

CS 602 Java, HW#2 – Group H(Individual)  
Charitha Kammari(ck369)

Use Eclipse, create, compile, run, print (program and results) and explain:

b. How Fig. 10.14 works, pages 393-394 (1 point)

Answer:

Description:

Interface:

Although this interface has a declared function called `getPaymentAmount()`, it is not used. For classes that implement it, it serves as a contract to offer functionality for computing payment amounts.

Classes:

shows a bill that has the part number, quantity, price per item, and part information.  
executes the `getPaymentAmount()` function as part of the `Payable` interface, which calculates the invoice's total payment based on the quantity and price per item.

Among the attributes of an abstract class that represents an employee are first, last, and social security number.

executes the interface for `Payable`.

provides an abstract method named `earnings()` that subclasses must implement in order to get the salary of their employees.

provides a default `getPaymentAmount()` function implementation that is based on the `profits()` method.

A certain subset of workers who receive a weekly salary.

utilizes the `earnings()` function to return the salary for that particular week and has a property for the weekly wage.

By using a `toString()` method override, a string representation of the salaried employee is supplied.

The `PayableInterfaceTest` main class's `main()` method is used to test the operation of the classes. creates instances of `SalariedEmployee` and `Invoice` among other `Payable` objects.

The code loops through the array, invoking the `toString()` and `getPaymentAmount()` methods polymorphically for each item in order to demonstrate polymorphism and dynamic method binding.

## Functionality:

Using the Payable interface, the program demonstrates polymorphic behavior, treating different object kinds (salaried staff and bills) identically.

The getPaymentAmount() function calculates the payment amount for each object using its own logic, which is quantity \* price per item for invoices and weekly compensation for salaried staff. The toString() function can be used to represent each object as a string. It also displays relevant information for employees on a salaried basis, such as personnel details and invoice data, such as component number and description.

Code:

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Eclipse, File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Eclipse toolbar icons.
- Package Explorer:** Shows packages CS602 and CS602HWone, and source files Analysis.java, Validate.java, ValidateInput.java, and ValidateInterfaceTest.java.
- Editor:** Displays the Java code for the PayableInterfaceTest.java file. The code defines an interface Payable with a method getPaymentAmount(), and a class Invoice that implements it. The Invoice class has fields partNumber, partDescription, quantity, and pricePerItem, with validation logic for non-negative values.
- Console:** Shows the output of the application execution:

```
Invoice and Employees processed polymorphically:
invoice:
part number: 84385 (house)
quantity: 1
price per item: $735.00
payment due: $735.00

invoice:
part number: 98465 (rent)
quantity: 4
price per item: $876.00
payment due: $3,504.00
This is from Employee

salaried employee: Charitha Kammani
social security number: 518-299-2453
weekly salary: $1,900.00
payment due: $1,900.00
This is from Employee

salaried employee: Bhaa Bhau
social security number: 983-293-8677
weekly salary: $900.00
payment due: $900.00
```
- Bottom:** Mac OS X dock with various application icons.

The screenshot shows the Eclipse IDE interface with the following details:

- Toolbar:** Eclipse, File, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- ActionBar:** eclipse-workspace - CS602/PayableInterfaceTest.java - Eclipse IDE
- Package Explorer:** Shows projects CS602, JRE System Library [JavaSE-17], and CS602HWs. The CS602 project contains Analysis.java, BMIcalculator.java, Craps.java, Fairfax.java, GradeBookTest.java, HelloWorld.java, JavaApplet.java, JavaProgram.java, JavaScript.java, PayableInterfaceTest.java, TicTactoe.java, Validate.java, and ValidateInput.java.
- Editor:** The editor displays the content of `PayableInterfaceTest.java`. The code defines a `Payable` interface with methods `getQuantity()`, `getPricePerItem()`, and `toString()`. It also defines a `ValidateInput` class that implements `Payable`. The `toString()` method in `ValidateInput` uses string format to print invoice details like part number, quantity, price per item, and payment due.
- Terminal:** A terminal window at the bottom shows the output of running the application, demonstrating how invoices and employees are processed polymorphically. It prints two sets of details: one for an invoice (part number 84385, quantity 1, price per item \$735.00, payment due \$735.00) and one for an employee (part number 98465, quantity 4, price per item \$876.00, payment due \$3,504.00).

```

eclipse-workspace - CS602/PayableInterfaceTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X *PayableInterfaceTest.java X Validate.java X ValidateInput.java
CS602
  JRE System Library [JavaSE-17]
  (default package)
    Analysis.java
    BMIcalculator.java
    Craps.java
    FairTax.java
    GradeBookTest.java
    HelloWorld.java
    JavaApplet.java
    JavaProgram.java
    JavaScript.java
    PayableInterfaceTest.java
    TicTactoe.java
    Validate.java
    ValidateInput.java
  CS602HWork
  JRE System Library [JavaSE-17]
  src
    (default package)
      Analysis.java
      module-info.java
  Problems @ Javadoc Declaration Console X Coverage
<terminated> PayableInterfaceTest [Java Application] /Users/ck/p2/pool/plugins/org.eclipse.jdt.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (Mar 23, 2024, 12:22:29 AM - 12:22:29 AM) [pid: 26160]
Invoices and Employees processed polymorphically:
invoice:
part number: 84385 (house)
quantity: 1
price per item: $735.00
payment due: $735.00

invoice:
part number: 98465 (rent)
quantity: 4
price per item: $876.00
payment due: $3,504.00
This is from Employee

salaried employee: Charitha Kammari
social security number: 518-299-2453
weekly salary: $1,900.00
payment due: $1,900.00
This is from Employee

salaried employee: Bhaa Bhuu
social security number: 983-293-8677
weekly salary: $900.00
payment due: $900.00

```

Output:

```

Problems @ Javadoc Declaration Console X Coverage
<terminated> PayableInterfaceTest [Java Application] /Users/ck/p2/pool/plugins/org.eclipse.jdt.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (Mar 23, 2024, 9:23:48 AM - 9:23:48 AM) [pid: 28627]
Invoices and Employees processed polymorphically:
invoice:
part number: 84385 (house)
quantity: 1
price per item: $735.00
payment due: $735.00

invoice:
part number: 98465 (rent)
quantity: 4
price per item: $876.00
payment due: $3,504.00
This is from Employee

salaried employee: Charitha Kammari
social security number: 518-299-2453
weekly salary: $1,900.00
payment due: $1,900.00
This is from Employee

salaried employee: Bhaa Bhuu
social security number: 983-293-8677
weekly salary: $900.00
payment due: $900.00

```

d. How Fig. 12.19 works on pages 460-465 (2 points)

Answer:

Description:

It imports the necessary classes from the packages javafx.application, javafx.fxml, javafx.scene, and javafx.stage.

Based on its extension of Application, TipCalculator can be identified as an application created in JavaFX.

It takes precedence over the start method, which is called upon program launch. By using FXMLLoader, this method loads the GUI (graphical user interface) from an FXML file.

An entity called Parent is established by loading the root element from the FXML file. Then, using the root node to build it, it associates the recently formed Scene object with the designated Stage.

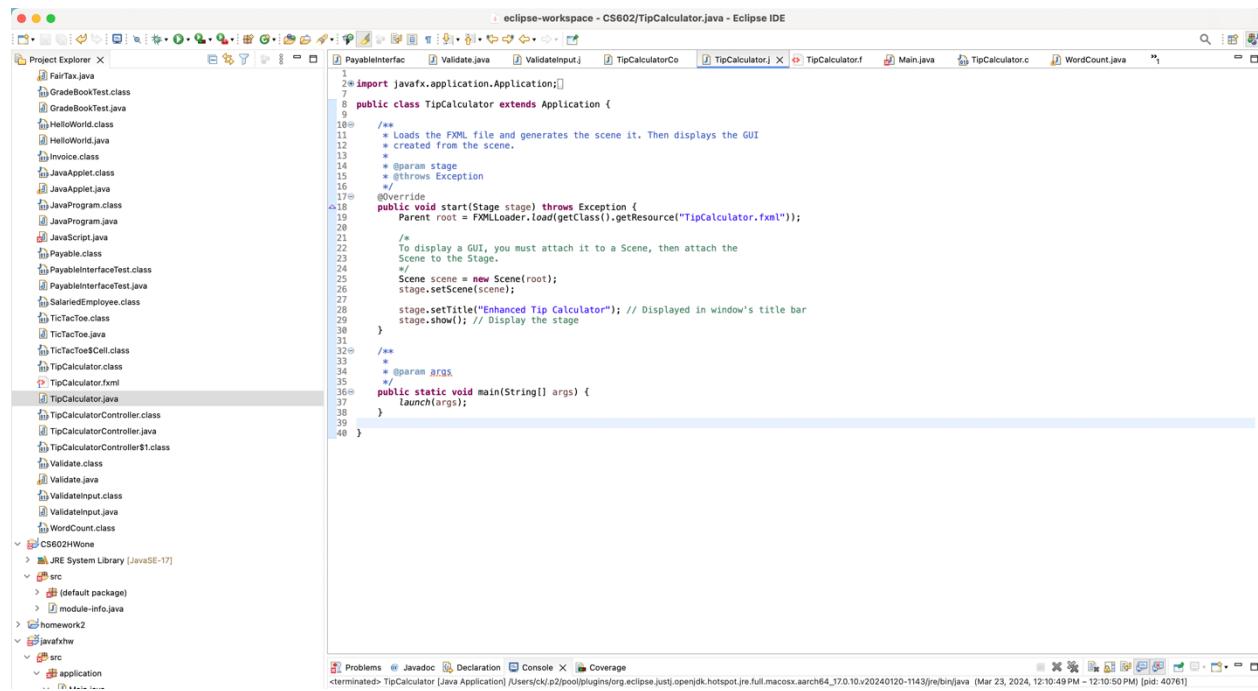
The "Enhanced Tip Calculator" is the new title for the stage.

It concludes by calling stage.show().

The application enters through this technique.

With the arguments provided, it invokes the launch function to start the JavaFX program.

Code:



The screenshot shows the Eclipse IDE interface with the project 'eclipse-workspace - CS602/TipCalculator.java' open. The Project Explorer view on the left lists various Java files and resources. The main editor window displays the TipCalculator.java code. The code imports javafx.application.Application and javafx.fxml.FXMLLoader. It defines a TipCalculator class that extends Application. The start method loads the FXML file 'TipCalculator.fxml' using FXMLLoader.load, creates a Stage, and sets its title to 'Enhanced Tip Calculator'. The main method calls launch(args). The code is annotated with Javadoc-style comments explaining the purpose of each step.

```
2# import javafx.application.Application;
3
4 public class TipCalculator extends Application {
5
6     /**
7      * Loads the FXML file and generates the scene it. Then displays the GUI
8      * created from the scene.
9      */
10    @Override
11    public void start(Stage stage) throws Exception {
12        Parent root = FXMLLoader.load(getClass().getResource("TipCalculator.fxml"));
13
14        /* param stage
15         * @throws Exception
16         */
17        Scene scene = new Scene(root);
18        stage.setScene(scene);
19        stage.setTitle("Enhanced Tip Calculator"); // Displayed in window's title bar
20        stage.show(); // Display the stage
21
22        /**
23         * To display a GUI, you must attach it to a Scene, then attach the
24         * Scene to the Stage.
25        */
26    }
27
28    /**
29     * @param args
30     */
31    public static void main(String[] args) {
32        launch(args);
33    }
34
35    /**
36     * @param args
37     */
38}
39
40}
```

Description:

The code imports the necessary classes from the java.math and java.text libraries, respectively, to handle mathematical operations and format numbers.

It also imports classes from the JavaFX library to construct the GUI application.

It is likely that this class will function as TipCalculatorController, the controller for the JavaFX application.

The class contains @FXML annotations on its instance variables. For user interface components, these variables correspond to those specified in a linked FXML file. They are composed of:

collectingTextField: TextField used to indicate the numerical size of the party.

overallIn this TextField, enter the check amount.

proportional gratuityLabel: The label displays the percentage of the tip.

proportional gratuityWith this slider, you may adjust the tip percentage.

hintTextField: TextField showing the anticipated gratuity.

an individualTextField: In TextField, the total for each individual is displayed.

completeTextField: TextField with the total amount, including the tip, displayed.

The method calculateButtonPressed(ActionEvent event) is triggered upon the pressing of the calculate button. It computes the tip, total, and amount per person after parsing the text field input values and displaying the results in the corresponding text fields. Invalid inputs are handled by handling NumberFormatExceptions.

initialize(): To initialize the controller, FXMLLoader calls this method. In addition to adding a listener to the tipPercentageSlider to update the tip percentage label and calculation when the slider value changes, it also sets up the rounding option for currency formatting.

currency and percentage: A numericalFormat objects for formatting based on percentage and currency, respectively.

tipPercentage: A BigDecimal that symbolizes the standard tip percentage of fifteen percent. Overall, this code serves as the foundation for a straightforward tip calculator application that lets users enter the amount of the bill, change the tip %, and divide the total among a predetermined number of recipients.

Code:

eclipse-workspace - CS602/TipCalculatorController.java - Eclipse IDE

```

Project Explorer X
File JavaGradeBookTest.class
File JavaProgramTest.class
File JavaProgram.java
File JavaScript.java
File Payable.class
File PayableInterfaceTest.class
File PayableInterfaceTest.java
File SalariedEmployee.class
File TicTacToe.class
File TicTacToe.java
File TicTacToe$Cell.class
File TipCalculator.class
File TipCalculator.fxml
File TipCalculator.java
File TipCalculatorController.class
File TipCalculatorController$1.class
File Validate.class
File Validate.java
File ValidateInput.class
File ValidateInput.java
File WordCount.class
CS602HW
JRE System Library [JavaSE-17]
src
└─ (default package)
    module-info.java
homework2
javavhw
src
└─ application
    └─ TipCalculatorController.java

PayableInterface Validate ValidateInput TipCalculatorCo TipCalculator TipCalculator.f Main.java TipCalculator.c WordCount.java

1
import java.math.BigDecimal;
2
import java.math.RoundingMode;
3
import java.text.NumberFormat;
4
import javafx.beans.value.ChangeListener;
5
import javafx.beans.value.ObservableValue;
6
import javafx.event.ActionEvent;
7
import javafx.fxml.FXML;
8
import javafx.scene.control.Label;
9
import javafx.scene.control.TextField;
10 import javafx.scene.control.TextFormatter;
11 import javafx.scene.control.TextFormatter.Change;
12 /**
13 * Grabs the user input from the text fields. Attempts to parse the String input
14 * into numeric values. Errors will display if not valid numbers. Calculates the
15 * check total with tip amount as well as a total amount per person.
16 *
17 * @author Christina Austin
18 * @author Cory Siebler
19 */
20 public class TipCalculatorController {
21
22     // Formatters for currency and percentages
23     private static final NumberFormat currency = NumberFormat.getCurrencyInstance();
24     private static final NumberFormat percent = NumberFormat.getPercentInstance();
25
26     private BigDecimal tipPercentage = new BigDecimal(.15); // 15% default
27
28     /**
29      * The @FXML annotation preceding an instance variable indicates that the
30      * variable's name is used in the FXML file that describes the app's GUI.
31     */
32     @FXML
33     private TextField partyTextField;
34
35     @FXML
36     private TextField amountTextField;
37
38     @FXML
39     private Label tipPercentageLabel;
40
41     @FXML
42     private Slider tipPercentageSlider;
43
44     @FXML
45     private TextField tipTextField;
46
47     @FXML
48     private TextField perPersonTextField;
49
50     @FXML
51     private TextField totalTextField;
52
53     /**
54      * A Button's event handler receives an ActionEvent, which indicates that
55      * the button was clicked.
56      */
57     /**
58      * The @FXML annotation preceding a method indicates that the method can be
59
60
61
62     private void calculateButtonPressed(ActionEvent event) {
63         try {
64             BigDecimal size = new BigDecimal(Integer.parseInt(partyTextField.getText()));
65             BigDecimal amount = new BigDecimal(amountTextField.getText());
66             BigDecimal tip = amount.multiply(tipPercentage);
67             BigDecimal total = amount.add(tip);
68             BigDecimal perPerson = total.divide(size);
69
70             tipTextField.setText(currency.format(tip));
71             perPersonTextField.setText(currency.format(perPerson));
72             totalTextField.setText(currency.format(total));
73
74         } catch (NumberFormatException ex) {
75             // If the party size is not a valid integer
76             if (!partyTextField.getText().matches("\\d+")) {
77                 // Invalid Party Size Input
78                 partyTextField.setText("Enter whole value");
79                 partyTextField.selectAll();
80                 partyTextField.requestFocus();
81             } else {
82                 // Invalid Check Amount Input
83                 amountTextField.setText("Enter amount");
84                 amountTextField.selectAll();
85                 amountTextField.requestFocus();
86             }
87         }
88
89     /**
90      * Called by FXMLLoader to initialize the controller
91     */
92     public void initialize() {
93         // Round down, 5.0 rounds up
94         currency.setRoundingMode(RoundingMode.HALF_UP);
95
96         // Listener for changes to tipPercentageSlider's value
97         tipPercentageSlider.valueProperty().addListener(new ChangeListener<Number>() {
98
99             /**
100             * @param ov
101             * @param oldValue
102             * @param newValue
103             */
104             @Override
105             public void changed(ObservableValue<? extends Number> ov, Number oldValue, Number newValue) {
106                 tipPercentage = BigDecimal.valueOf(newValue.intValue() / 100.0);
107                 tipPercentageLabel.setText(percent.format(tipPercentage));
108             }
109         });
110     };
111 }
112 }
```

Problems Javadoc Declaration Console Coverage

<terminated> TipCalculator [Java Application] /Users/ck/p2/pool/plugins/org.eclipse.jst/openjdk.hotspot.jre.full.macosx.aarch64\_17.0.10.v20240120-1143/rebinjava (Mar 23, 2024, 12:10:49 PM - 12:10:50 PM) [pid: 40761]

eclipse-workspace - CS602/TipCalculatorController.java - Eclipse IDE

```

Project Explorer X
File JavaGradeBookTest.class
File JavaProgramTest.class
File JavaProgram.java
File JavaScript.java
File Payable.class
File PayableInterfaceTest.class
File PayableInterfaceTest.java
File SalariedEmployee.class
File TicTacToe.class
File TicTacToe.java
File TicTacToe$Cell.class
File TipCalculator.class
File TipCalculator.fxml
File TipCalculator.java
File TipCalculatorController.class
File TipCalculatorController$1.class
File Validate.class
File Validate.java
File ValidateInput.class
File ValidateInput.java
File WordCount.class
CS602HW
JRE System Library [JavaSE-17]
src
└─ (default package)
    module-info.java
homework2
javavhw
src
└─ application
    └─ TipCalculatorController.java

PayableInterface Validate ValidateInput TipCalculatorCo TipCalculator TipCalculator.f Main.java TipCalculator.c WordCount.java

56
57     /**
58      * The @FXML annotation preceding a method indicates that the method can be
59      * used to specify a control's event handler in the FXML file that describes
60      * the app's GUI.
61
62     private void calculateButtonPressed(ActionEvent event) {
63         try {
64             BigDecimal size = new BigDecimal(Integer.parseInt(partyTextField.getText()));
65             BigDecimal amount = new BigDecimal(amountTextField.getText());
66             BigDecimal tip = amount.multiply(tipPercentage);
67             BigDecimal total = amount.add(tip);
68             BigDecimal perPerson = total.divide(size);
69
70             tipTextField.setText(currency.format(tip));
71             perPersonTextField.setText(currency.format(perPerson));
72             totalTextField.setText(currency.format(total));
73
74         } catch (NumberFormatException ex) {
75             // If the party size is not a valid integer
76             if (!partyTextField.getText().matches("\\d+")) {
77                 // Invalid Party Size Input
78                 partyTextField.setText("Enter whole value");
79                 partyTextField.selectAll();
80                 partyTextField.requestFocus();
81             } else {
82                 // Invalid Check Amount Input
83                 amountTextField.setText("Enter amount");
84                 amountTextField.selectAll();
85                 amountTextField.requestFocus();
86             }
87         }
88
89     /**
90      * Called by FXMLLoader to initialize the controller
91     */
92     public void initialize() {
93         // Round down, 5.0 rounds up
94         currency.setRoundingMode(RoundingMode.HALF_UP);
95
96         // Listener for changes to tipPercentageSlider's value
97         tipPercentageSlider.valueProperty().addListener(new ChangeListener<Number>() {
98
99             /**
100             * @param ov
101             * @param oldValue
102             * @param newValue
103             */
104             @Override
105             public void changed(ObservableValue<? extends Number> ov, Number oldValue, Number newValue) {
106                 tipPercentage = BigDecimal.valueOf(newValue.intValue() / 100.0);
107                 tipPercentageLabel.setText(percent.format(tipPercentage));
108             }
109         });
110     };
111 }
112 }
```

Problems Javadoc Declaration Console Coverage

<terminated> TipCalculator [Java Application] /Users/ck/p2/pool/plugins/org.eclipse.jst/openjdk.hotspot.jre.full.macosx.aarch64\_17.0.10.v20240120-1143/rebinjava (Mar 23, 2024, 12:10:49 PM - 12:10:50 PM) [pid: 40761]

Description:

The file's initial XML declaration contains the version and encoding specification. The file declares namespaces for JavaFX and FXML in order to take advantage of JavaFX UI components and attributes unique to FXML.

A GridPane is a layout container in JavaFX that arranges its offspring in a grid. This element is the FXML file's root.

The GridPane element's alignment and inter-child spacing are controlled by its alignment, vgap, and hgap properties.

The fx:controller field returns the full qualified name of the controller class (TipCalculatorController) associated with this FXML file.

Within the GridPane element, there are numerous child components representing various tip calculator user interface elements.

"Total," "Party Size," "Amount," "Tip," "Per Person," and so forth are examples of text labels that can be applied to components.

User input is received and calculated values are displayed using the TextField elements partyTextField, amountTextField, tipTextField, perPersonTextField, and totalTextField.

A slider element called tipPercentageSlider lets you adjust the tip percentage. CalculateButton is a button element that initiates the calculation.

The parameters rowConstraints and columnConstraints define the column and row sizing bounds of the GridPane.

By defining the padding surrounding the GridPane, the padding element provides the gap between the children of the grid and its edges.

Code:

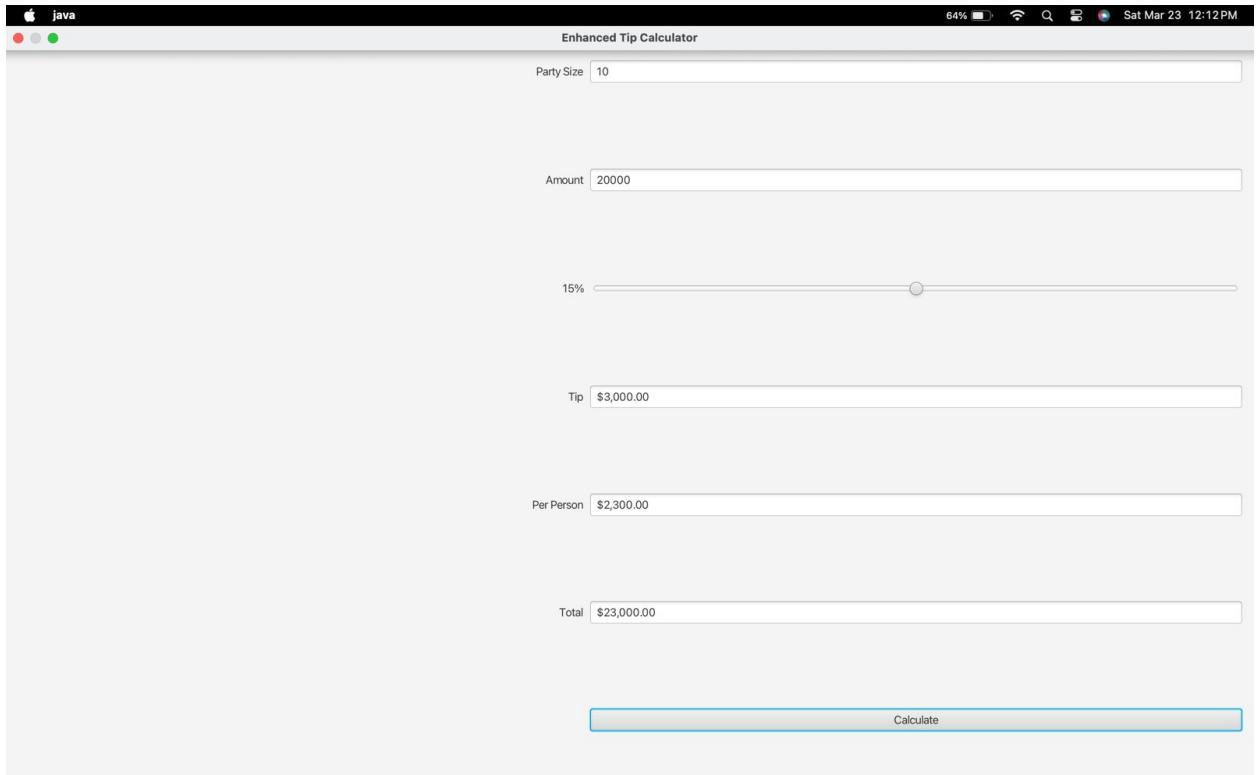
eclipse-workspace - CS602/TipCalculator.fxml - Eclipse IDE

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE GridPane [
    <!-- Party Size -->
    <Label text="Party Size" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="0" />
    <Label text="Amount" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="1" />
    <Label text="Tip percentage" text="15%" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="2" />
    <Label text="Tip" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="3" />
    <Label text="Per Person" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="4" />
    <Label text="Total" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="5" />
    <Textfield id="partySize" fx:id="partyTextField" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="0" />
    <Textfield id="amountText" fx:id="amountTextField" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="1" />
    <Textfield fx:id="tipTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="2" />
    <Textfield fx:id="perPersonTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="4" />
    <Textfield fx:id="totalTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="5" />
    <Button id="calculateButton" maxWidth="1.797693148623157e308" mnemonicParsing="false" onAction="#calculateButtonPressed" text="Calculate" GridPane.columnIndex="1" GridPane.rowIndex="6" />
]
><GridPane alignment="TOP_LEFT" hgap="8.0" vgap="0.0" xmlns:fx="http://javafx.com/xml/1" xmlns="http://javafx.com/javafx/2.2" fx:controller="TipCalculatorController">
<children>
    <Label text="Party Size" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="0" />
    <Label text="Amount" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="1" />
    <Label text="Tip percentage" text="15%" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="2" />
    <Label text="Tip" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="3" />
    <Label text="Per Person" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="4" />
    <Label text="Total" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="5" />
    <Textfield id="partySize" fx:id="partyTextField" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="0" />
    <Textfield id="amountText" fx:id="amountTextField" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="1" />
    <Textfield fx:id="tipTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="2" />
    <Textfield fx:id="perPersonTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="4" />
    <Textfield fx:id="totalTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="5" />
    <Button id="calculateButton" maxWidth="1.797693148623157e308" mnemonicParsing="false" onAction="#calculateButtonPressed" text="Calculate" GridPane.columnIndex="1" GridPane.rowIndex="6" />
</children>
<columnConstraints>
    <ColumnConstraints halign="RIGHT" alignment="RIGHT" hgrow="SOMETIMES" minWidth="10.0" prefWidth="1.0" />
    <ColumnConstraints halign="RIGHT" alignment="RIGHT" hgrow="SOMETIMES" minWidth="10.0" prefWidth="1.0" />
</columnConstraints>
<rowConstraints>
    <RowConstraints minHeight="10.0" prefHeight="39.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="39.0" vgrow="SOMETIMES" />
</rowConstraints>
</gridpane>

```

## Output:



f. How Fig. 14.20 works, pages 545-547 (1 point)

Answer:

Description:

Java.util.Scanner is imported by the program in order to enable console-based user input. The class Validate contains the declaration for the main method, which serves as the program's entry point.

This method starts by creating a Scanner object in order to read user input from the console.

First and last names, address, zip code, city, state, and phone number must all be entered by the user individually.

Each input is saved in the relevant variables (firstName, lastName, address, city, state, zip, phone) using the scanner.nextLine() method.

The program validates each piece of information after gathering user input.

The phone number, address, city, state, zip code, first and last name, and address are all validated using static methods from the ValidateInput class.

An error message describing the type of erroneous input is shown to the console if any input is found to be invalid.

The validation result for each input is printed by the software.

A message expressing gratitude to the user for submitting accurate information is printed if all inputs are genuine.

Code:

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Eclipse, File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Title Bar:** eclipse-workspace - CS602/Validate.java - Eclipse IDE
- Left Sidebar (Package Explorer):** Shows the project structure under CS602, including JRE System Library [JavaSE-17] and various Java files like Analysis.java, Craps.java, FairTax.java, GradeBookTest.java, HelloWord.java, JavaApplet.java, JavaProgram.java, JavaScript.java, PayableInterfaceTest.java, TicTactoe.java, Validate.java, and ValidateInput.java.
- Central Area (Code Editor):** Displays the content of Validate.java. The code uses a Scanner to read user input for first name, last name, address, city, state, zip code, and phone number, then prints validation results for each.
- Bottom Area (Console):** Shows the terminal output of the program's execution. The user enters their information, and the program prints validation messages for each field.
- Mac OS Dock:** At the bottom, showing various application icons.

```

import java.util.Scanner;
public class Validate {
    public static void main(String[] args) {
        // get the input from the user
        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter your first name: ");
        String firstName = scanner.nextLine();
        System.out.println("Please enter your last name: ");
        String lastName = scanner.nextLine();
        System.out.println("Please enter your address: ");
        String address = scanner.nextLine();
        System.out.println("Please enter your city: ");
        String city = scanner.nextLine();
        System.out.println("Please enter your state: ");
        String state = scanner.nextLine();
        System.out.println("Please enter your zip code: ");
        String zip = scanner.nextLine();
        System.out.println("Please enter your phone number: ");
        String phone = scanner.nextLine();
        // validate the input
        System.out.print("Validate Result:");
        if (ValidateInput.validateFirstName(firstName)) {
            System.out.println("Valid first name");
        } else if (!ValidateInput.validateFirstName(firstName)) {
            System.out.println("Invalid first name");
        } else if (ValidateInput.validateAddress(address)) {
            System.out.println("Valid address");
        } else if (!ValidateInput.validateAddress(address)) {
            System.out.println("Invalid address");
        } else if (ValidateInput.validateCity(city)) {
            System.out.println("Valid city");
        } else if (!ValidateInput.validateCity(city)) {
            System.out.println("Invalid city");
        } else if (ValidateInput.validateState(state)) {
            System.out.println("Valid state");
        } else if (!ValidateInput.validateState(state)) {
            System.out.println("Invalid state");
        } else if (ValidateInput.validateZip(zip)) {
            System.out.println("Valid zip code");
        } else if (!ValidateInput.validateZip(zip)) {
            System.out.println("Invalid zip code");
        } else if (ValidateInput.validatePhone(phone)) {
            System.out.println("Valid phone number");
        } else {
            System.out.println("Invalid phone number");
        }
    }
}

```

## Description:

utilizes the regular expression pattern `[A-Z].[a-z]*` to ensure that at least one lowercase character and zero uppercase character follow the initial name.

Verify the most recent name that was entered.

makes use of the regular expression pattern `[a-zA-Z]+([-][a-zA-Z]+)*` to allow letters, hyphens, and apostrophes in the last name.  
verifies the entered address.

utilizes the regular expression pattern `\d+\s+` to match the street name(s) and house number.(<\s[a-zA-Z]+[a-zA-Z]+|[a-zA-Z]+).

verifies the input for the city.

matches single or multiple-word city names using a regular expression pattern `([a-zA-Z]+|[a-zA-Z]+\s[a-zA-Z]+)`.

verifies the input for the state.

matches single or multiword state names using the same regular expression pattern as validateCity.

verifies the entered zip code.

ensures that the zip code has precisely five digits by using the regular expression pattern `\d{5}`.

Checks the phone number that was entered.

matches, using a regular expression pattern, the format of a normal US phone number in the pattern `###-###-#### [1-3]\d{2}-[1-3]\d{2}-\d{4}`.

Code:

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Eclipse, File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. Date: Sat Mar 23 12:19:AM.
- Left Sidebar (Package Explorer):** Shows the project structure for CS602, including packages like CS602, JRE System Library [JavaSE-17], and CS602HOne, along with various Java files.
- Central Area:** The code editor displays `ValidateInput.java` with the following content:

```
1 public class ValidateInput {
2     public static boolean validateFirstName(String firstName) {
3         return firstName.matches("[A-Z][a-z]*");
4     }
5
6     public static boolean validateLastName(String lastName) {
7         return lastName.matches("[a-zA-Z]+[A-Z][a-zA-Z]*");
8     }
9
10    public static boolean validateAddress(String address) {
11        return address.matches("\\d+\\s+[a-zA-Z]+[a-zA-Z]+\\s[a-zA-Z]+");
12    }
13
14    public static boolean validateCity(String city) {
15        return city.matches("[a-zA-Z]+[a-zA-Z]+\\s[a-zA-Z]+");
16    }
17
18    public static boolean validateState(String state) {
19        return state.matches("[a-zA-Z]+[a-zA-Z]+\\s[a-zA-Z]+");
20    }
21
22    public static boolean validateZip(String zip) {
23        return zip.matches("\\d{5}");
24    }
25
26    public static boolean validatePhone(String phone) {
27        return phone.matches("[1-9]\\d{2}-[1-9]\\d{2}-\\d{4}");
28    }
29 }
```

- Bottom Area:** Shows the execution output in the Console tab, displaying the user input and the program's response.

Output:

The terminal window shows the following interaction:

```
<terminated> Validate [Java Application] /Users/ck/p2/pool/plugins/org.eclipse.just.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (Mar 23, 2024, 9:25:37 AM - 9:27:31 AM) [pid: 28756]
Please enter your first name:
Charitha
Please enter your last name:
Kannan
Please enter your address:
126 Newark Avenue
Please enter your city:
Jersey City
Please enter your state:
NJ
Please enter your zip code:
07306
Please enter your phone number:
574-725-9893
Validate Result:Valid input. Thanks you.
```

h. How Fig. 16.17 works, pages 632-634 (1 point)

Answer:

Description:

The required classes, Map, HashMap, Set, TreeSet, and Scanner, are imported by the application from the java.util package.

The program's entry point, the main method, is declared in the class WordCount. A HashMap with the name myMap is initialized by the main method.

Then, it invokes two methods: displayMap(myMap) to output the contents of the map, and createMap(myMap) to add word counts to the map.

This method involves the user entering a string.

The input text is split into tokens, or words, using the space character as a delimiter.

Before moving on, the procedure lowerscases every token and makes sure it's in the map.

If a term is already on the map, it gains more points. If not, the word on the map gets one count added to it.

In this way, the contents of the map are displayed.

The keys are sorted using a TreeSet after being removed from the map.

After sorting the keys, iterating through them, it prints each key-value pair in an organized manner.

The size and empty status of the map are printed at the end.

Code:

Eclipse IDE - CS602/WordCount.java

```

public class WordCount {
    public static void main(String[] args) {
        Map<String, Integer> myMap = new HashMap<>();
        createMap(myMap);
        displayMap(myMap);
    }

    private static void createMap(Map<String, Integer> map) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string:");
        String input = scanner.nextLine();
        String[] tokens = input.split(" ");
        for (String token : tokens) {
            String word = token.toLowerCase();
            if (map.containsKey(word)) {
                int count = map.get(word);
                map.put(word, count + 1);
            } else {
                map.put(word, 1);
            }
        }
    }

    private static void displayMap(Map<String, Integer> map) {
        Set<String> keys = map.keySet();
        TreeSet<String> sortedKeys = new TreeSet<>(keys);
        System.out.printf("%nMap contains:%n");
        for (String key : sortedKeys) {
            System.out.printf("%-10s%03d%n", key, map.get(key));
        }
        System.out.printf(
            "%nsize: %d%nisEmpty: %b%n", map.size(), map.isEmpty());
    }
}

Enter a string:
hello world

Map contains:
Key          Value
hello        1
world        1

size: 2
isEmpty: false

```

Output:

Java Application

```

Map contains:
Key          Value
hello        1
world        1

size: 2
isEmpty: false

```