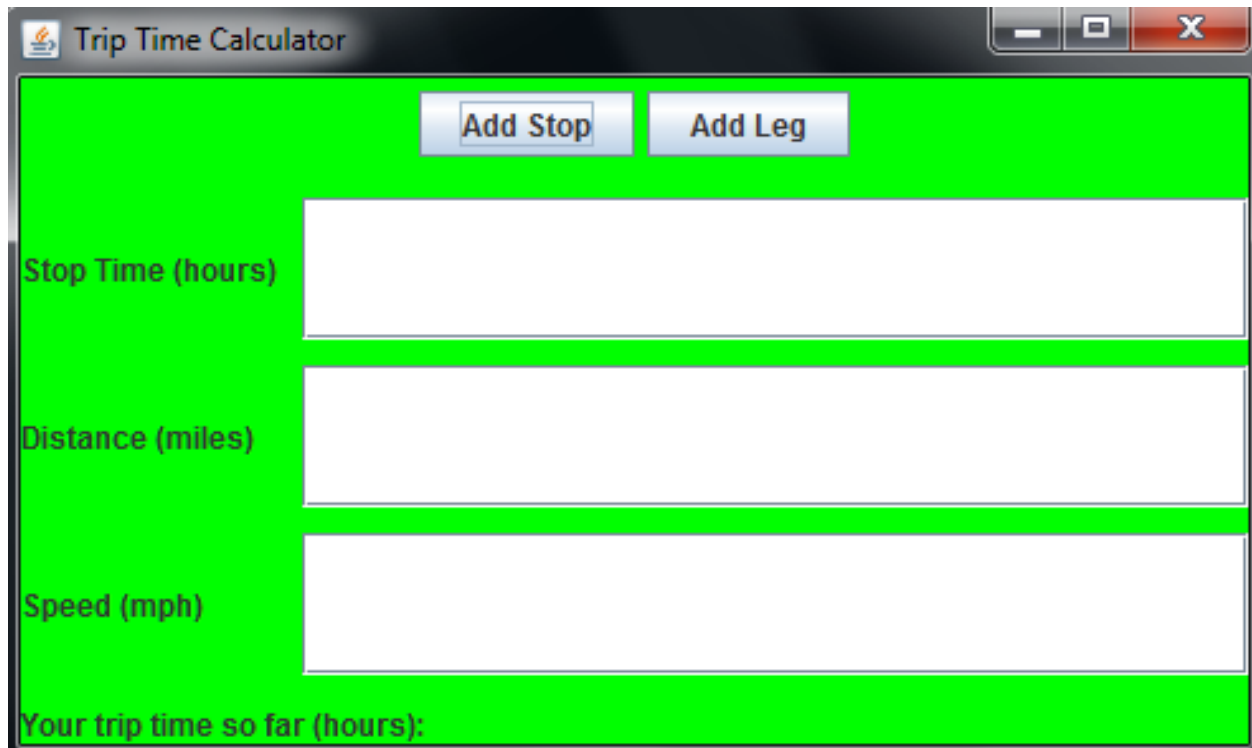


CIT 149: Java I

Chapter 9 Lab 2

This application will determine the amount of time it will take you to a destination and whether you have time to take stops along the way. This is based on stops in hours, distance and speed.

This is a GUI application and when run it will look like:



Let's get started. There will be a total of three classes for this assignment.

TripComputerException class

This class will be responsible for creating a message that users will understand when they enter incorrect data.

1. Open a new document window in TextPad and save the file as TripComputerException.java.
2. This class is very short so I will give the entire code, with explanations afterwards. Type:

```
public class TripComputerException extends Exception
{
```

```

/**
 * Creates a new instance of TripComputerException
 */

public TripComputerException(String reason)
{
    super(reason);
}
}

```

- This class has a single constructor method with the parameters of a String variable. When an instance of this class is created it will pass the text to be displayed to the user.

3. Compile the program.

TripComputer class

This class will be responsible the times and number of stops as well as the legs of the journey.

1. Open a new document window in TextPad and save the file as TripComputer.java.
2. Type the class header and opening brace.
3. We will create two private variables for the total time and the number of rest stops taken. Type:

```

private double totalTime;
private boolean restStopTaken;

```

4. The default constructor method will set initial values of the declared variables. Type:

```

public TripComputer()
{
    totalTime = 0.0;
    restStopTaken = true;
}

```

5. Our next method will compute the legs of our journey. Since an error might occur it throws a TripComputerException. Type:

```

public void computeLegTime(double distance, double speed) throws TripComputerException
{
    if(distance <=0)
        throw new TripComputerException("Negative distance not allowed");
    if(speed <=0)
        throw new TripComputerException("Negative speed not allowed");
}

```

```

double legTime = distance / speed;

totalTime += legTime;
restStopTaken = false;

}

```

- The if statements give a specific message to display if the distance or speed are negative numbers.

6. The next method will keep track of the number of rest stops. This is similar to the previous method. Type:

```

public void takeRestStop(double time) throws TripComputerException
{
    if(time <=0)
        throw new TripComputerException("Negative time not allowed");
    if(restStopTaken)
        throw new TripComputerException("Can not take another rest stop");

    totalTime += time;
    restStopTaken = true;

}

```

7. The final method simply returns the total time taken. Type:

```

public double getTripTime()
{
    return totalTime;
}

```

- This is considered to be an accessor method in that it accesses the current value of the totalTime variable.

8. Close the class and compile the program.

TripComputerApplication class

This class will set up the frame and perform the necessary functions by constructing a TripComputer object so that the methods from the class can process the necessary computations.

1. Open a new document window in TextPad and save the file as TripComputerApplication.java.
2. For first step is to import the necessary, predefined classes, by typing:

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

3. This class will need to extend the JFrame class and implement the ActionListener interface. Type:

```
public class TripComputerApplication extends JFrame implements ActionListener
{
```

4. We will create two constants that set the width and height of our frame. Since these are constant they values cannot be changed as the program runs. Type:

```
public static final int WIDTH = 500;
public static final int HEIGHT = 300;
```

5. We declare our new TripComputer object by typing:

```
private TripComputer theComputer;
```

6. We also must declared the different components for our frame. Type:

```
public JLabel totalTimeLabel, timeLabel, distanceLabel, speedLabel ;
public JTextField distanceTextField, speedTextField,timeTextField;
public JPanel buttonPanel,labelPanel,fieldPanel;
public JButton stopButton,legButton ;
```

- Notice that since I am declaring several objects of the same type I declared them on the same line, separating each by commas?

7. Our default constructor method will set up the frame. Type:

```
public TripComputerApplication()
{
    setTitle("Trip Time Calculator");
    setSize(WIDTH, HEIGHT);
    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    Container contentPane = getContentPane();
    contentPane.setLayout(new BorderLayout(10,10));
```

```
contentPane.setBackground(Color.GREEN);
```

```
theComputer = new TripComputer();
```

```
createLabels();
```

```
createFields();
```

```
createButtons();
```

```
createPanels();
```

- Instead of setting the size, and title in the main method I set them here. Either place is fine.
- The contentPane has been set with a layout manager of BorderLayout. This is the default layout manager for frames. However I wanted to add a little horizontal and vertical spacing between components in the frame so I needed to set the layout manager so that I could add these settings which are 10 horizontal and vertical pixels.
- I construct my TripComputer object as a new TripComputer object.
- Instead of constructing my labels, fields, buttons and panels within the constructor method I chose to do so in separate methods and invoke the methods. The order in which I invoke the methods is important. Since the panels will hold the other components I must invoke the other methods before invoking the createPanels() method.

8. Let's continue with the constructor method by typing:

```
contentPane.add(buttonPanel,BorderLayout.NORTH);
contentPane.add(labelPanel,BorderLayout.WEST);
contentPane.add(fieldPanel, BorderLayout.CENTER);
contentPane.add(totalTimeLabel,BorderLayout.SOUTH);
}
```

- The different panels I create will be added to the different areas of the BorderLayout.

9. The createLabels() method will simply construct the declared labels and set the text for each. Type:

```
public void createLabels()
{
    timeLabel = new JLabel("Stop Time (hours)");
    distanceLabel = new JLabel("Distance (miles)");
    speedLabel = new JLabel("Speed (mph)");
    totalTimeLabel = new JLabel("Your trip time so far (hours): ");
}
```

10. The createFields() method constructs each field. Type:

```

public void createFields()
{
    timeTextField = new JTextField();
    distanceTextField = new JTextField();
    speedTextField = new JTextField();
}

```

11. The createButtons() method constructs each button and adds a listener to each. Without the listener the program will not know how to handle a situation where the button is pressed. Type:

```

public void createButtons()
{
    stopButton = new JButton("Add Stop");
    stopButton.addActionListener(this);

    legButton = new JButton("Add Leg");
    legButton.addActionListener(this);
}

```

12. The createPanels() method constructs each panel, sets the background color, the layout manager and adds each of the labels, fields and buttons to the appropriate panel. Type:

```

public void createPanels()
{
    buttonPanel = new JPanel();
    buttonPanel.setBackground(Color.GREEN);

    labelPanel = new JPanel();
    labelPanel.setLayout(new GridLayout(3,1,0,10));
    labelPanel.setBackground(Color.GREEN);

    fieldPanel = new JPanel();
    fieldPanel.setLayout(new GridLayout(3, 1, 0, 10));
    fieldPanel.setBackground(Color.GREEN);

    buttonPanel.add(stopButton);
    buttonPanel.add(legButton);

    labelPanel.add(timeLabel);
    labelPanel.add(distanceLabel);
    labelPanel.add(speedLabel);

    fieldPanel.add(timeTextField);
}

```

```

fieldPanel.add(distanceTextField);
fieldPanel.add(speedTextField);

}

```

- The buttonPanel has a layout manager of FlowLayout. Since this is the default for panels I did not set it layout.
- The labelPanel has its layout manager changed to a GridLayout of 3 rows and 1 column and with 10 vertical pixels between each label.
- The fieldPanel has been set the same as the labelPanel.
- The buttonPanel has the buttons added to it.
- We add the labels to the labelPanel and the fields to the fieldPanel.

13. If we implement the ActionListener interface and because of this we must have the actionPerformed method. Type:

```

public void actionPerformed(ActionEvent e)
{

```

- The method header is written the same, although you can use a different identifier than *e*.

14. Since our actions are going to be based on the buttons' label we need to create a new String that receives its value from the ActionEvent's getActionCommand() method. Type:

```

String actionCommand = e.getActionCommand();

```

15. Our first action is for the button with the label of "Add Leg". Be very careful here. Java is case sensitive even for the label on a button. Spaces are considered Strings as well so do not include more or less spaces than your label. Type:

```

if (actionCommand.equals("Add Leg"))
{
    try
    {
        double speed = Double.parseDouble(speedTextField.getText().trim());
        double distance = Double.parseDouble(distanceTextField.getText().trim());
        theComputer.computeLegTime(distance, speed);
        totalTimeLabel.setText("Your trip time so far (hours): " + theComputer.getTripTime());
    }
    catch(TripComputerException ex)
    {
        totalTimeLabel.setText("Error: " + ex.getMessage());
    }
}

```

```

catch(Exception ex)
{
    totalTimeLabel.setText("Error in speed or distance: " + ex.getMessage());
}
}

```

- Our code goes within a try/catch statement since errors may occur.
- We assign the speed and distance variables values taken from two different text fields. We trim() away any excess using the trim() method. We probably won't have any excess but some users might hit the space bar after typing in a value.
- We invoke the computerLegTime() method for the TripComputer object, passing to it the values of the variables.
- We set the text on a label to the time of the trip in hours.
- Two different catch statements are used. One is for a TripComputerException and the other for any other exception.

16. The code for when the button with "Add Stop" as its label is similar. Type:

```

else if (actionCommand.equals("Add Stop"))
{
    try
    {
        double time = Double.parseDouble(timeTextField.getText().trim());
        theComputer.takeRestStop(time);
        totalTimeLabel.setText("Your trip time so far (hours): " + theComputer.getTripTime());
    }
    catch(TripComputerException ex)
    {
        totalTimeLabel.setText("Error: " + ex.getMessage());
    }
    catch(Exception ex)
    {
        totalTimeLabel.setText("Error in time: " + ex.getMessage());
    }
}

```

17. We conclude with an else in case it did not find a button with either label. Type:

```

else
    System.out.println("Error in button interface.");

```

18. Close the method.

19. Our final method is the main method. This method will simply create a `ComputerTripApplication` object and set the visibility of the frame. Type:

```
public static void main(String[] args)
{
    TripComputerApplication gui = new TripComputerApplication();
    gui.setVisible(true);
}
```

20. Close the class.
21. Compile the program and fix any errors if necessary.
22. Compress the following files into a single zip file:

`TripComputer.class`
`TripComputer.java`
`TripComputerApplication.class`
`TripComputerApplication.java`
`TripComputerException.class`
`TripComputerException.java`