GUI Forms and Events, Part II

Problem Statement and Specifications

This is the second part of an exercise where we present a GUI interface to the user and then based upon mouse click events, we perform certain activities. There are many types of mouse and key centered events that our programs can respond to, but we will restrict ourselves to mouse clicks for the time being. In the previous exercise, we built the GUI design that is displayed in Figure 1. Our goal in this exercise is to implement appropriate algorithms that respond to mouse clicks and calculate various tax amounts based upon input values.

Assignment 5 Statement:
Write a program in a file named PropertyTax5.java that uses the form shown in Figure 1 to solicit input from a user. When the calculate JButton is clicked, use the Assessment Home Value and the two Tax Rates to calculate the School and County Taxes as well as the Total Taxes. When the exit JButton is clicked, the program should shut down. All monetary amounts should be displayed with appropriate formatting.

Assignment 5 Specifications:
- Process sample JButtons for mouse clicks
- Input if the exit JButton is clicked
- Output close the window
- Input if the calculate JButton is clicked
- Input assessment values and tax rates
- Process calculate school, county and total tax values
- Output tax values to be displayed

Object Analysis and Algorithmic Development

We approached the development of this algorithm in the last exercise and will need to develop it further. To begin with, we’ll ignore the use of the JLabel objects other than their ability to identify the JTextField objects
for us. Arbitrarily, we will use the top 3 JTextField areas of Figure 1 as input fields. Let’s refer to these as assessTF, schoolrateTF and the countyrateTF. The bottom 3 JTextField areas will be used for output and let’s refer to these as schooltaxTF, countytaxTF and totaltaxTF. The JTextField objects have two methods that we’ll be using. We’ll use getText() for input and setText() for output. Both of these methods should be thought of as manipulating String objects, as was the case with the JOptionPane methods. So we’ll need methods such as Double.parseDouble() for input and String.format() for output.

As we focus on the JButton objects, we’ll need to discuss events and their processing in Java. The clicking of a JButton with the mouse is an action event which sends a message to an object known as an event listener. This will cause some method of the object to execute with the event as a parameter. Since there are only 2 JButton objects in this exercise, we’ll create an event listener object for each one. In the class statement for these objects, we’ll state that they implement ActionListener. This is a class that is available by name, but is not yet implemented. Such classes in Java are referred to as interfaces. The realization of the class method known as actionPerformed() represents the code that will be executed when the JButton is clicked.

With each JButton relying upon an action listener class to execute different lines of code, it is necessary to identify which class corresponds to which JButton object. This is referred to as registering the listener. For the calculate JButton, we’ll arbitrarily call that class the CalculateHandler. Similarly, for the exit JButton, we’ll call that class the ExitHandler. We’ll build upon the algorithm that we had established in the last exercise and identify those objects that we need to complete the program. For step 1, we’ll need to include the instantiations of the objects of type CalculateHandler as well as the ExitHandler. We’ll also need to register the instances of these classes with the JButton objects. Steps 2. and 3. will be separate classes of type ExitHandler and CalculateHandler, respectively, that we’ll pseudo code below as well.

1. **output** the form and wait for the user to create a JButton event

2. **input** if the user clicks on the exit JButton
   
   (a) **output** to shut down the window

3. **input** if the user clicks on the calculate JButton
   
   (a) **input** the 3 String values from the JTextField objects that represent the assessed value, the regional school tax rate and the county tax rate
   
   (b) **process** to convert the 3 JTextField String objects to double
(c) process to calculate the regional school tax, county tax and total tax
(d) output those three tax values in the appropriate JTextField objects

Areas that need to be added from the previous exercise are displayed in blue. As was the case in that exercise, declarations and initializations (steps 0-3) are best declared within the PropertyTax5 class but before the PropertyTax5() constructor. The remaining code (steps 4-12) represent the body of the PropertyTax5() constructor. Lines that are displayed in blue represent areas where code needs to be written or modified to complete the assignment.

1. Pseudo code for Creating and Displaying the GUI Form

<table>
<thead>
<tr>
<th>Statement</th>
<th>Data objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>0) declare &amp; initialize an action listener object for each JButton</td>
<td>CalculateHandler &amp; ExitHandler objects</td>
</tr>
<tr>
<td>1) declare &amp; initialize 6 labels</td>
<td>JLabel, String and Swing.Constants objects</td>
</tr>
<tr>
<td>2) declare &amp; initialize 6 text I/O fields</td>
<td>JTextField &amp; int objects</td>
</tr>
<tr>
<td>3) declare &amp; initialize 2 button objects</td>
<td>JButton &amp; String objects</td>
</tr>
<tr>
<td>4) declare &amp; initialize a Container</td>
<td>Container object &amp; getContentPane() method</td>
</tr>
<tr>
<td>5) configure the Container</td>
<td>Container.setLayout() method, GridLayout &amp; int objects</td>
</tr>
<tr>
<td>6) place GUI objects in the Container</td>
<td>JLabel, JTextField, JButton objects and Container.add() method</td>
</tr>
<tr>
<td>7) set the dimensions of the window</td>
<td>setSize() method &amp; int objects</td>
</tr>
<tr>
<td>8) set the title of the window</td>
<td>setTitle() method &amp; String object</td>
</tr>
<tr>
<td>9) set the visibility of the window</td>
<td>setVisible() method &amp; boolean object</td>
</tr>
<tr>
<td>10) set the default close operation</td>
<td>setDefaultCloseOperation() method &amp; int objects</td>
</tr>
<tr>
<td>11) register the listener class for the exit JButton</td>
<td>JButton.addActionListener() method and ExitHandler object</td>
</tr>
<tr>
<td>12) register the listener class for the calculate JButton</td>
<td>JButton.addActionListener() method and CalculateHandler object</td>
</tr>
</tbody>
</table>

When the user clicks on the exit JButton, we’ll be executing the actionPerformed() method of the ExitHandler class. This will simply invoke the System.exit(0) method.

2. Pseudo code for the exit JButton Event

<table>
<thead>
<tr>
<th>Statement</th>
<th>Data objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) terminate the program and close the JFrame</td>
<td>System.exit(0) method</td>
</tr>
</tbody>
</table>

When the user clicks on the calculate JButton, we’ll be executing the actionPerformed() method of the CalculateHandler class. This will read the String values in the top 3 JTextFields, convert them to double for numeric computation and then display the results in the bottom 3 JTextFields.
3. Pseudo code for the calculate JButton Event

<table>
<thead>
<tr>
<th>Statement</th>
<th>Data objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) convert assessed value to double</td>
<td>JTextField.getText() and Double.parseDouble() methods &amp; String and double objects</td>
</tr>
<tr>
<td>2) convert school tax rate to double</td>
<td>JTextField.getText() and Double.parseDouble() methods &amp; String and double objects</td>
</tr>
<tr>
<td>3) convert county tax rate to double</td>
<td>JTextField.getText() and Double.parseDouble() methods &amp; String and double objects</td>
</tr>
<tr>
<td>4) calculate school tax</td>
<td>double objects</td>
</tr>
<tr>
<td>5) calculate county tax</td>
<td>double objects</td>
</tr>
<tr>
<td>6) calculate total tax</td>
<td>double objects</td>
</tr>
<tr>
<td>7) display school tax</td>
<td>JTextField.setText() &amp; String.format() methods &amp; String and double objects</td>
</tr>
<tr>
<td>8) display county tax</td>
<td>JTextField.setText() &amp; String.format() methods &amp; String and double objects</td>
</tr>
<tr>
<td>9) display total tax</td>
<td>JTextField.setText() &amp; String.format() methods &amp; String and double objects</td>
</tr>
</tbody>
</table>

Coding for Compilation

Logon to your csc.oakton.edu account and create a new subdirectory of the labs directory called 5 to work on your assignment.

```bash
cd labs
mkdir 5
cd 5
```

You’ll need to copy over your solution from the previous lab over to your local directory by issuing the following command.

```bash
cp ../4/PropertyTax4.java PropertyTax5.java
emacs PropertyTax5.java &
```

Given that we’ve changed the file name, it will become necessary to change the name of the class as well as the name of the constructor. So, start your editing session by changing every occurrence of PropertyTax2 to PropertyTax5. In addition, to recognize events, we’ll need to include the following import statements.

```java
import java.awt.*;
import java.awt.event.*;
```

Declaration of Variables

By declaring our symbols for objects and constants within the class but outside of any methods, we are giving them what is referred to as class scope. This means that any line of code in the class can refer to the object. A more restrictive scope that is available is when the declaration occurs within a block that either defines a method or is embedded within the method of the class. Then, the symbol has what is referred to as block scope. This means that any line of code within the block can refer to the symbol. Declaring the objects in the manner that we have, (i.e. within the class, but outside of any method) means that we are giving these objects class scope and any method within the class can refer to them.
CSC 156 - Assignment 5 PropertyTax

Follow the Pseudo Code for 1.

Step 0
// declare and initialize an action listener object for each JButton

We’re going to implement the ExitHandler class before theCheckpoint and the CalculateHandler class after theCheckpoint. So, for now, we’ll code the CalculateHandler reference as a comment.

ExitHandler ebHandler = new ExitHandler();
// CalculateHandler cbHandler = new CalculateHandler();

Steps 1-10
// declare & initialize 6 labels
// declare & initialize 6 text I/O fields
// declare & initialize 2 button objects
// declare & initialize a Container
// configure the Container
// place GUI objects in the Container
// set the dimensions of the window
// set the title of the window
// set the visibility of the window
// set the default close operation

These steps were coded in the previous program and should simply be repeated. As was the case in the last exercise, steps 1 through 3 are best placed within the class but before the constructor. Steps 4 through 10 are best placed within the constructor.

Steps 11-12
// register the listener class for the exit JButton
// register the listener class for the calculate JButton

Once again, we’ll code the reference to the CalculateHandler class as a comment until we pass theCheckpoint. These would be the last lines coded within the constructor PropertyTax5().

exit.addActionListener(ebHandler);
// calculate.addActionListener(cbHandler);

Methods

As we develop the remainder of this solution, we’ll elaborate on methods and their features that we need to create comprehensive maintainable programs. Whenever we discuss a method in Java, there will always be some discussion on how it is invoked (or executed) and how it is implemented (or defined). The first method that we’ll discuss is the newest one that we’ve encountered.

Constructor Methods

Each object created with a class must have a means of instantiating or declaring it. Sometimes, there are options for initialization and other times there are default initialization that are available. The simplest type of invocation takes advantage of default initializations and conforms to the following format.

class_name symbolic_name = new class_name();
The invocation of the constructor appears after the `new` operator. Essentially, a new instance of the class is created (or instantiated) and is assigned to the `symbolic_name` for use within the program. The implementation of the constructor, like other methods, is defined within the class and follows the following format. Some classes provide multiple overloaded constructors that provide options for initialization that are distinguishable by the uniqueness of their parameter lists. In general, constructors do not have any return type, must have the same name as the class and should probably be identified with the keyword `public` so as to be invoked whenever necessary.

```java
// default constructor
class class_name {
    declarations;
    statement1;
    statement2;
    : 
    statement_n;
}

// constructor with initialization parameters
class class_name( data_type1 param1,..., data_typek paramk )
{
    declarations;
    statement1;
    statement2;
    : 
    statement_n;
}
```

**void methods**

These and other methods do not rely upon the `new` operator. In most cases, they will be invoked by placing the method name on a statement by itself with however many arguments are necessary for the method to function correctly. The general format is as follows.

```java
method_name(arg1, ..., argk);
```

All arguments must be constants, variables or expressions. The definition or implementation of these types of methods have a general format that adheres to the following example.

```java
public static void method_name( data_type1 param1,..., data_typek paramk )
{
    declarations;
    statement1;
    statement2;
    : 
    statement_n;
}
```

The primary example of such a method in our programs has been the `main()` method that each of our classes has used all along. In this particular program, the method looks as follows.
public static void main(String[] args) {
    PropertyTax5 proptax = new PropertyTax5();
}

The main() method is the default method invoked in a Java application program that is executed on the command line by the java command. The String arguments can be input from the command line along with the name of the class file that contains the bytecode of the compiled program. Our main() method simply executes a constructor to generate an instance of class PropertyTax5 that is stored in proptax.

Follow the Pseudo Code for 2.

Step 1
// terminate the program and close the JFrame

In order to implement the algorithm that we need for the exit JButton, we’ll need to create the ExitHandler class and then implement the actionPerformed() method, which is another void method, within that class. The ExitHandler class should be coded within the PropertyTax5 class, but outside of all other methods.

    public class ExitHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
    }

Checkpoint

Save your program to disk by choosing the Save command from the Files menu of your emacs session. Then, compile your program by choosing the Compile... command from the Tools menu and change the make -k that is displayed to javac PropertyTax5.java. Compiler errors can be parsed with the keystroke C-x ' and need to be repaired before your program can execute. When your program has compiled, click on your xterm window to access your command line prompt, and issue the command java PropertyTax5.

The output displayed should look in the same manner as Figure 1 however clicking on the exit JButton should close the JFrame. Once you are sure that the you’re on the right track, you should move on to implement the algorithm for the calculate JButton.

Follow the Pseudo Code for 3.

The class CalculateHandler should be coded identical to the class ExitHandler except for two instances. First of all, the lines within the actionPerformed() method will be different and, of course, the name of the class should be different.

Steps 1-3
// convert the assessed value to double
// convert the school tax rate to double
// convert the county tax rate to double

You will need 3 variables that are declared within the actionPerformed() method of type double for these assignments. Assuming that the JTextField containing the assessment information is named assessTF and that the double to contain this data is named assessD, the following declaration and initialization should be appropriate.
double assessD = Double.parseDouble(assessTF.getText());

You will need two similar assignments to convert the tax rates.

**Steps 4-6**

// calculate school tax
// calculate county tax
// calculate total taxes

Three additional double variables would be useful to store these results. The tax rates should be divided by 100 before being multiplied by the assessed value. Code these 3 assignments before continuing.

**Step 7-9**

// display the school tax
// display the county tax
// display the total taxes

These are three more statements that simply take the double results of the previous 3 computations and display the values in the remaining 3 JTextField areas of the form. For example, if totaltaxD is the double field that contains the sum of the school and county taxes and if totaltaxTF is the JTextField of the form that is designed to display that result, then the following line would be appropriate for our interests.

    totaltaxTF.setText("" + String.format("%.2f", totaltaxD));

You’ll need to code two similar statements for the display of the calculated school and county taxes.

**Testing for errors**

At this point, your form should be fully functioning. Uncomment the lines that refer to the CalculateHandler class at Step 0 and Step 12. Recompile your program and execute it to display the image shown in Figure 1. Enter the data shown in Figure 2 and click on the calculate JButton. The display should now look like Figure 3. Moreover, each time that you change any of the top 3 JTextField areas and click on the calculate
JButton, the results should be updated. Finally, the exit JButton should still be closing down your JFrame. There is a correctly function version of the program for PropertyTax5.java at this link.

**Printing and submitting**

Once you are satisfied with the correctness of your program, print it as you did with previous assignments by using the following command that assumes that you are working in the room 1234 at Oakton. Retrieve your copy from the printer.

    printer 1234 PropertyTax5.java

Finally, submit your program with the following command that assumes that you are registered in section abc of CSC 156.

    submit csc156abc 5