    }
});
```

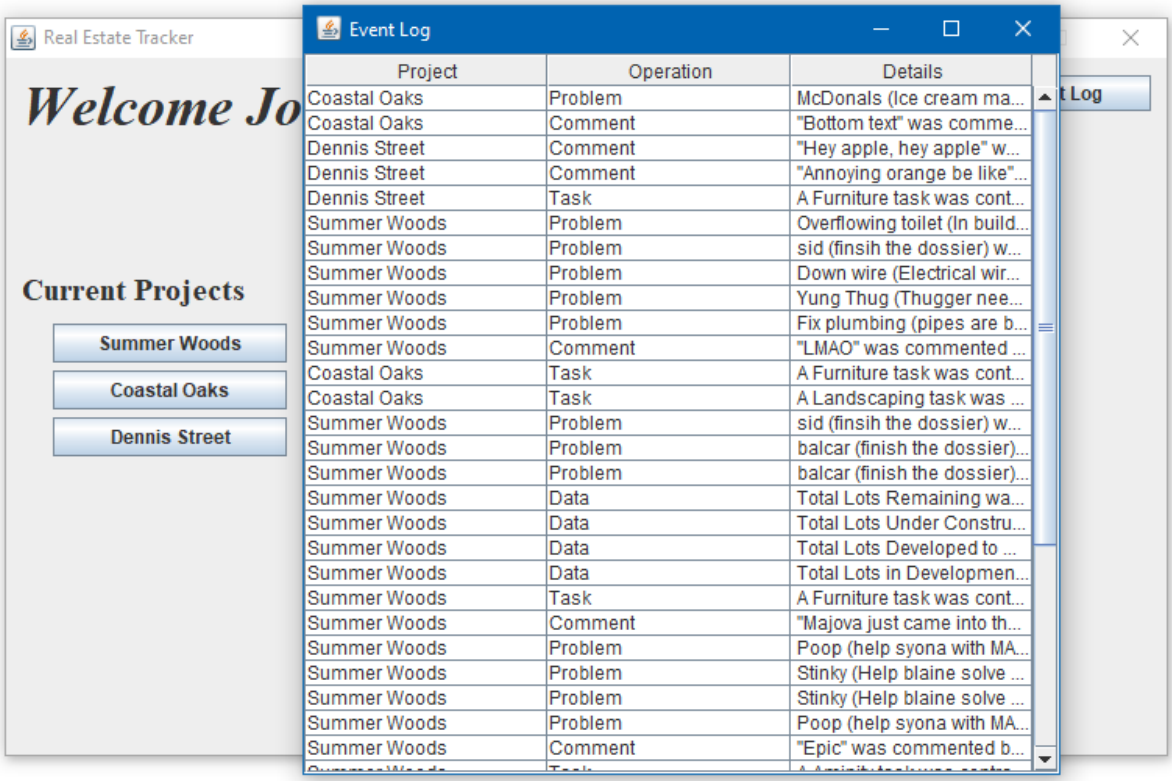


Figure 20: An "unsorted/chronological" sort displayed in the Event Log

"Unsorted/Chronological" returns a list of events in the order they were created with the following EventScripter method

```
public static String[][] getUnsortedEventTable(String name){

    buildStackFromEventLogFile();

    //constructs a properly sized 2D array
    String data[][] = new String[eventStack.size()][3];

    //copies the existing Stack to a temp variable
    Stack<Event> tempStack = (Stack<Event>) eventStack.clone();

    int c=0;
    while(!tempStack.isEmpty()) {
        Event tempEvent = tempStack.pop();
        if(tempEvent.getClient().equals(name)) {

            data[c][0] = tempEvent.getProjectName();

            //removes the unnessary content from the toString that complicates...
            //...the already mentioned details by removing everything before the colon
            String[] tempTokens = tempEvent.toString().split(":");
            String details = tempTokens[1];

            //distinguishes which operation was used and properly...
            //...writes the subsequent data
            if(tempEvent.getOperation().trim().equals("Data")) {
                data[c][1] = "Data";
                data[c][2] = details;
                c++;
            }
            if(tempEvent.getOperation().trim().equals("Task")) {
                data[c][1] = "Task";
                data[c][2] = details;
                c++;
            }
            if(tempEvent.getOperation().trim().equals("Comment")) {
                data[c][1] = "Comment";
                data[c][2] = details;
                c++;
            }
            if(tempEvent.getOperation().trim().equals("Problem")) {
                data[c][1] = "Problem";
                data[c][2] = details;
                c++;
            }
        }
    }
```

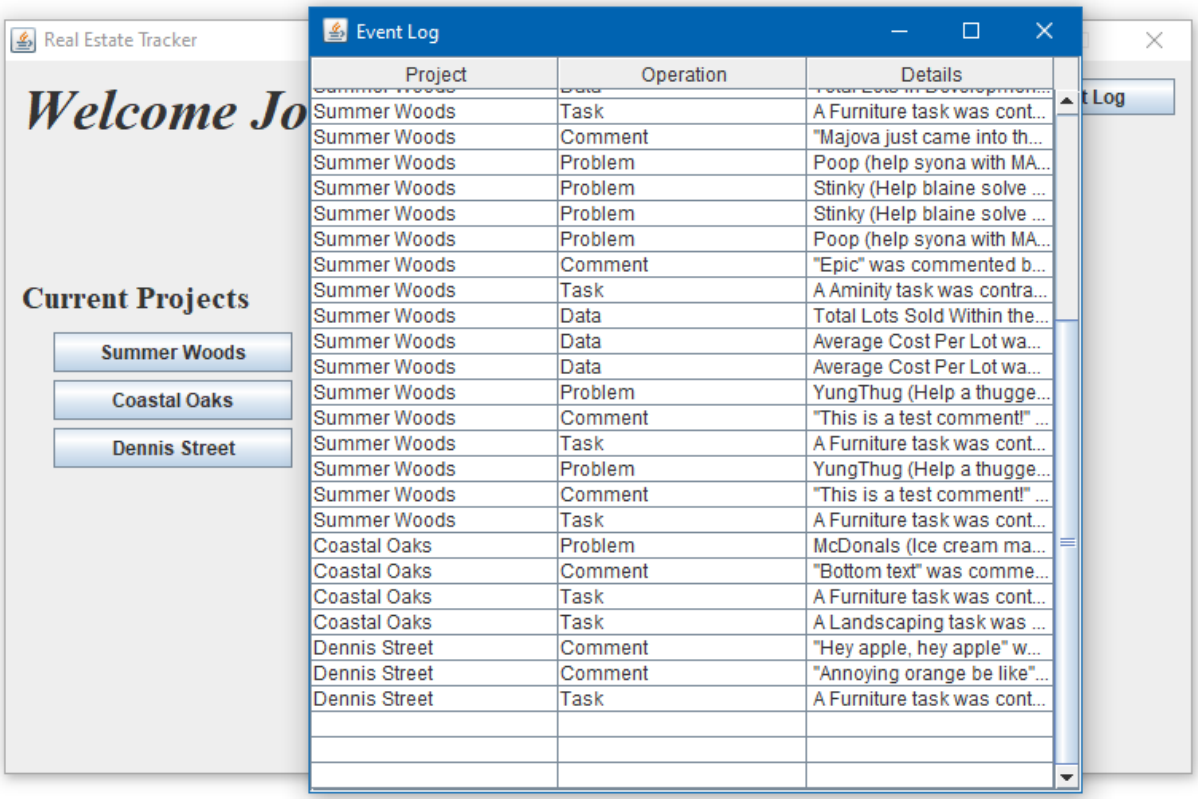


Figure 21: A "Sorted by Project" sort displayed in the Event Log. Although the vast majority of events logged are with the Summer Woods project, the Coastal Oaks and Dennis Street events are noticeably grouped together at the bottom of the table.

```
public static String[][] getProjectSortedEventTable(String name){

    buildStackFromEventLogFile();

    //creates a properly sized 2D array
    String data[][] = new String[eventStack.size()][3];

    //copies the existing stack to a temp variable
    Stack<Event> tempStack = (Stack<Event>) eventStack.clone();

    //adds every element in the stack to the new ArrayList
    ArrayList<Event> eventList = new ArrayList<Event>();
    while(!tempStack.isEmpty()) {
        Event tempEvent = tempStack.pop();
        if(tempEvent.getClient().equals(name)) {
            eventList.add(tempEvent);
        }
    }

    //Uses RandomAccessFileEditor to generate a list of projects
    String[] projectArr = null;
    try {projectArr = RandomAccessFileEditor.getProjects("listOfProjectsRAF.txt");}
    catch (Exception e) {e.printStackTrace();}

    /*Creates an array of ArrayLists to store events to. Each ArrayList contains
    all the events from one project. Thus, the array contains all the ArrayLists
    of all the different projects the client is a part of. The loop manually
    instantiates each ArrayList to avoid null pointer errors*/
    ArrayList<Event>[] sortedEventList = new ArrayList[projectArr.length];
    for(int x=0;x<sortedEventList.length;x++) {
        sortedEventList[x] = new ArrayList<Event>();
    }

    for(int x=0;x<eventList.size();x++) {
        for(int y=0;y<projectArr.length;y++) {
            StringTokenizer st = new StringTokenizer(eventList.get(x).getProjectName());
            String projectName = "";
            while(st.hasMoreTokens()) projectName += st.nextToken();
            //^modifies the Project Name's title to contain no spaces
            if(projectName.equals(projectArr[y])) {
                sortedEventList[y].add(eventList.get(x));
            }
            //^if an event matches a project, it's added to its respective ArrayList
        }
    }

     ArrayList<Event> finishedSortedEventList = new ArrayList<Event>();
     //^The final ArrayList to contain all the merged events, now in proper order
     for(int x=0;x<sortedEventList.length;x++) {
         if(!sortedEventList[x].isEmpty()) {
         //^checks to see if the project even has any events
             for(int y=0;y<sortedEventList[x].size();y++) {
                 finishedSortedEventList.add(sortedEventList[x].get(y)); //adds event in order
             }
         }
     }
```

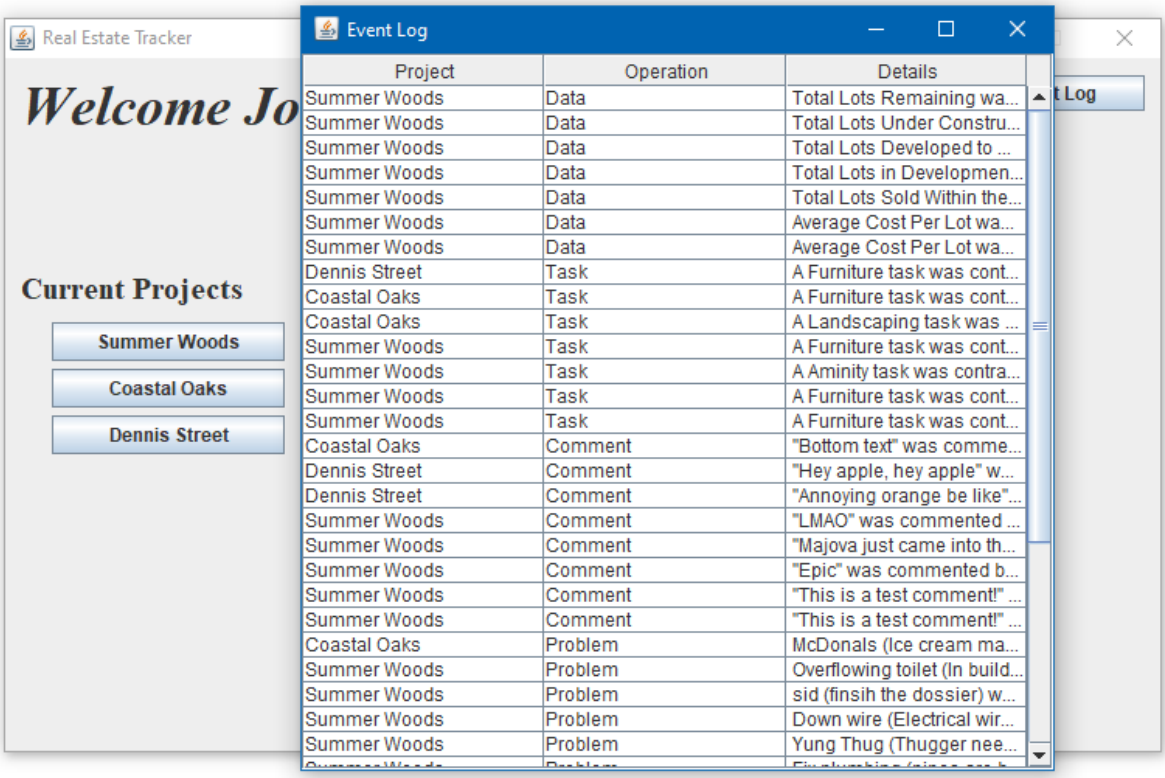
```
    for(int z=0;z<finishedSortedEventList.size();z++) {
        Event tempEvent = finishedSortedEventList.get(z);
        data[z][0] = tempEvent.getProjectName();

        String[] tempTokens = tempEvent.toString().split(":");
        String details = tempTokens[1];

        if(tempEvent.getOperation().trim().equals("Data")) {
            data[z][1] = "Data";
            data[z][2] = details;
        }
        if(tempEvent.getOperation().trim().equals("Task")) {
            data[z][1] = "Task";
            data[z][2] = details;
        }
        if(tempEvent.getOperation().trim().equals("Comment")) {
            data[z][1] = "Comment";
            data[z][2] = details;
        }
        if(tempEvent.getOperation().trim().equals("Problem")) {
            data[z][1] = "Problem";
            data[z][2] = details;
        }
    }

    //System prints for debugging
    for(int x=0;x<data.length;x++) {
        for(int y=0;y<data[x].length;y++) {
            System.out.print(data[x][y]);
            System.out.print("\t");
        }
        System.out.print("\n");
    }

    return data;
}
```



Project	Operation	Details
Summer Woods	Data	Total Lots Remaining wa...
Summer Woods	Data	Total Lots Under Constru...
Summer Woods	Data	Total Lots Developed to ...
Summer Woods	Data	Total Lots in Developmen...
Summer Woods	Data	Total Lots Sold Within the...
Summer Woods	Data	Average Cost Per Lot wa...
Summer Woods	Data	Average Cost Per Lot wa...
Dennis Street	Task	A Furniture task was cont...
Coastal Oaks	Task	A Furniture task was cont...
Coastal Oaks	Task	A Landscaping task was ...
Summer Woods	Task	A Furniture task was cont...
Summer Woods	Task	A Aminity task was contra...
Summer Woods	Task	A Furniture task was cont...
Summer Woods	Task	A Furniture task was cont...
Coastal Oaks	Comment	"Bottom text" was comme...
Dennis Street	Comment	"Hey apple, hey apple" w...
Dennis Street	Comment	"Annoying orange be like"...
Summer Woods	Comment	"LMAO" was commented ...
Summer Woods	Comment	"Majova just came into th...
Summer Woods	Comment	"Epic" was commented b...
Summer Woods	Comment	"This is a test comment!" ...
Summer Woods	Comment	"This is a test comment!" ...
Coastal Oaks	Problem	McDonals (Ice cream ma...
Summer Woods	Problem	Overflowing toilet (In build...
Summer Woods	Problem	sid (finsih the dossier) w...
Summer Woods	Problem	Down wire (Electrical wir...
Summer Woods	Problem	Yung Thug (Thugger nee...

Figure 22: A "Sorted by Operation" sort displayed in the Event Log

```
public static String[][] getOperationSortedEventTable(String name){


    buildStackFromEventLogFile();

    String data[][] = new String[eventStack.size()][3];

    Stack<Event> tempStack = (Stack<Event>) eventStack.clone();

    ArrayList<Event> eventList = new ArrayList<Event>();
    while(!tempStack.isEmpty()) {
        Event tempEvent = tempStack.pop();
        if(tempEvent.getClient().equals(name)) {
            eventList.add(tempEvent);
        }
    }

    //lists the operations in a String array
    String[] operationArr = {"Undeclaired","Data","Task","Comment","Problem"};;

    /*Creates an array of ArrayLists to store events to. Each ArrayList contains
    all the events from one type of operation. Thus, the array contains all the
    ArrayLists of all the different operations the client is a part of. The loop
    manually instantiates each ArrayList to avoid null pointer errors*/
    ArrayList<Event>[] sortedEventList = new ArrayList[operationArr.length];
    for(int x=0;x<sortedEventList.length;x++) {
        sortedEventList[x] = new ArrayList<Event>();
    }

    for(int x=0;x<eventList.size();x++) {
        for(int y=0;y<operationArr.length;y++) {
            if(eventList.get(x).getOperation().equals(operationArr[y])) {
                sortedEventList[y].add(eventList.get(x));
            }
            //^if an event matches an operation, it's added to its respective ArrayList
        }
    }

    ArrayList<Event> finishedSortedEventList = new ArrayList<Event>();
    //^The final ArrayList to contain all the merged events, now in proper order
    for(int x=0;x<sortedEventList.length;x++) {
        if(!sortedEventList[x].isEmpty()) {
            //^checks to see if the project even has any events
            for(int y=0;y<sortedEventList[x].size();y++) {
                finishedSortedEventList.add(sortedEventList[x].get(y)); //adds event in order
            }
        }
    }
```

```
    for(int z=0;z<finishedSortedEventList.size();z++) {
        Event tempEvent = finishedSortedEventList.get(z);
        data[z][0] = tempEvent.getProjectName();

        String[] tempTokens = tempEvent.toString().split(":");
        String details = tempTokens[1];

        if(tempEvent.getOperation().trim().equals("Data")) {
            data[z][1] = "Data";
            data[z][2] = details;
        }
        if(tempEvent.getOperation().trim().equals("Task")) {
            data[z][1] = "Task";
            data[z][2] = details;
        }
        if(tempEvent.getOperation().trim().equals("Comment")) {
            data[z][1] = "Comment";
            data[z][2] = details;
        }
        if(tempEvent.getOperation().trim().equals("Problem")) {
            data[z][1] = "Problem";
            data[z][2] = details;
        }
    }

    //System prints for debugging
    for(int x=0;x<data.length;x++) {
        for(int y=0;y<data[x].length;y++) {
            System.out.print(data[x][y]);
            System.out.print("\t");
        }
        System.out.print("\n");
    }

    return data;
}
```

Sources

Oracle. “Java™ Platform, Standard Edition 7 API Specification.” Oracle, docs.oracle.com/javase/7/docs/api/.

"Stack Overflow - Where Developers Learn, Share, & Build Careers". Stack Overflow, 1215, https://stackoverflow.com/. Accessed 25 Jan 2019.

Cook, Charles E. Blue Pelican Java. 3rd ed., Virtual Book Worm, 2004, Accessed 25 Jan 2019.

Words: 940