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.

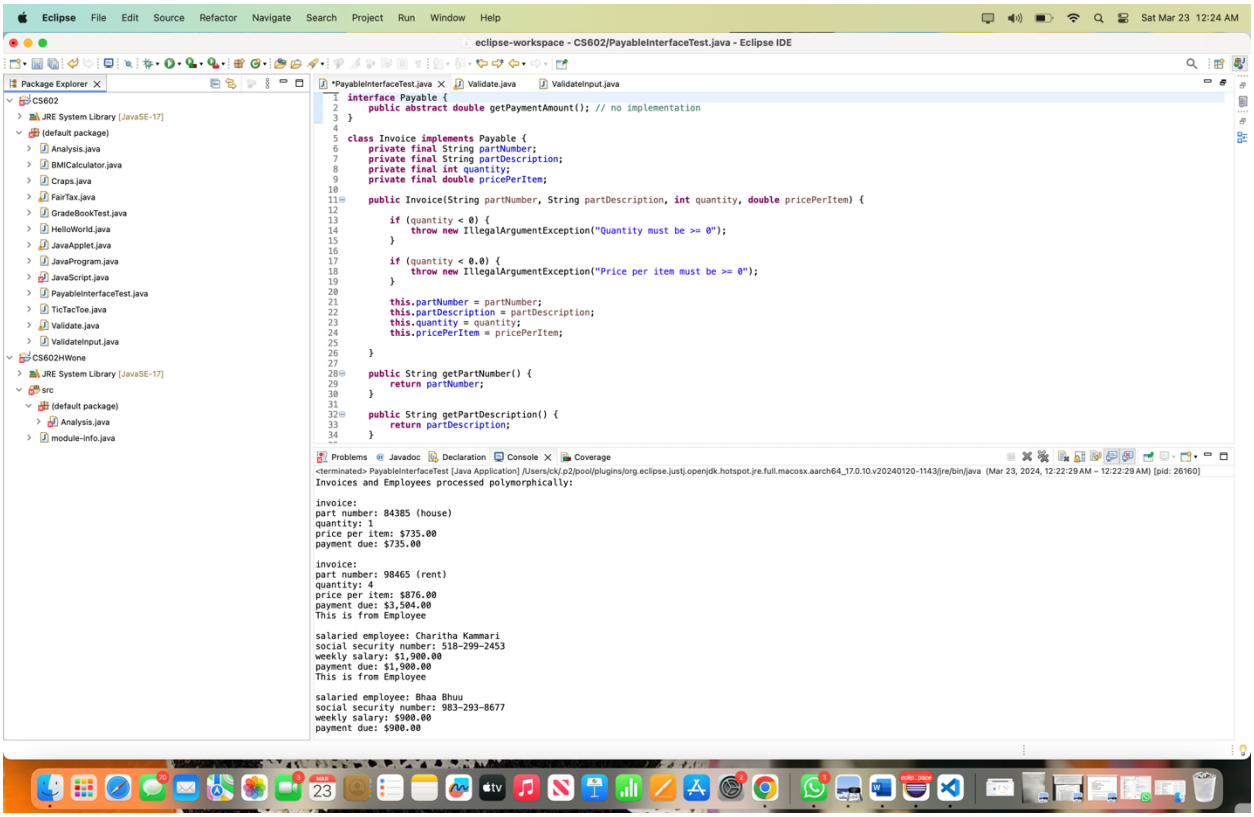
A certain subset of workers who receive a weekly salary.
utilizes the earnings() function to return the salary for that particular week and has a property for the weekly wage.
By using a toString() method override, a string representation of the salaried employee is supplied.

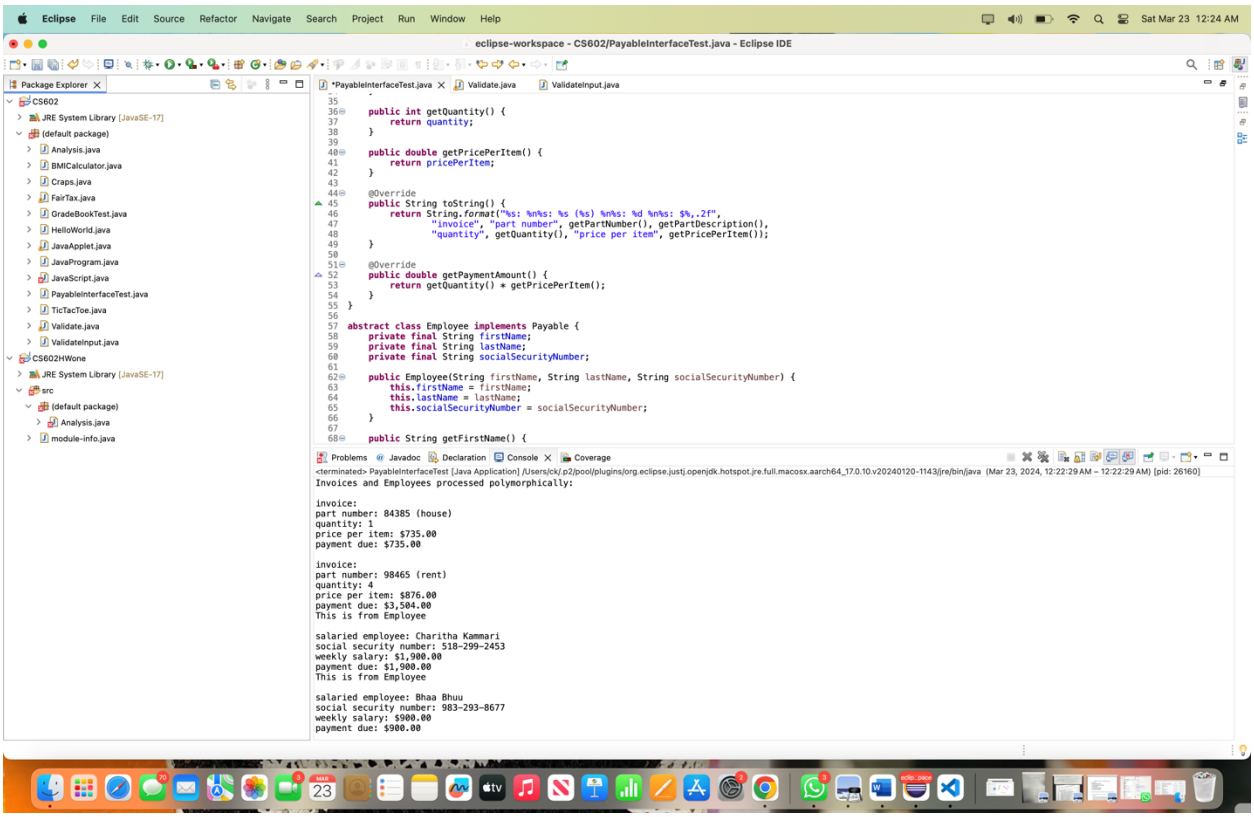
The PayableInterfaceTest main class's main() method is used to test the operation of the classes.
creates instances of SalariedEmployee and Invoice among other Payable objects.
The code loops through the array, invoking the toString() and getPaymentAmount() methods polymorphically for each item in order to demonstrate polymorphism and dynamic method binding.

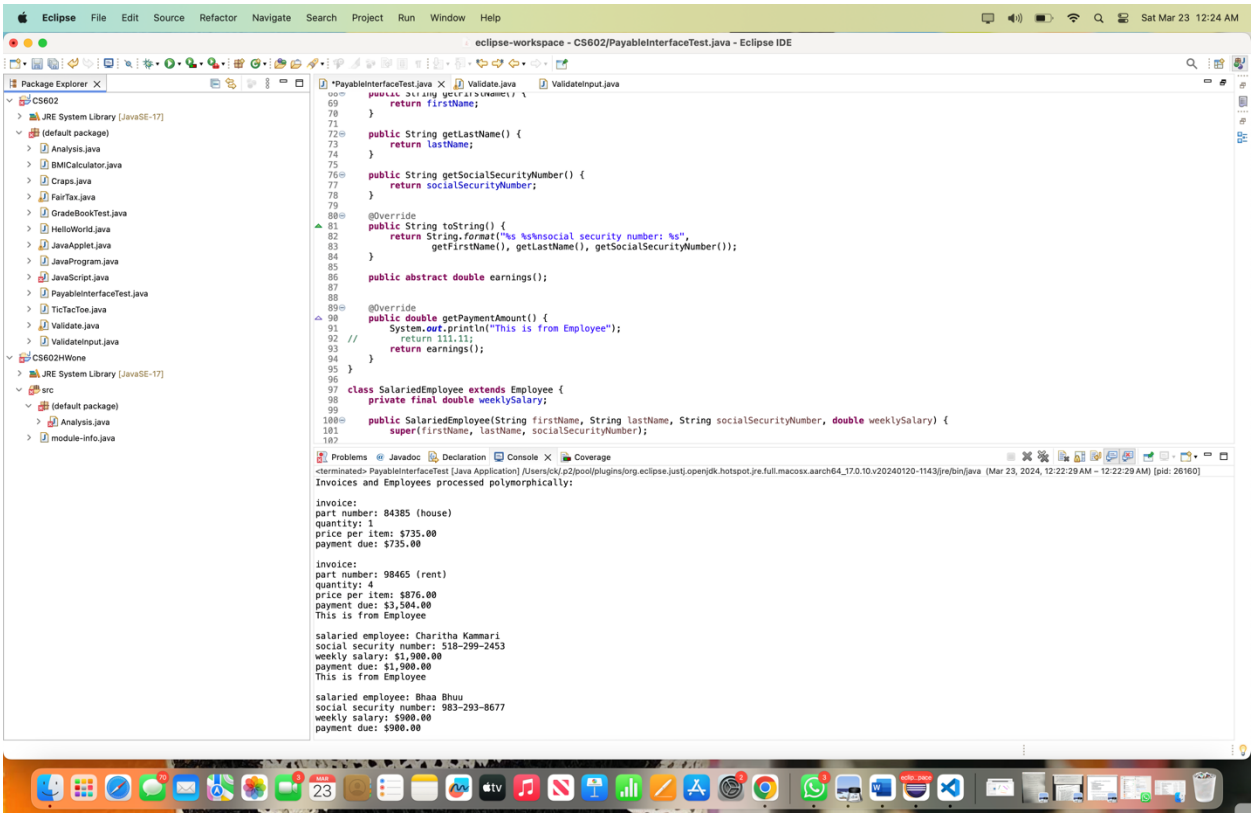
Functionality:

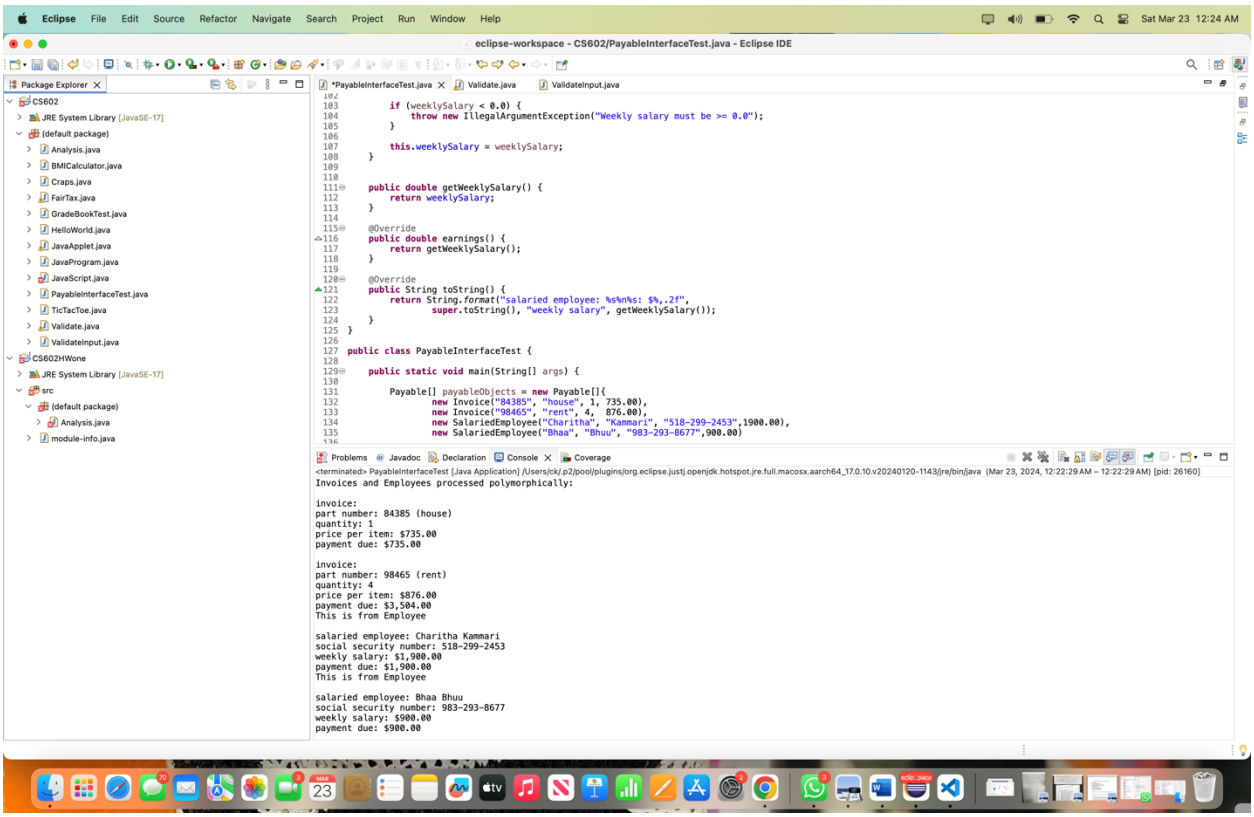
Using the Payable interface, the program demonstrates polymorphic behavior, treating different object kinds (salaried staff and bills) identically.
The getPaymentAmount() function calculates the payment amount for each object using its own logic, which is quantity * price per item for invoices and weekly 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:

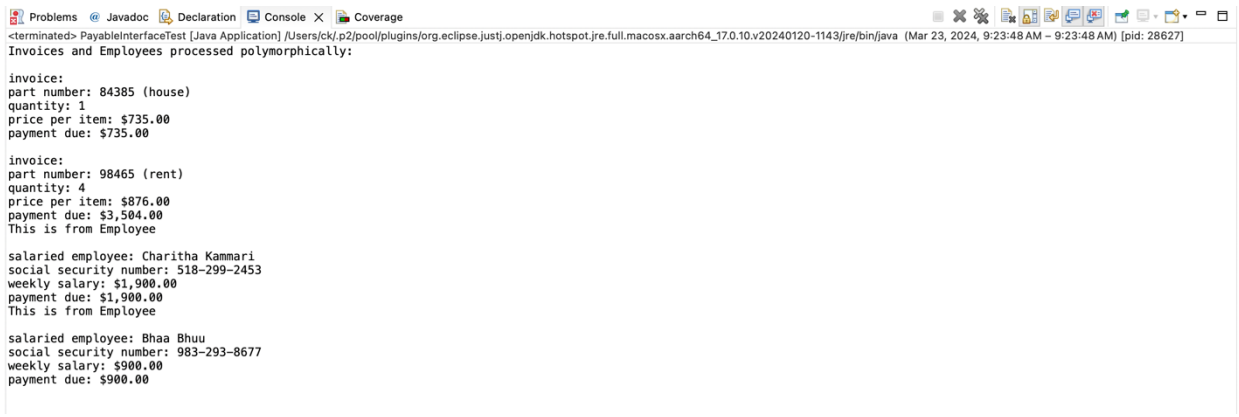



eclipse-workspace - CS602/PayableInterfaceTest.java - Eclipse IDE
Package Explorer
CS602
JRE System Library [JavaSE-17]
(default package)
Analysis.java
BMICalculator.java
Craps.java
FairTax.java
GradeBookTest.java
HelloWorld.java
JavaApplet.java
JavaProgram.java
JavaScript.java
PayableInterfaceTest.java
TicTacToe.java
Validate.java
ValidateInput.java
CS602HWone
JRE System Library [JavaSE-17]
src
(default package)
Analysis.java
module-info.java
*PayableInterfaceTest.java
Validate.java
ValidateInput.java
Problems
Javadoc
Declaration
Console
Coverage
<terminated> PayableInterfaceTest [Java Application] /Users/ck/.p2/pool/plugins/org.eclipse.justj.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


eclipse-workspace - CS602/PayableInterfaceTest.java - Eclipse IDE
Package Explorer
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
CS602HWone
JRE System Library [JavaSE-17]
src
(default package)
Analysis.java
module-info.java
*PayableInterfaceTest.java
Validate.java
ValidateInput.java
Problems
Javadoc
Declaration
Console
Coverage
<terminated> PayableInterfaceTest [Java Application] /Users/ck/.p2/pool/plugins/org.eclipse.justj.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:



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.

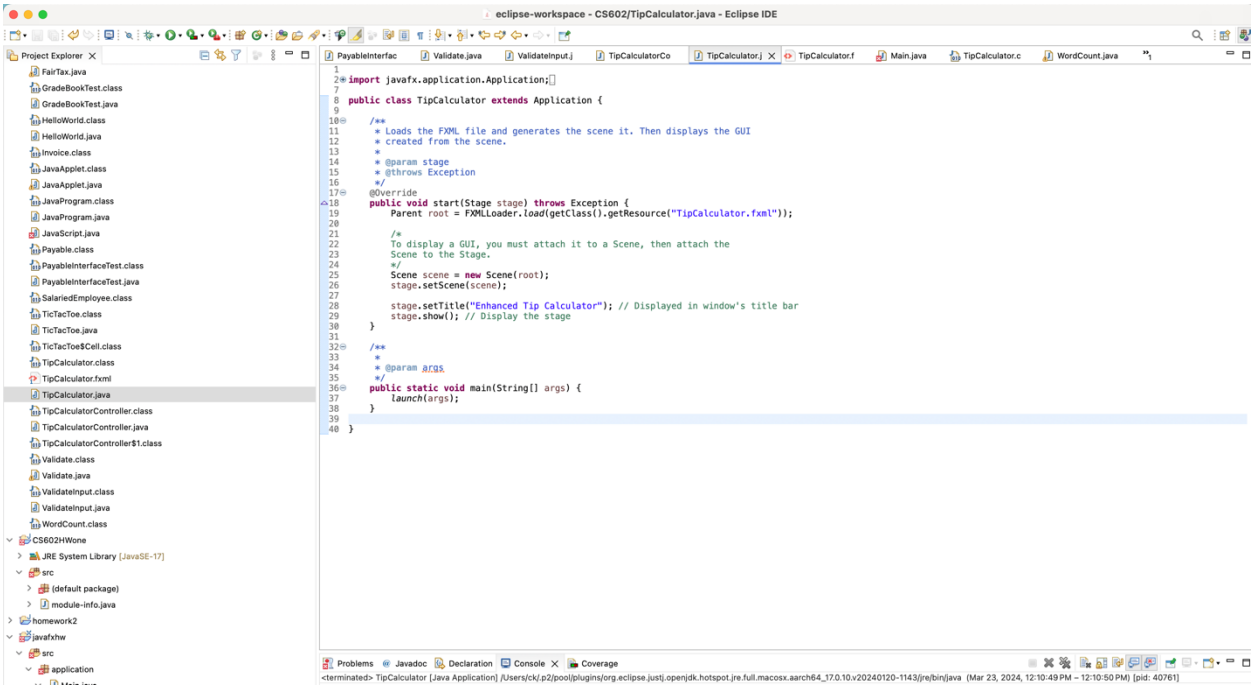
The "Enhanced Tip Calculator" is the new title for the stage.

It concludes by calling stage.The stage is displayed with show().

The application enters through this technique.

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

Code:



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.

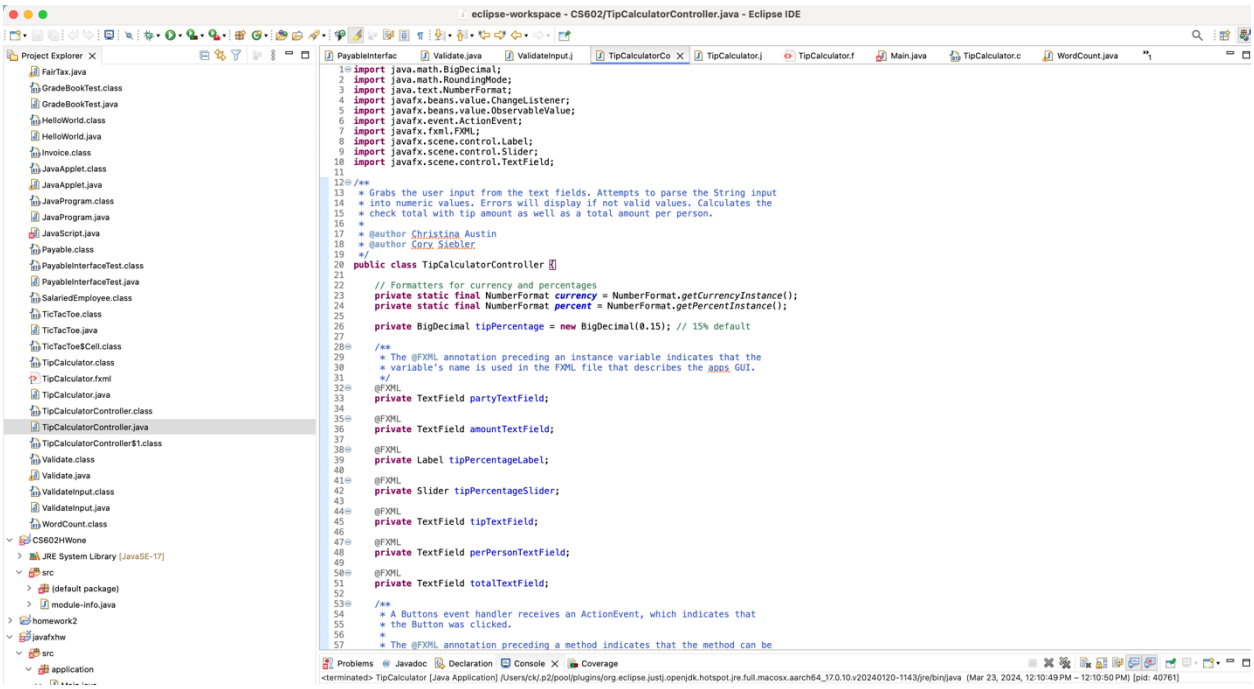
It is likely that this class will function as TipCalculatorController, the controller for the JavaFX application.

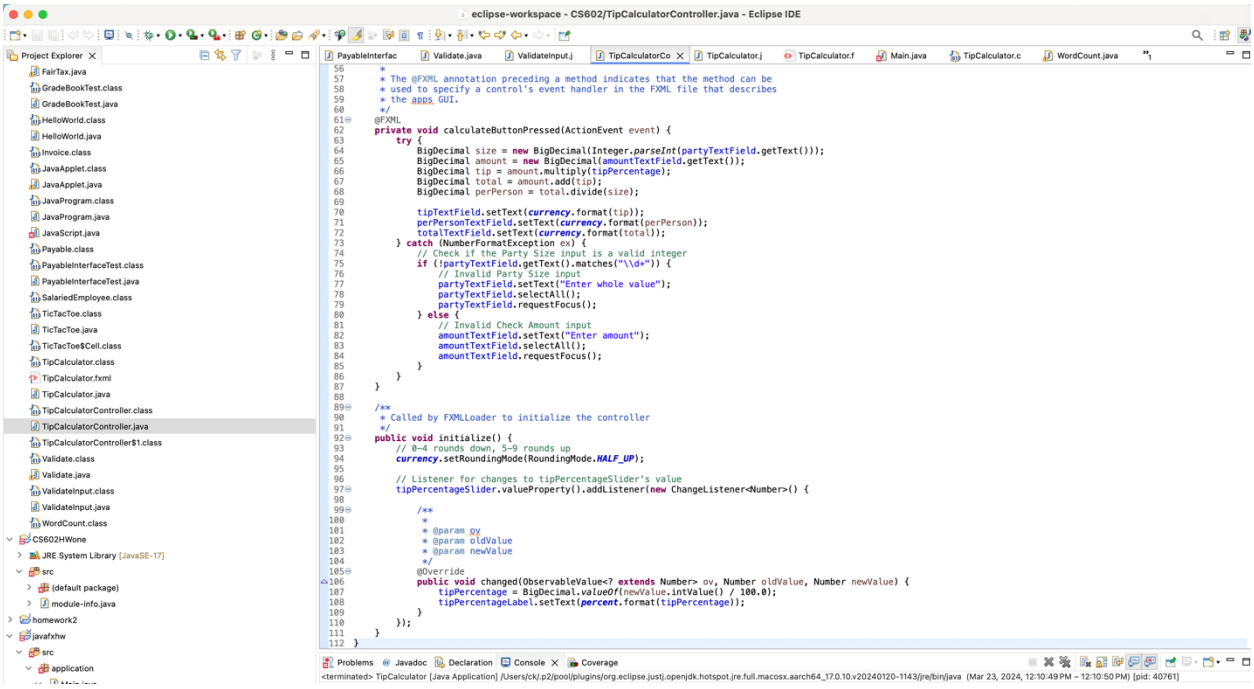
The class contains @FXML annotations on its instance variables. For user interface components, these variables correspond to those specified in a linked FXML file. They are composed of:
collectingTextField: TextField used to indicate the numerical size of the party.
overallIn this TextField, enter the 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
FairTax.java
GradeBookTest.class
GradeBookTest.java
HelloWorld.class
HelloWorld.java
Invoice.class
JavaApplet.class
JavaApplet.java
JavaProgram.class
JavaProgram.java
JavaScript.java
Payable.class
PayableInterfaceTest.class
PayableInterfaceTest.java
SalariedEmployee.class
TicTacToe.class
TicTacToe.java
TicTacToe$Cell.class
TipCalculator.class
TipCalculator.fxml
TipCalculator.java
TipCalculatorController.class
TipCalculatorController.java
TipCalculatorController$1.class
Validate.class
Validate.java
ValidateInput.class
ValidateInput.java
WordCount.class
CS602HWone
JRE System Library [JavaSE-17]
src
(default package)
module-info.java
homework2
javafxhw
src
application
PayableInterfac
Validate.java
ValidateInput.j
TipCalculatorCo
TipCalculator.j
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.Slider;
10 import javafx.scene.control.TextField;
11
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 values. 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(0.15); // 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 apps 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 Buttons event handler receives an ActionEvent, which indicates that
55 * the Button was clicked.
56 *
57 * The @FXML annotation preceding a method indicates that the method can be
Problems
Javadoc
Declaration
Console
Coverage
<terminated> TipCalculator [Java Application] /Users/ck/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (Mar 23, 2024, 12:10:49 PM – 12:10:50 PM) [pid: 40761]


eclipse-workspace - CS602/TipCalculatorController.java - Eclipse IDE
Project Explorer
FairTax.java
GradeBookTest.class
GradeBookTest.java
HelloWorld.class
HelloWorld.java
Invoice.class
JavaApplet.class
JavaApplet.java
JavaProgram.class
JavaProgram.java
JavaScript.java
Payable.class
PayableInterfaceTest.class
PayableInterfaceTest.java
SalariedEmployee.class
TicTacToe.class
TicTacToe.java
TicTacToe$Cell.class
TipCalculator.class
TipCalculator.fxml
TipCalculator.java
TipCalculatorController.class
TipCalculatorController.java
TipCalculatorController$1.class
Validate.class
Validate.java
ValidateInput.class
ValidateInput.java
WordCount.class
CS602HWone
JRE System Library [JavaSE-17]
src
(default package)
module-info.java
homework2
javafxhw
src
application
PayableInterfac
Validate.java
ValidateInput.j
TipCalculatorCo
TipCalculator.j
TipCalculator.f
Main.java
TipCalculator.c
WordCount.java
56 *
57 * The @FXML annotation preceding a method indicates that the method can be
58 * used to specify a control's event handler in the FXML file that describes
59 * the apps GUI.
60 */
61 @FXML
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 } catch (NumberFormatException ex) {
74 // Check if the Party Size input is a valid integer
75 if (!partyTextField.getText().matches("\\d+")) {
76 // Invalid Party Size input
77 partyTextField.setText("Enter whole value");
78 partyTextField.selectAll();
79 partyTextField.requestFocus();
80 } else {
81 // Invalid Check Amount input
82 amountTextField.setText("Enter amount");
83 amountTextField.selectAll();
84 amountTextField.requestFocus();
85 }
86 }
87 }
88
89 /**
90 * Called by FXMLLoader to initialize the controller
91 */
92 public void initialize() {
93 // 0-4 rounds down, 5-9 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 *
101 * @param ov
102 * @param oldValue
103 * @param newValue
104 */
105 @Override
106 public void changed(ObservableValue<? extends Number> ov, Number oldValue, Number newValue) {
107 tipPercentage = BigDecimal.valueOf(newValue.intValue() / 100.0);
108 tipPercentageLabel.setText(percent.format(tipPercentage));
109 }
110 });
111 }
112 }
Problems
Javadoc
Declaration
Console
Coverage
<terminated> TipCalculator [Java Application] /Users/ck/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (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.

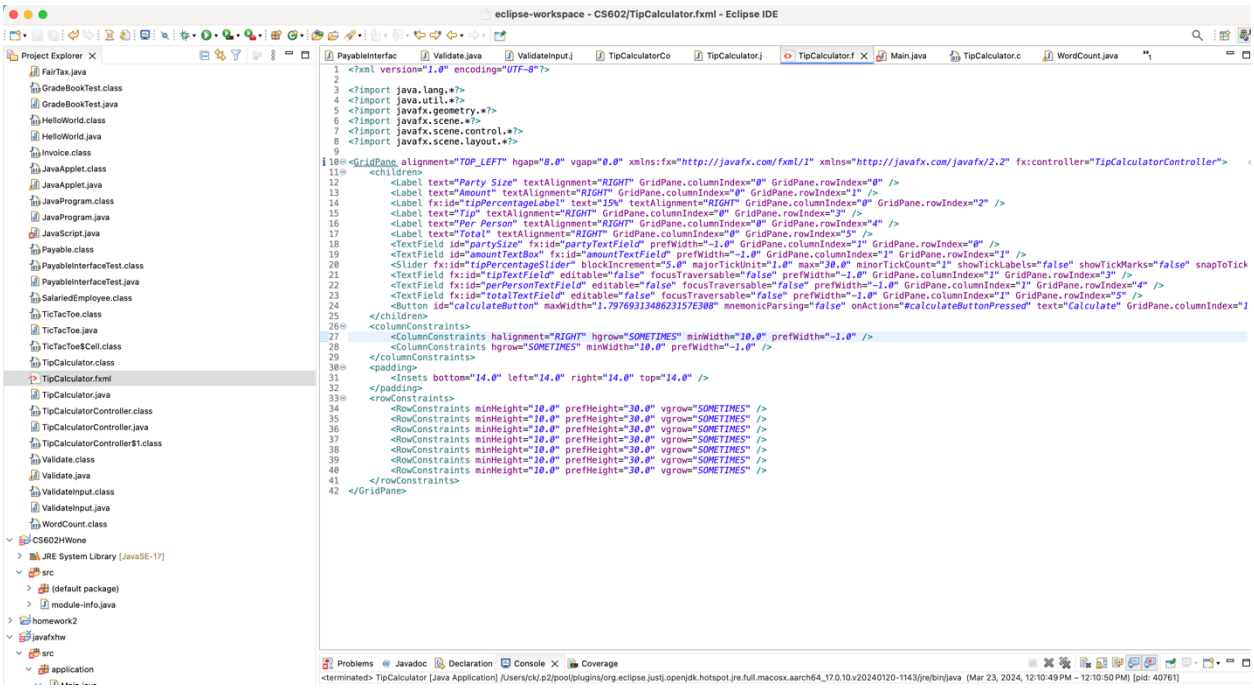
User input is received and calculated values are displayed using the TextField elements partyTextField, amountTextField, tipTextField, perPersonTextField, and totalTextField.

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

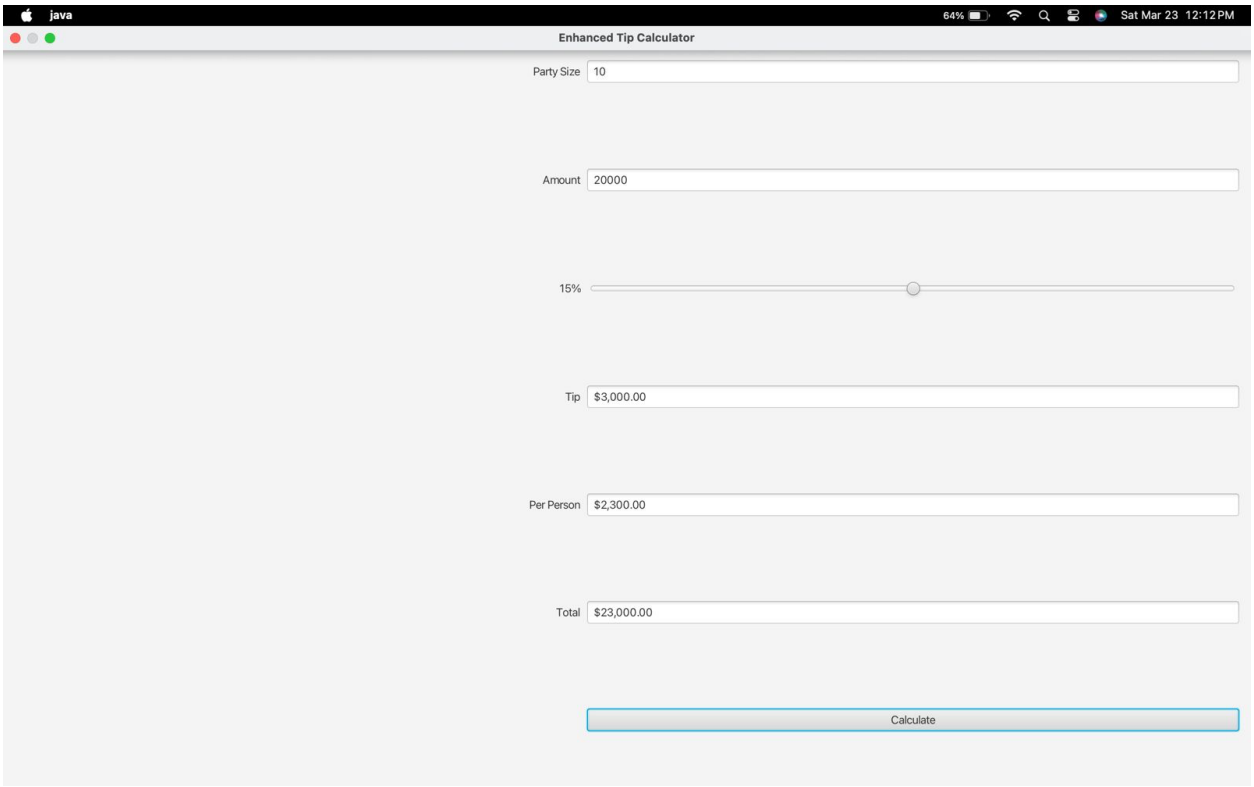
The parameters rowConstraints and columnConstraints define the column and row sizing bounds of the GridPane.

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
Project Explorer
FairTax.java
GradeBookTest.class
GradeBookTest.java
HelloWorld.class
HelloWorld.java
Invoice.class
JavaApplet.class
JavaApplet.java
JavaProgram.class
JavaProgram.java
JavaScript.java
Payable.class
PayableInterfaceTest.class
PayableInterfaceTest.java
SalariedEmployee.class
TicTacToe.class
TicTacToe.java
TicTacToe$Cell.class
TipCalculator.class
TipCalculator.fxml
TipCalculator.java
TipCalculatorController.class
TipCalculatorController.java
TipCalculatorController$1.class
Validate.class
Validate.java
ValidateInput.class
ValidateInput.java
WordCount.class
CS602HWone
JRE System Library [JavaSE-17]
src
(default package)
module-info.java
homework2
javafxhw
src
application
PayableInterfac
Validate.java
ValidateInput.j
TipCalculatorCo
TipCalculator.j
TipCalculator.f
Main.java
TipCalculator.c
WordCount.java
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import java.lang.*?>
4 <?import java.util.*?>
5 <?import javafx.geometry.*?>
6 <?import javafx.scene.*?>
7 <?import javafx.scene.control.*?>
8 <?import javafx.scene.layout.*?>
9
10 <GridPane alignment="TOP_LEFT" hgap="8.0" vgap="0.0" xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/2.2" fx:controller="TipCalculatorController">
11 <children>
12 <Label text="Party Size" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="0" />
13 <Label text="Amount" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="1" />
14 <Label fx:id="tipPercentageLabel" text="15%" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="2" />
15 <Label text="Tip" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="3" />
16 <Label text="Per Person" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="4" />
17 <Label text="Total" textAlignment="RIGHT" GridPane.columnIndex="0" GridPane.rowIndex="5" />
18 <TextField id="partySize" fx:id="partyTextField" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="0" />
19 <TextField id="amountTextBox" fx:id="amountTextField" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="1" />
20 <Slider fx:id="tipPercentageSlider" blockIncrement="5.0" majorTickUnit="1.0" max="30.0" minorTickCount="1" showTickLabels="false" showTickMarks="false" snapToTick
21 <TextField fx:id="tipTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="3" />
22 <TextField fx:id="perPersonTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="4" />
23 <TextField fx:id="totalTextField" editable="false" focusTraversable="false" prefWidth="-1.0" GridPane.columnIndex="1" GridPane.rowIndex="5" />
24 <Button id="calculateButton" maxWidth="1.7976931348623157E308" mnemonicParsing="false" onAction="#calculateButtonPressed" text="Calculate" GridPane.columnIndex="1
25 </children>
26 <columnConstraints>
27 <ColumnConstraints halignment="RIGHT" hgrow="SOMETIMES" minWidth="10.0" prefWidth="-1.0" />
28 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="-1.0" />
29 </columnConstraints>
30 <padding>
31 <Insets bottom="14.0" left="14.0" right="14.0" top="14.0" />
32 </padding>
33 <rowConstraints>
34 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
35 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
36 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
37 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
38 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
39 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
40 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
41 </rowConstraints>
42 </GridPane>
Problems
Javadoc
Declaration
Console
Coverage
<terminated> TipCalculator [Java Application] /Users/ck/p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (Mar 23, 2024, 12:10:49 PM – 12:10:50 PM) [pid: 40761]

Output:


java
64%
Sat Mar 23 12:12 PM
Enhanced Tip Calculator
Party Size
10
Amount
20000
15%
Tip
$3,000.00
Per Person
$2,300.00
Total
$23,000.00
Calculate

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.

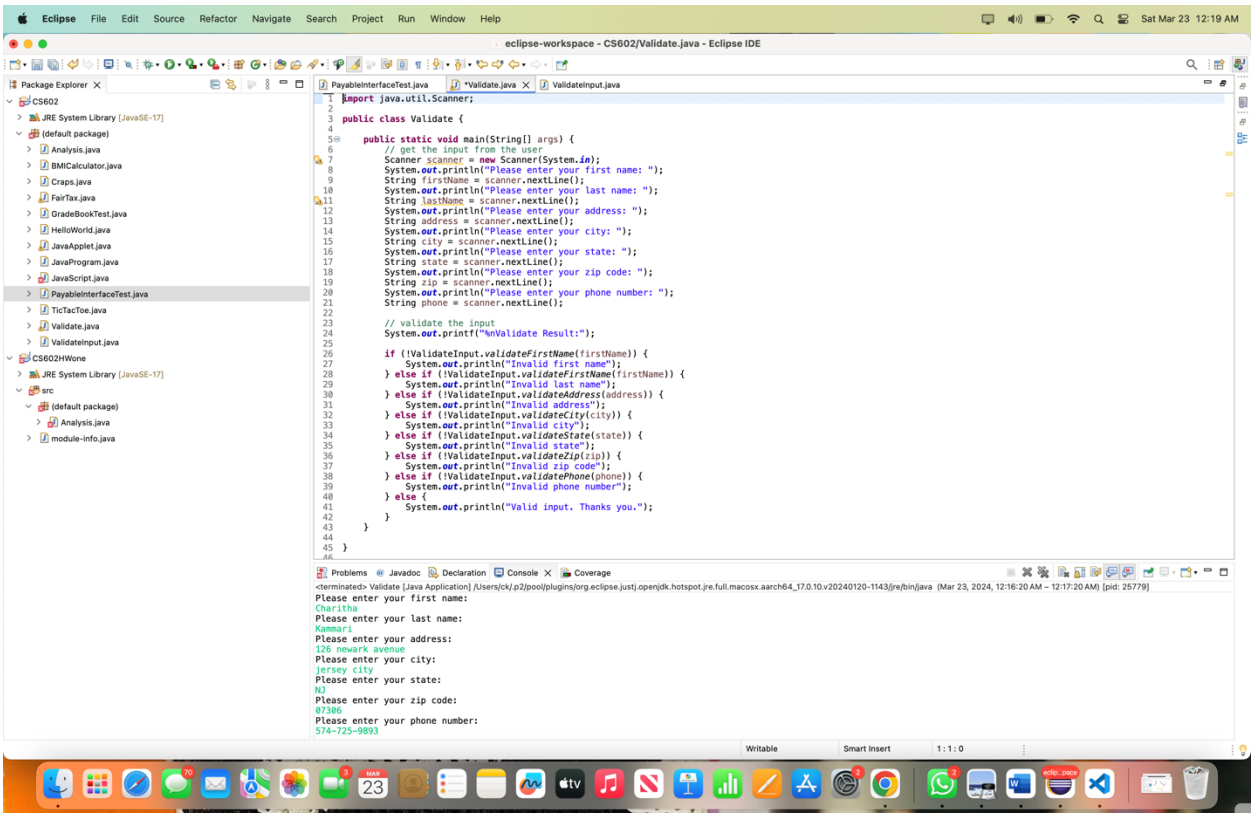
First and last names, address, zip code, city, state, and phone number must all be entered by the user individually.
Each input is saved in the relevant variables (firstName, lastName, address, city, state, zip, phone) using the scanner.nextLine() method.

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:



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]+).

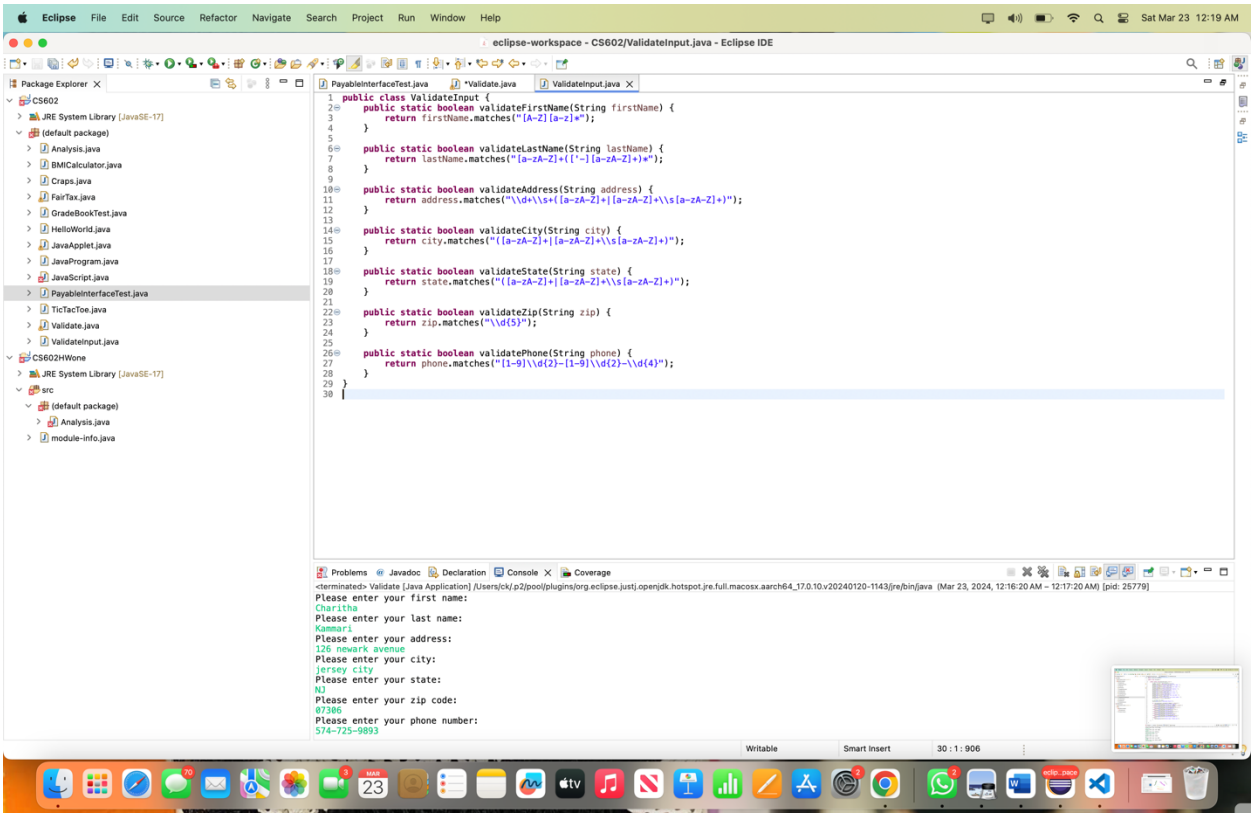
verifies the input for the state.

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

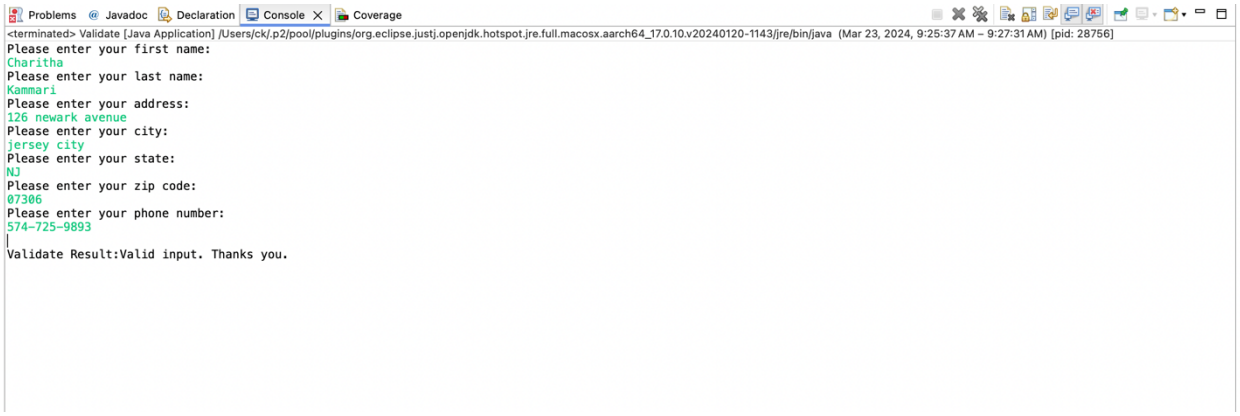
verifies the entered zip code.
ensures that the zip code has precisely five digits by using the regular expression pattern \\d{5}.

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:



Output:



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.

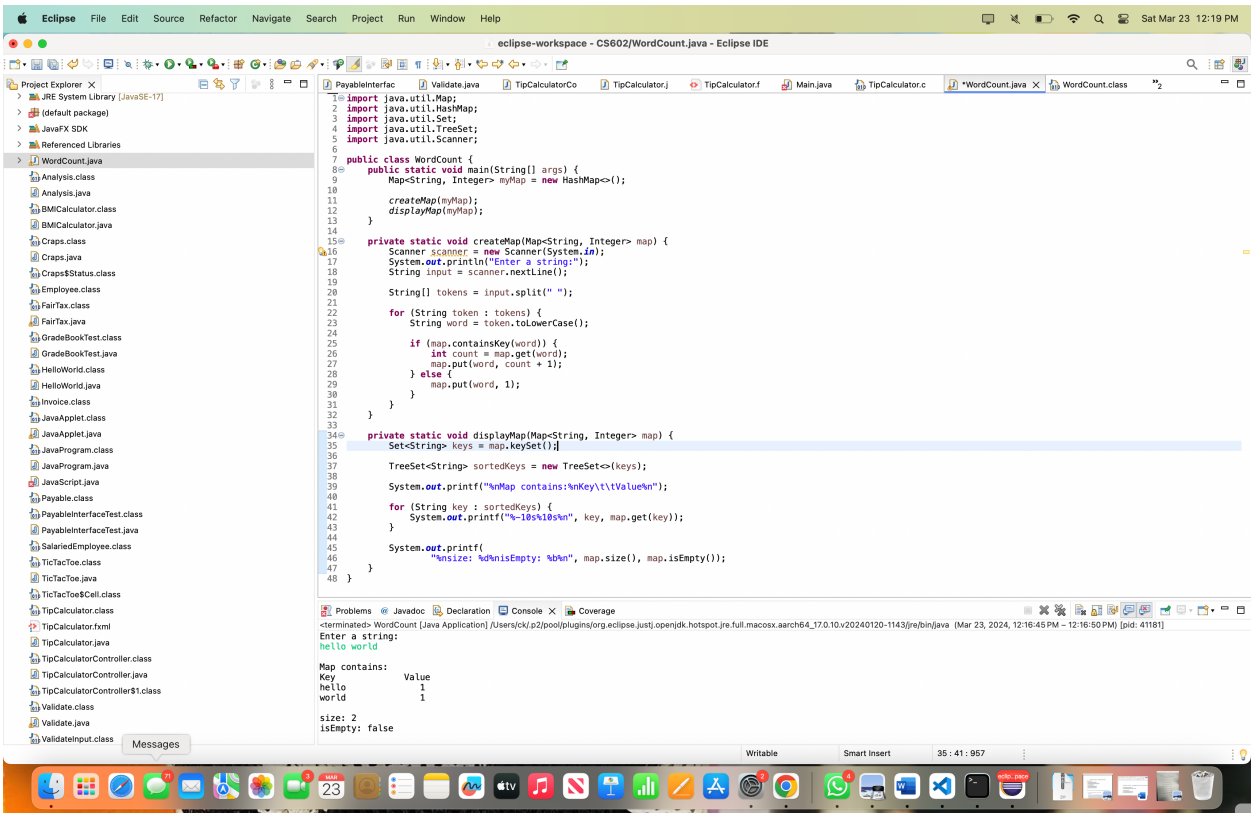
Then, it invokes two methods: displayMap(myMap) to output the contents of the map, and createMap(myMap) to add word counts to the map.

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 lowercases 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.

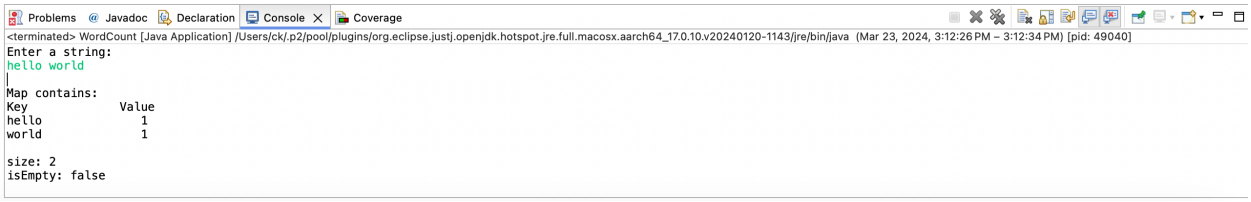
In this way, the contents of the map are displayed.
The keys are sorted using a TreeSet after being removed from the map.
After sorting the keys, iterating through them, it prints each key-value pair in an organized manner.

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

Code:


eclipse-workspace - CS602/WordCount.java - Eclipse IDE
Project Explorer
WordCount.java
Analysis.class
Analysis.java
BMICalculator.class
BMICalculator.java
Craps.class
Craps.java
Craps$Status.class
Employee.class
FairTax.class
FairTax.java
GradeBookTest.class
GradeBookTest.java
HelloWorld.class
HelloWorld.java
Invoice.class
JavaApplet.class
JavaApplet.java
JavaProgram.class
JavaProgram.java
JavaScript.java
Payable.class
PayableInterfaceTest.class
PayableInterfaceTest.java
SalariedEmployee.class
TicTacToe.class
TicTacToe.java
TicTacToe$Cell.class
TipCalculator.class
TipCalculator.fxml
TipCalculator.java
TipCalculatorController.class
TipCalculatorController.java
TipCalculatorController$1.class
Validate.class
Validate.java
ValidateInput.class
Messages
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.TreeSet;
import java.util.Scanner;
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:%nKey\t\tValue%n");
for (String key : sortedKeys) {
System.out.printf("%-10s%10s%n", key, map.get(key));
}
System.out.printf(
"%nsize: %d%nisEmpty: %b%n", map.size(), map.isEmpty());
}
}
Problems
Javadoc
Declaration
Console
Coverage
<terminated> WordCount [Java Application] /Users/ck/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (Mar 23, 2024, 12:16:45 PM – 12:16:50 PM) [pid: 41181]
Enter a string:
hello world
Map contains:
Key Value
hello 1
world 1
size: 2
isEmpty: false
Writable
Smart Insert
35 : 41 : 957

Output:


Problems
Javadoc
Declaration
Console
Coverage
<terminated> WordCount [Java Application] /Users/ck/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.10.v20240120-1143/jre/bin/java (Mar 23, 2024, 3:12:26 PM – 3:12:34 PM) [pid: 49040]
Enter a string:
hello world
Map contains:
Key Value
hello 1
world 1
size: 2
isEmpty: false