### Exercise 3.1. Carpet cost

* Assuming the carpet for the room described in the worked example costs £5.00 per square metre, amend the program to calculate and display the cost of the required amount of carpet.

Pseudocode:

Input Length as the next value inputted by user

Input Width as the next value inputted by user

Set Area to Length \* Width

Set Price to Area \* £5.00

Print Area and Price

import java.util.Scanner;

import java.util.InputMismatchException;

class carpetCost

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

//Input width and length, giving messages to let the user know what is being requested

System.out.println("Enter carpet length and width in meters.");

//define and set default doubles for input

Double length = 0.0, width = 0.0;

//try to catch InputMismatchException thrown if the input is not the correct type.

try

{

//write the start of lines to symbolise what each input is for.

System.out.print("Length: ");

length = input.nextDouble();

System.out.print("Width: ");

width = input.nextDouble();

}

catch (InputMismatchException e)

{

//tell the user their input was not decimal and exit with code 0 (normal)

System.out.println("You need to input a decimal number.");

System.exit(0);

}

//calculate area and cost of the carpet, where 5 is £5

Double area = length \* width;

Double cost = area \* 5;

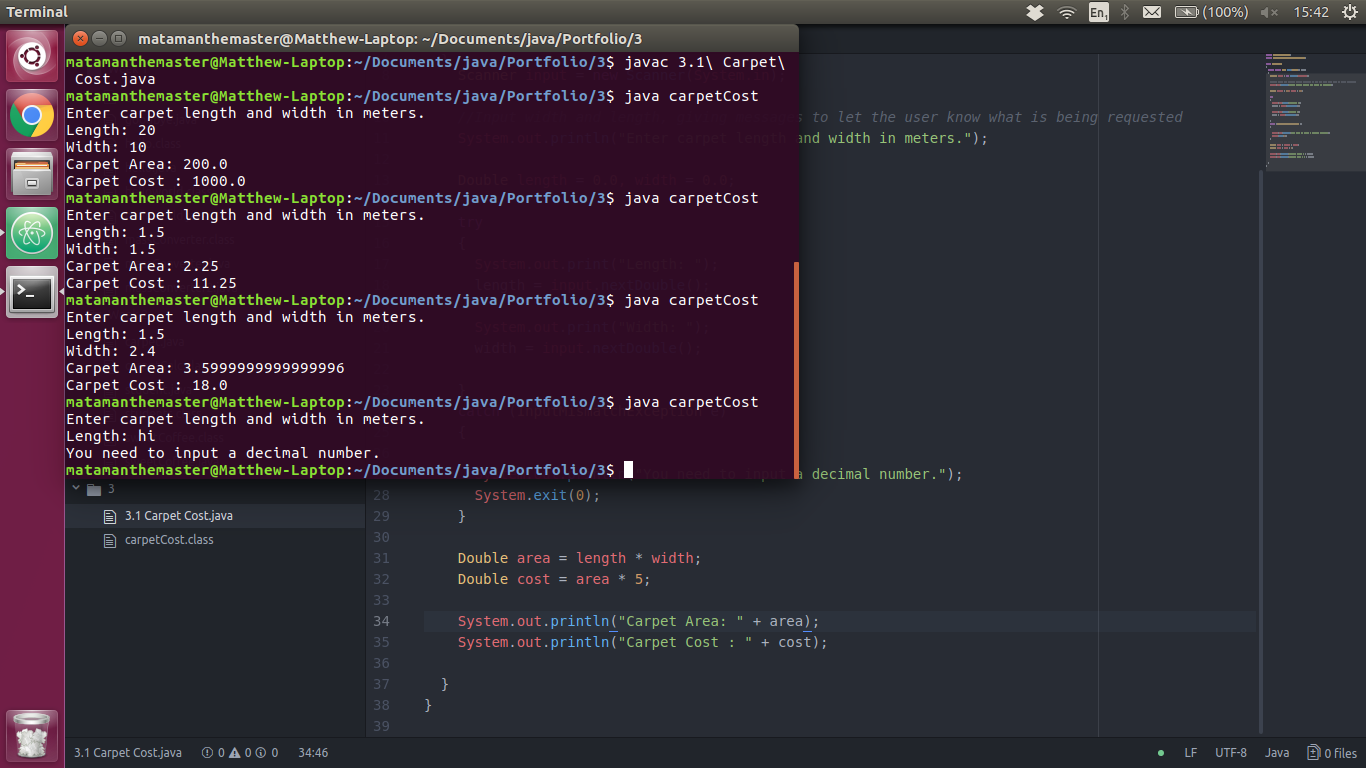
//output results.

System.out.println("Carpet Area: " + area);

System.out.println("Carpet Cost : " + cost);

}

}



### Exercise 3.2. Age in 2050

* Design and build a program to input the year in which you were born and display the age you will be in the year 2050.

Pseudocode:

Input Birth Year

Set Age to Birth Year - 2050

Print Age

import java.util.Scanner;

import java.util.InputMismatchException;

class agein2050

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

int birthYear = 0;

try

{

//try to input their birth year and catch errors

System.out.println("Enter your birth year.");

System.out.print("Year: ");

birthYear = input.nextInt();

}

catch (InputMismatchException e)

{

//Catch an InputMismatchException, they inputted the wrong type. Exit the program.

System.out.println("You need to enter a birth year as a number.");

System.exit(0);

}

if (birthYear < 2050 && birthYear > -1)

{

//get age in 2050 by subtraction.

int endAge = 2050 - birthYear;

//output.

System.out.println("In 2050 you will be " + endAge);

}

else

{

//Tell the user their input was invalid, exit program.

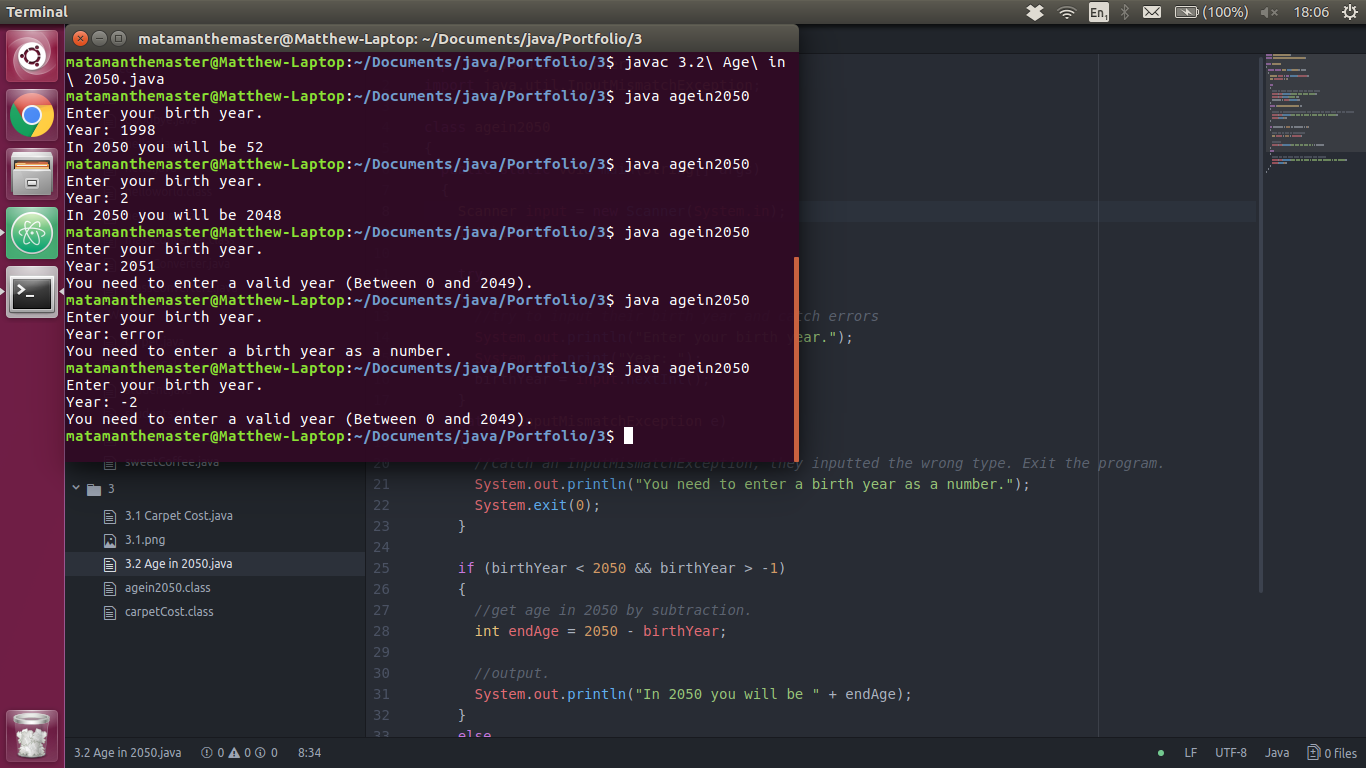
System.out.println("You need to enter a valid year (Between 0 and 2049).");

System.exit(0);

}

}

}



### Exercise 3.3. Average

* Design and build a program to input a set of 4 numbers and display their average (be careful with data types!)

Pseudocode:

Set i to 0

for i to 3

Set Average = Average + the next value inputted by the user

end

Set Average = Average / 4

Print Average

import java.util.Scanner;

import java.util.InputMismatchException;

class average

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

Double average = 0.0;

System.out.println("Please input the 4 numbers to mean.");

for (int i = 0; i < 4; i++)

{

try

{

System.out.print(i + 1 + ": ");

//increment by the input.

average += input.nextDouble();

}

catch (InputMismatchException e)

{

//catch the user supplying invalid type.

System.out.println("Value inputed was not a number.");

System.exit(0);

}

}

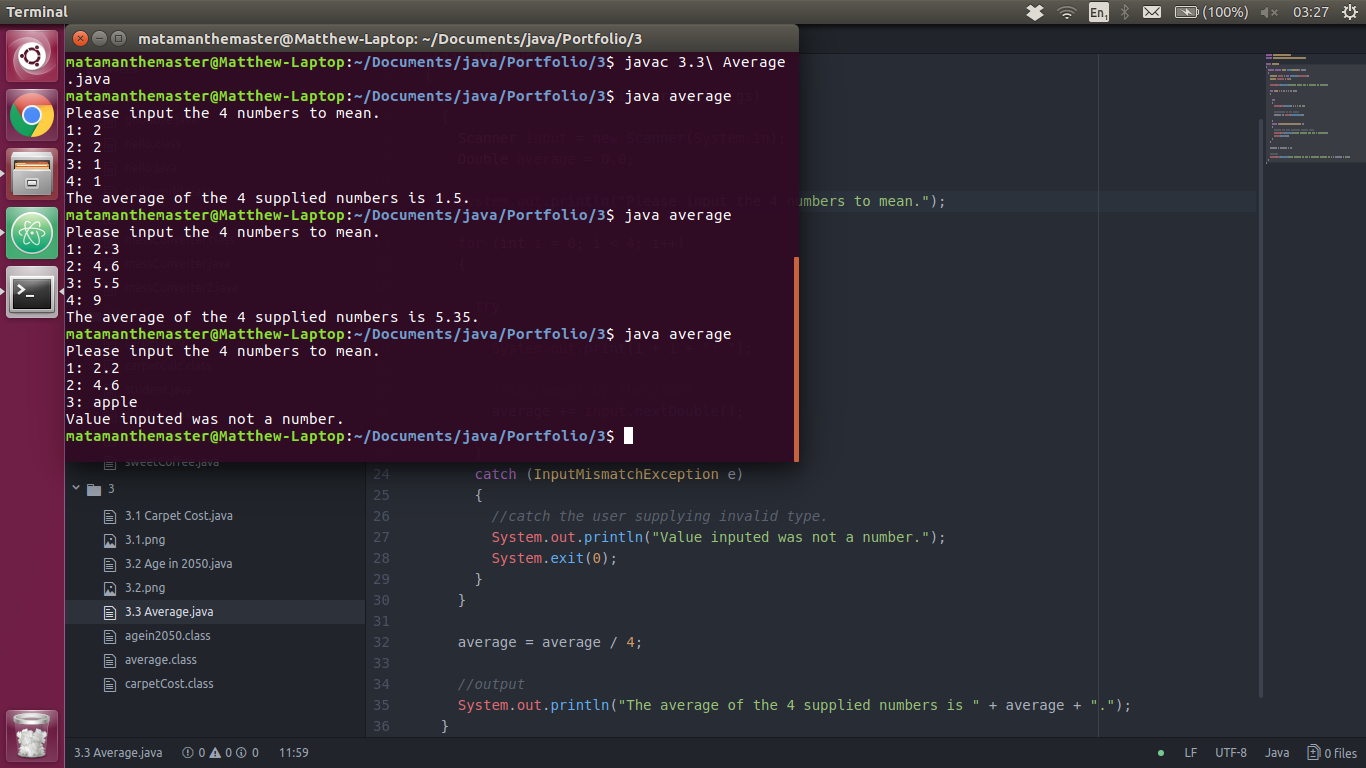
average = average / 4;

//output

System.out.println("The average of the 4 supplied numbers is " + average + ".");

}

}



### Exercise 3.4. Money

* Design and build a program to input the number of £5, £10 and £20 notes you have, then display the total number of notes and the total amount of money.

Pseudocode:

Set noFives to the next intager inputted by the user

Set noTens to the next intager inputted by the user

Set noTwenties to the next intager inputted by the user

Set totalNotes to noFives + noTens + noTwenties

Set totalMoney to noFives \* 5

Set totalMoney to totalMoney + noTens \* 10

Set totalMoney to totalMoney + noTwenties \* 20

Print totalNotes and totalMoney

import java.util.Scanner;

import java.util.InputMismatchException;

class money

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

System.out.println("Input the number of notes you have");

int noFives = 0;

int noTens = 0;

int noTwenties = 0;

try

{

//input the quantities of each notes the client has.

System.out.print("£5: ");

noFives = input.nextInt();

System.out.print("£10: ");

noTens = input.nextInt();

System.out.print("£20: ");

noTwenties = input.nextInt();

}

catch (InputMismatchException e)

{

//Invalid inputs.

System.out.println("Only whole numbers can be used to represent the quantity of each note.");

System.exit(0);

}

//validation

if (noFives < 0 || noTens < 0 || noTwenties < 0)

{

System.out.println("You cannot have negative quantities of notes.");

System.exit(0);

}

else

{

//calculate end numbers for display.

int totalNotes = noFives + noTens + noTwenties;

int totalValue = (noFives \* 5) + (noTens \* 10) + (noTwenties \* 20);

System.out.println("For the " + totalNotes + " notes, you have a total of £" + totalValue + ".");

}

}

}

### Exercise 4.1. Capital city

* Design, write and test a program which asks the user to input the capital city of France. If they answer **Paris**, give them a congratulations message.

Pseudocode:

Set Answer to the value provided by the user

if Answer when floored = "paris"

Print "Congradulations"

else

Print "Incorrect"

end

import java.util.Scanner;

class capitalCity

{

public static void main(String[] args)

{

//Scanner for console input

Scanner input = new Scanner(System.in);

//ask the question, preformat.

System.out.println("What is the capital city of France?");

System.out.print("> ");

//set answer to the user input. Don't care about checking for an invalid value as strings can be any input.

String answer = input.next();

//changes to lower case

if (answer.toLowerCase().equals("paris"))

{

System.out.println("congratulations");

}

else

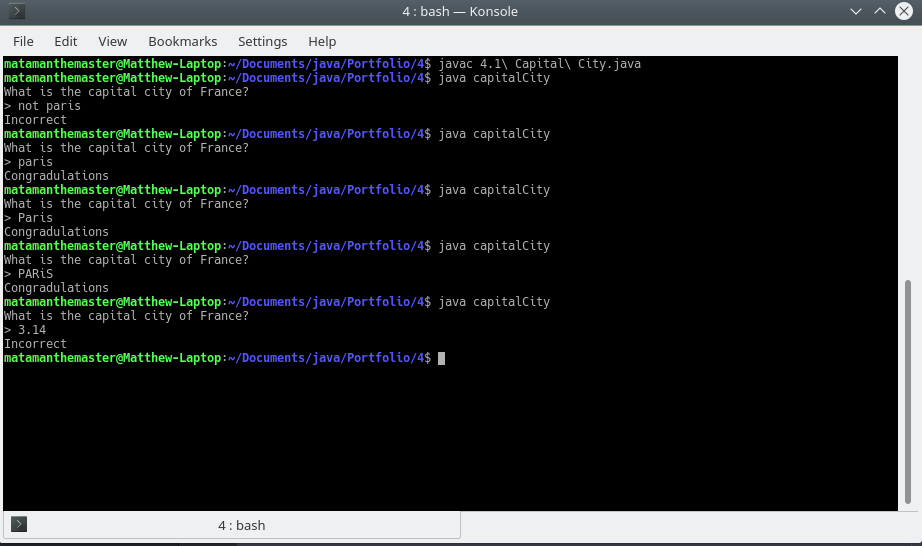
{

System.out.println("Incorrect");

}

}

}



### Exercise 4.2. Income and expenditure

* Design, write and test a program which asks the user their income and expenditure, and output 'SPEND SPEND SPEND!", or "Put some money in the bank!" depending on the outcome.

Pseudocode:

Set Threshold to value entered by client

Set Income to value entered by client

Set Expenditure to value entered by client

if Income >= 0 and Expenditure >= 0

Set Total = income - Expenditure

if Total < Threshold

Print "Put some money in the bank!"

else

Print "SPEND SPEND SPEND!"

end

end

import java.util.Scanner;

import java.util.InputMismatchException;

class incomeExpenditure

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

//vars to be inputted by scanner and for processing.

Double threshold = 0.0;

Double income = 0.0;

Double expenditure = 0.0;

Double total = 0.0;

try

{

//input vars

System.out.print("Threshold: ");

threshold = input.nextDouble();

System.out.print("Income: ");

income = input.nextDouble();

System.out.print("Expenditure: ");

expenditure = input.nextDouble();

}

catch (InputMismatchException e)

{

//if invalid type then exit program.

System.out.println("You need to input a number (whole or decimal).");

System.exit(0);

}

//check all variables inputted are valid (>= 0)

if (income >= 0 && expenditure >= 0)

{

//calculate total money left.

total = income - expenditure;

if (total < threshold)

{

//outcome

System.out.println();//blank line to seperate inputs and outputs

System.out.println("Put some money in the bank!");

}

else

{

//outcome

System.out.println();//blank line to seperate inputs and outputs

System.out.println("SPEND SPEND SPEND!");

}

}

else

{

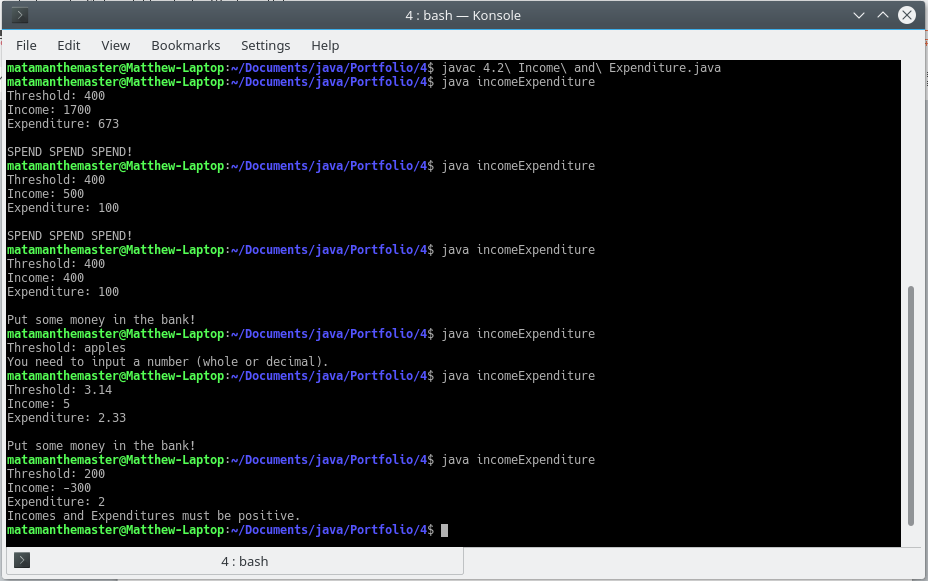
//if income and expenditure is < 0

System.out.println("Incomes and Expenditures must be positive.");

}

}

}



### Exercise 4.3. Meal cost

* Design, write and test a program which asks a waiter the cost of a customer's meal. If the cost is 10 pounds or more, give a 5% discount.

Pseudocode

Input MealCost

if MealCost >= 10

FivePercent = (MealCost / 100) \* 5

MealCost = MealCost - FivePercent

end

Print MealCost

Code

import java.util.Scanner;

import java.util.InputMismatchException;

class mealCost

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

Double mealCost = 0.0;

Double fivePercent = 0.0;

try

{

System.out.println("Input a meal price.");

System.out.print("£");

mealCost = input.nextDouble();

}

catch (InputMismatchException e)

{

System.out.println("You need to input a whole or decimal number.");

System.exit(0);

}

//if positive

if (mealCost >= 0)

{

if (mealCost >= 10)

{

//take 5% of the meal cost away

fivePercent = (mealCost / 100) \* 5;

mealCost -= fivePercent;

}

//outcome

System.out.println("The meal cost £" + mealCost);

}

else

{

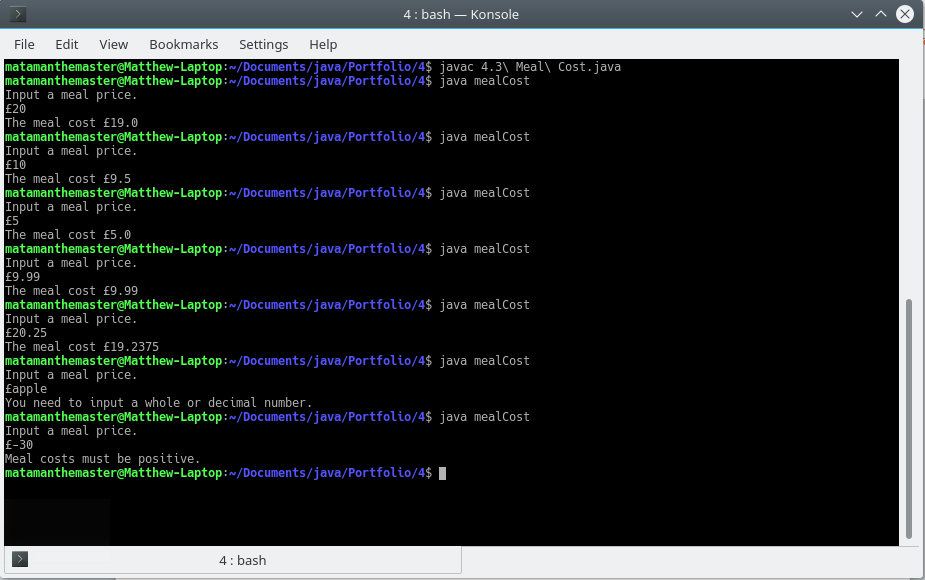
//if mealCost is negative

System.out.println("Meal costs must be positive.");

}

}

}



### Exercise 4.4. Logic teaser

* Below is a table. Columns 1 and 2 indicate two tests, A and B. As we know, tests always evaluate to true (T) and false (F). The rows of the table indicate combinations of the possible values for A and B. The other columns in the table show more complex tests which use A and B in various combinations along with conditional and logical operators. Fill out the rest of the table for each row, given the True/False values of A and B. To start you off, the first complex condition (A or B) has been completed.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **A||B** | **A&&B** | **!(A||B)** | **!(A&&B)** | **!A||B** | **!(!A&&!B)** | **A||!B** |
| **T** | **T** | T | T | F | F | T | T | T |
| **T** | **F** | T | F | F | T | F | T | T |
| **F** | **T** | T | F | F | T | T | T | F |
| **F** | **F** | F | F | F | T | T | F | T |

### Exercise 4.5. Payday

* Design, code and test a program which inputs a code for the name of a day ("Sun" for Sunday, "Wk" for other Day) together with 2 numbers representing normal hours worked and overtime hours worked. Output the day name and the pay according to the following rules:

Sunday: ordinary hours @ £15 per hour; overtime @ £20 per hour   
Other day: ordinary hours @ £8 per hour; overtime @ £12.00 per hour.

Pseudocode

Set DayCode to the lower case value inputted by user

Set NormalHours to the value inputted by the user

Set OvertimeHours to the value inputted by the user

if DayCode = "sun"

Set NormalRate to 15

Set OvertimeRate to 20

Set Day to "Sunday"

else if DayCode = "wk"

Set NormalRate to 8

Set OvertimeRate to 12

Set Day to "Other Day"

end

Set NormalPay = NormalRate \* NormalHours

Set OvertimePay = OvertimeRate \* OvertimeHours

Print Day

Print NormalPay

Print OvertimePay

Code:

import java.util.Scanner;

import java.util.InputMismatchException;

class payday

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

//inputs

String dayCode = "";

Double hoursNormal = 0.0;

Double hoursOvertime = 0.0;

//Processing & outputs

int rateNormal = 0;

int rateOvertime = 0;

String day = "";

Double payNormal = 0.0;

Double payOvertime = 0.0;

System.out.println("Input a day code (wk for other day, sun for sunday).");

try

{

//convert the next string to lower case

dayCode = input.next().toLowerCase();

System.out.print("Normal Hours: ");

hoursNormal = input.nextDouble();

System.out.print("Overtime Hours: ");

hoursOvertime = input.nextDouble();

}

catch (InputMismatchException e)

{

//InputMismatchException cannot happen on string, so user must have not entered hours correctly

System.out.println("Normal hours and overtime hours must be numbers.");

System.exit(0);

}

//if hours are >= 0 and <= 168 (number of hours in a week) and so is valid

if (hoursNormal > -1 && hoursOvertime > -1 && hoursNormal < 169 && hoursOvertime < 169)

{

if (dayCode.equals("sun"))

{

day = "sunday";

//sunday pay rates

rateNormal = 15;

rateOvertime = 20;

}

else if (dayCode.equals("wk"))

{

day = "other day";

//other day pay rates

rateNormal = 8;

rateOvertime = 12;

}

else

{

//invalid day code (not wk or sun)

System.out.println("You provided an invalid day code.");

System.exit(0);

}

//calculates the pay for the work hours done (using rates for respective day)

payNormal = rateNormal \* hoursNormal;

payOvertime = rateOvertime \* hoursOvertime;

//print Day, Normal Pay, Overtime Pay, Total Pay

System.out.println("Day: " + day + ", Normal Pay: " + payNormal + ", Overtime Pay: " + payOvertime + ", Total Pay: " + (payNormal + payOvertime));

}

else

{

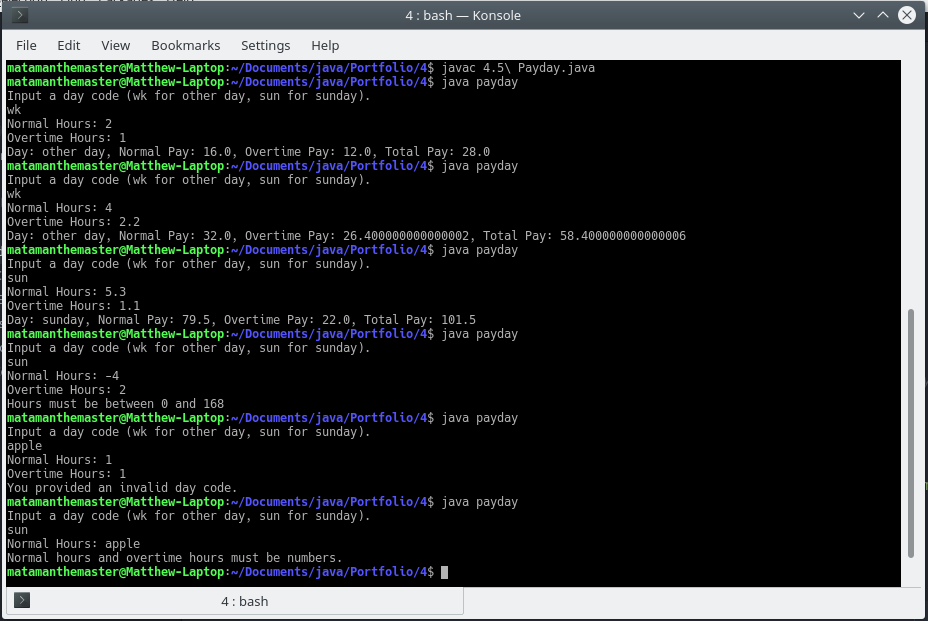
//if the hours entered are < 0 || > 168

System.out.println("Hours must be between 0 and 168");

}

}

}



### Exercise 4.6. Insurance

* *Bike Direct* insurance offers bike insurance as follows:   
    
  basic rate = £30  
  add £10 for a mountain bike  
  add £5 for cyclists under 25.   
    
  Design, write and test a program to input the type of bike - mountain or touring, and the cyclist's age, and calculate and display the premium payable.

Pseudocode:

Set BikeType to the floored value inputted by the user

Set Age to the value inputted by the user

Set InsuranceTotal = 30

if BikeType == "mountain"

InsuranceTotal += 10

end

if Age < 25

InsuranceTotal +=5

end

Print InsuranceTotal

import java.util.Scanner;

import java.util.InputMismatchException;

class insurance

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

//define input variables with default (null) values

String type = null;

int age = -1;

try

{

//input variables

System.out.print("Bike Type: ");

type = input.next().toLowerCase();/\*converts the next string entered into the cmd tolower case.\*/

System.out.print("Your Age: ");

age = input.nextInt();

}

catch (InputMismatchException e)

{

//if the type (for input.nextInt()) is not correct, stop the program.

System.out.println("Age must be a whole number");

System.exit(0);

}

//validate age

if (age > -1 && age < 151)

{

//var stores the final result

int totalInsurance = 30;

if (type.equals("mountain"))

{

totalInsurance += 10;

}

if (age < 25)

{

totalInsurance += 5;

}

//output result and answers provided by the client.

System.out.println("For a " + age + " year old with a " + type + " bike, it will cost you " + totalInsurance + " for

insurance.");

}

else

{

//else, if age is invalid, stop program.

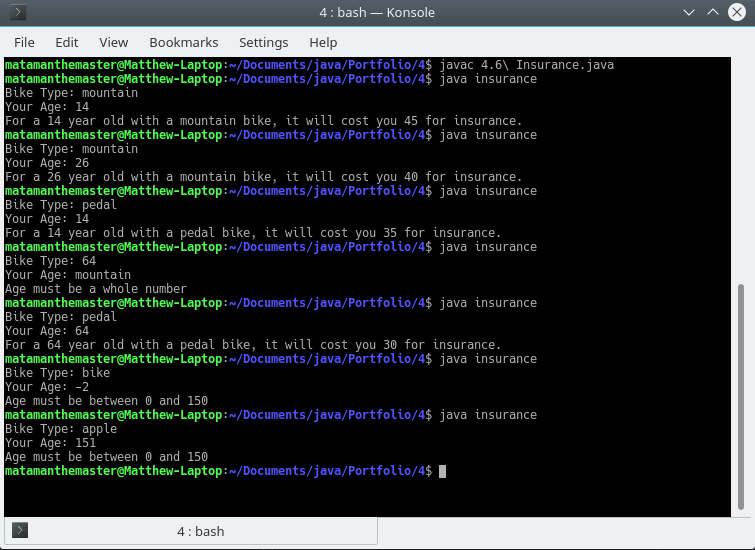
System.out.println("Age must be between 0 and 150");

System.exit(0);

}

}

}



### Exercise 5.1. More coffee

* In the coffee survey program, what would happen if **nothing**, **sugar** and **sweetener** were not originally set to zero? Would the program work?

No, the compiler would think that the variable was used before a value was set, as we set the value during a for block and during a try catch block.

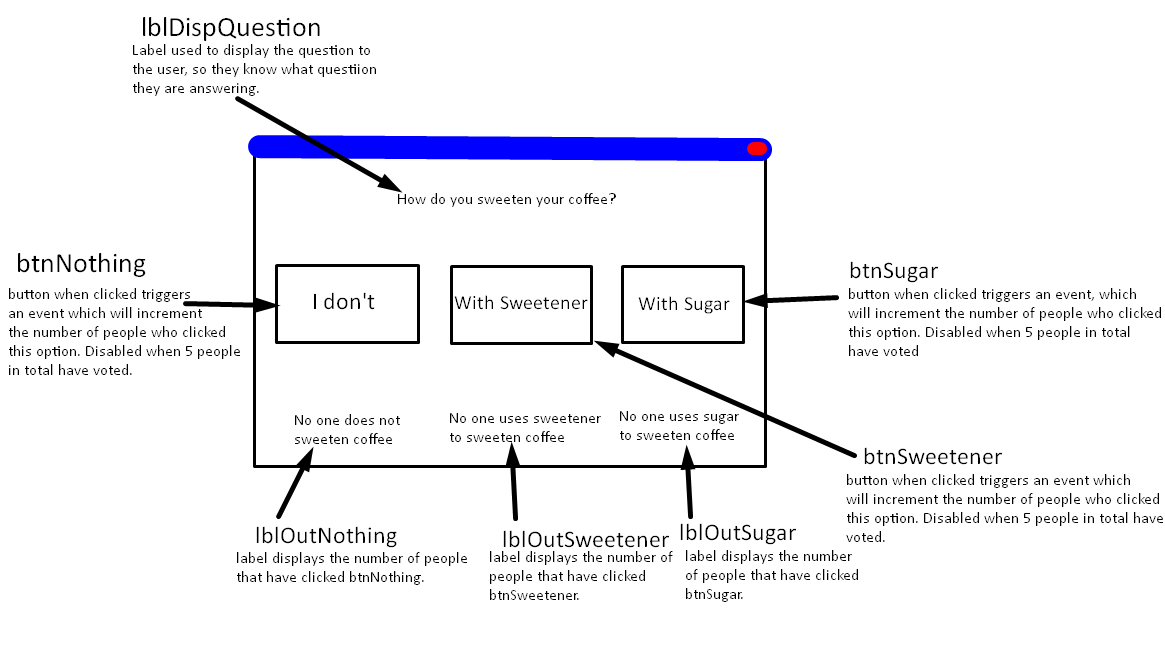
* Type in the coffee survey program and get it working. Test to see what happens when only one person likes a particular way of drinking coffee? Modify your program to report in correct English by stating "Only 1 person ... " rather than "1 people ...".
* Modify the coffee survey program to validate the three choices. Trap for out of range values (i.e. where the user types in a number other than 1, 2, or 3). **Make sure your program still processes 5 valid selections.**

For 5.1 I started to program GUI applications using the JavaFX library, with FXML as the template generated using a program, Gluon Scene Builder. FXML controls point to a control class and controls invoke methods on particular events occurring, all controlled by the library.

I changed how the application would run to better suit the event driven nature of JavaFX. I opted to use buttons that trigger events, calling my own methods for the user to choose their answer. For this, no loops are required for asking the user the question or getting their answer; the question can always be displayed, and the answers can always be displayed at the same time via buttons.

Design

GUI Design:



Pseudocode:

Define btnNothing, btnSugar, btnSweetener as buttons

Define lblOutNothing, lblOutSugar, lblOutSweetener, lblOutNoVotes as labels

Define noNone, noSugar, noSweetener, totalVotes as 0

On event btnNothing pressed

If noNone == 1

Set lblOutNothing’s text to “1 person does not sweeten coffee.”

Else if noNone >= 1

Set lblOutNothing’s text to noNone + “ people do not sweeten coffee.”

End

Set totalVotes as noNone + noSugar + noSweetener

If totalVotes > 4

Disable btnNothing

Disable btnSugar

Disable btnSweetener

End

End

On event btnSweetener pressed

If noSweetener == 1

Set lblOutSweetener’s text to “1 person uses sweetener.”

Else if noSweetener >= 1

Set lblOutSweetener’s text to noSweetener + “ people use sweetener.”

End

Set totalVotes as noNone + noSugar + noSweetener

If totalVotes > 4

Disable btnNothing

Disable btnSugar

Disable btnSweetener

End

End

On event btnSugar pressed

If noSugar == 1

Set lblOutSugar’s text to “1 person uses sugar.”

Else if noSugar >=

Set lblOutSugar’s text to noSugar + “ people use sugar.”

End

Set totalVotes as noNone + noSugar + noSweetener

If totalVotes > 4

Disable btnNothing

Disable btnSugar

Disable btnSweetener

End

End

Code

Main Class:

package sweetCoffee;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class Main extends Application {  
  
 @Override  
 public void start(Stage stageMain) throws Exception {  
  
 //get the contents of Main.fxml to append to the scene  
 Parent root = FXMLLoader.*load*(getClass().getResource("Main.fxml"));  
  
 //set the scene with the contents of Main.fxml and default size.  
 Scene sceneMain = new Scene(root, 600, 275);  
  
 //get the css file and add it to the scene's stylesheets.  
 sceneMain.getStylesheets().add(getClass().getResource("css/style.css").toExternalForm());  
  
 //set title, scene and show the stage.  
 stageMain.setTitle("5.1 Sweet Coffee");  
 stageMain.setScene(sceneMain);  
 stageMain.show();  
 }  
  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

Some code here was generated by the IDE, mainly the package and class (which I modified the package name of) and the two methods. I wrote the code inside the start method.

Controller Class:

package sweetCoffee;  
  
import javafx.fxml.FXML;  
import javafx.scene.text.Text;  
import javafx.scene.control.Button;  
  
public class Controller {  
  
 //gets instances of elements from the FXML via their fx:id.  
 @FXML private Button btnNothing;  
 @FXML private Button btnSugar;  
 @FXML private Button btnSweetener;  
 @FXML private Text lblOutNothing;  
 @FXML private Text lblOutSugar;  
 @FXML private Text lblOutSweetener;  
 @FXML private Text lblOutNoVotes;  
  
 //incremental ints.  
 private int noNone = 0;  
 private int noSugar = 0;  
 private int noSweetener = 0;  
 private int totalVotes = 0;  
  
 //called when btnNothing is pressed  
 @FXML  
 protected void btnNothingPressed()  
 {  
 noNone++;  
  
 //set the number who voted none to the none display, grammatically formatting it.  
 if (noNone == 1)  
 {  
 lblOutNothing.setText("1 person does not sweeten coffee.");  
 }  
 else if (noNone >= 1)  
 {  
 lblOutNothing.setText(noNone + " people do not sweeten coffee.");  
 }  
  
 /\*runs method that should run each time a button is pressed.  
 Provides utilities such as checking visibility, counting total votes and disabling at 5 votes.\*/  
 buttonPressed();  
 }  
  
 //called when btnSugar is pressed.  
 @FXML  
 protected void btnSugarPressed()  
 {  
 noSugar++;  
  
 //set the number who voted sugar to the none display, grammatically formatting it.  
 if (noSugar == 1)  
 {  
 lblOutSugar.setText("1 person uses sugar.");  
 }  
 else if (noSugar >= 1)  
 {  
 lblOutSugar.setText(noSugar + " people use sugar.");  
 }  
  
 buttonPressed();  
 }  
  
 //called when btnSweetener is pressed.  
 @FXML  
 protected void btnSweetenerPressed()  
 {  
  
 noSweetener++;  
  
 //set the number who voted sweetener to the none display, grammatically formatting it.  
 if (noSweetener == 1)  
 {  
 lblOutSweetener.setText("1 person uses sweetener.");  
 }  
 else if (noSweetener >= 1)  
 {  
 lblOutSweetener.setText(noSweetener + " people use sweetener.");  
 }  
  
 buttonPressed();  
 }  
  
 private void buttonPressed()  
 {  
 //makes sure that the outputs are visible.  
 if (!lblOutNothing.isVisible())  
 {  
 lblOutNothing.setVisible(true);  
 lblOutSugar.setVisible(true);  
 lblOutSweetener.setVisible(true);  
 }  
  
 //calculates the new total of votes.  
 totalVotes = noNone + noSugar + noSweetener;  
  
 //displays the total votes on lblNoVotes with correct grammar.  
 if (totalVotes == 1)  
 {  
 lblOutNoVotes.setText("1 person has voted.");  
 }  
 else if (totalVotes > 1)  
 {  
 lblOutNoVotes.setText(totalVotes + " people have voted.");  
 }  
  
 //ends the voting by disabling buttons when we reach the maximum votes specified in the question. This prevents additional inputs.  
 if (totalVotes > 4)  
 {  
 btnNothing.setDisable(true);  
 btnSugar.setDisable(true);  
 btnSweetener.setDisable(true);  
 }  
 }  
  
}

The package and the class were auto-generated by the IDE, everything else I wrote.

Scene Builder generated FXML:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import java.lang.String?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.Pane?>  
<?import javafx.scene.layout.RowConstraints?>  
<?import javafx.scene.text.Text?>  
  
<GridPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="400.0" minWidth="600.0" prefHeight="400.0" prefWidth="700.0" xmlns="http://javafx.com/javafx/9" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sweetCoffee.Controller">  
 <columnConstraints>  
 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />  
 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />  
 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />  
 </columnConstraints>   
 <rowConstraints>  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 </rowConstraints>  
 <children>  
 <Pane prefHeight="200.0" prefWidth="200.0" styleClass="appHead" GridPane.columnSpan="2147483647" />  
 <Text fx:id="lblDispQuestion" strokeType="OUTSIDE" strokeWidth="0.0" text="How do you sweeten your coffee?" textAlignment="CENTER" textOrigin="CENTER" GridPane.columnIndex="1" GridPane.halignment="CENTER">  
 <styleClass>  
 <String fx:value="text" />  
 <String fx:value="question" />  
 </styleClass>  
 </Text>  
 <Button fx:id="btnNothing" mnemonicParsing="false" onAction="#btnNothingPressed" styleClass="btn" text="I don't" GridPane.halignment="CENTER" GridPane.rowIndex="1" />  
 <Button fx:id="btnSugar" mnemonicParsing="false" onAction="#btnSugarPressed" styleClass="btn" text="With Sugar" GridPane.columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="1" />  
 <Button fx:id="btnSweetener" mnemonicParsing="false" onAction="#btnSweetenerPressed" styleClass="btn" text="With Sweetener" GridPane.columnIndex="2" GridPane.halignment="CENTER" GridPane.rowIndex="1" />  
 <Text fx:id="lblOutNoVotes" strokeType="OUTSIDE" strokeWidth="0.0" styleClass="text" text="No one has voted yet." GridPane.columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="2" />  
 <Text fx:id="lblOutNothing" strokeType="OUTSIDE" strokeWidth="0.0" styleClass="text" text="No one doesn't sweeten coffee." textAlignment="CENTER" visible="false" wrappingWidth="180.0" GridPane.halignment="CENTER" GridPane.rowIndex="3" />  
 <Text fx:id="lblOutSugar" strokeType="OUTSIDE" strokeWidth="0.0" styleClass="text" text="No one uses sugar. " textAlignment="CENTER" visible="false" wrappingWidth="180.0" GridPane.columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="3" />  
 <Text fx:id="lblOutSweetener" strokeType="OUTSIDE" strokeWidth="0.0" styleClass="text" text="No one uses sweetener." textAlignment="CENTER" visible="false" wrappingWidth="180.0" GridPane.columnIndex="2" GridPane.halignment="CENTER" GridPane.rowIndex="3" />  
 </children>  
</GridPane>

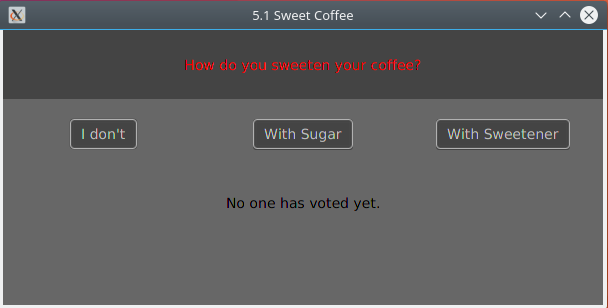
All was generated using Gluon Scene Builder, which I designed in its GUI.

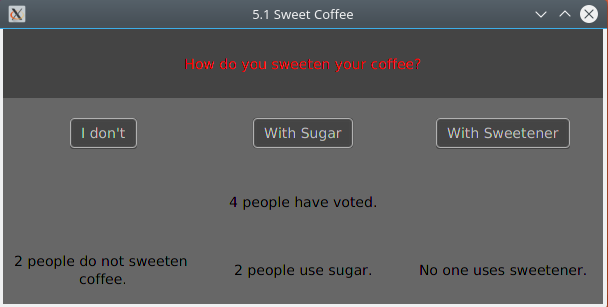
JavaFX CSS:

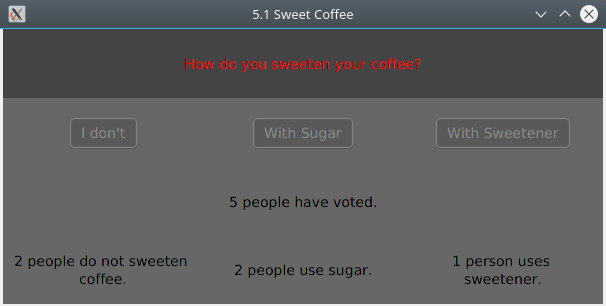
GridPane {  
 -fx-background-color: #676767;  
}  
  
.text {  
 -fx-wrap-text: true;  
 -fx-font-size: 14px;  
 -fx-text-fill: #cccccc;  
}  
  
.btn {  
 -fx-background-color: #444444;  
 -fx-text-fill: #bbbbbb;  
 -fx-font-size: 14px;  
 -fx-border-color: #bbbbbb;  
 -fx-border-radius: 4px;  
 -fx-width: 150px;  
}  
  
.btn:hover {  
 -fx-background-color: #555555;  
 -fx-fill: #ff0000;  
 -fx-text-fill: #ff0000 **!important**;  
}  
  
.appHead {  
 -fx-width: 100%;  
 -fx-background-color: #444444;  
}  
  
.question {  
 -fx-fill: #ff0000;  
}

All was written by me.

Output







### Exercise 5.2. Square and cube

* Design, write, and test a program to input 7 integers and, for each integer, calculate and display its square and cube.

For this exercise I made both a CLI and GUI application because I was unsure of how I would keep to the specification of the question. The CLI was supposed to keep to the specification whereas the GUI was supposed to make a more usable and intuitive application.

**CLI**

Design

Pseudocode

Define ans[]

Set i to 0

For i to 6

Set exp to user inputted number

Set base to user inputted number

Set result to power(base, exp)

Set ans[i] to result

End

Set i to 0

For i to 6

Output ans[i]

End

Code

import java.util.Scanner;

import java.util.InputMismatchException;

class cliSquareAndCube

{

public static void main(String[] args)

{

//variable for gettint values from the console

Scanner input = new Scanner(System.in);

//array to hold the results.

double[] results = new double[7];

//variables to be inputted by the user.

double base = 0;

double exp = 0;

//loop 1 to calculate exponential functions from user data.

for (int i = 0; i < 7; i++)

{

try

{

//input the number base.

System.out.print("Exponential Function " + i + " Base: ");

base = input.nextDouble();

//input the exponent.

System.out.print("Exponential Function " + i + " Exponent: ");

exp = input.nextDouble();

}

catch (InputMismatchException e)

{

//when the user enters an incorrect type (not an int), and Scanner throws the InputMismatchException

//exit and tell the user why.

System.out.println("Bases and Exponents should only be numbers");

System.exit(0);

}

//after the try catch for this loop, calculate the exponent based on the user input.

results[i] = Math.pow(base, exp);

}

//create whitespace and heading in terminal

System.out.println();

System.out.println();

System.out.println();

System.out.println();

System.out.println();

System.out.println("Results:");

//loop 2 to display the results of the exponential functions

for (int i = 0; i < 7; i++)

{

System.out.println(i + " " + results[i]);

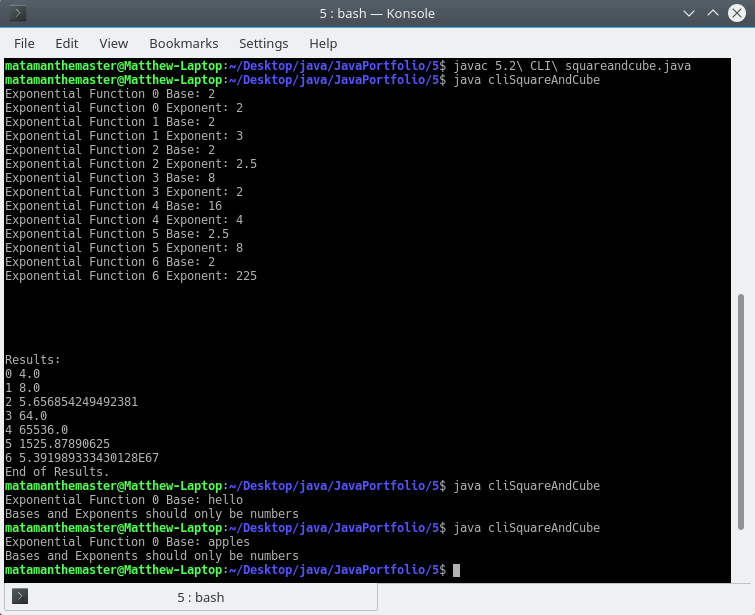
}

System.out.println("End of Results.");

}

}

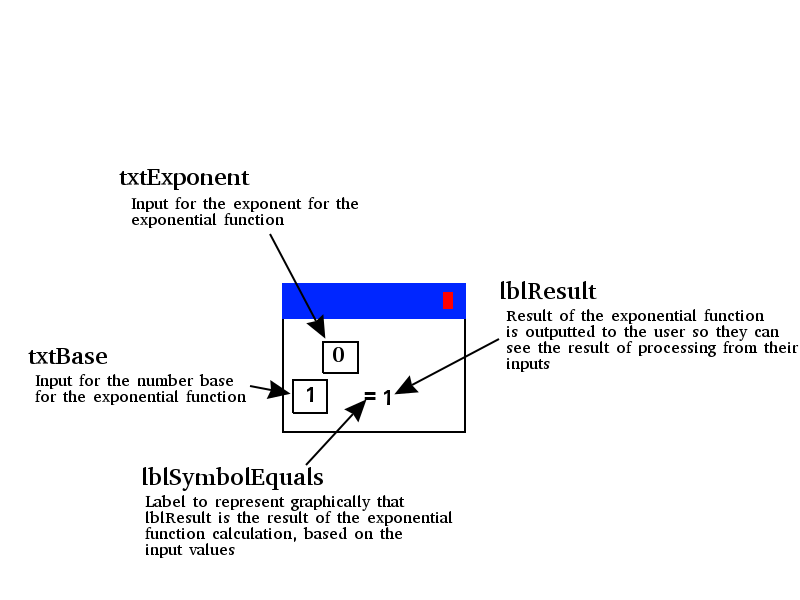
Output



**GUI**

Design

GUI Design



Pseudocode

On event txtBase’s text being changed

Set base to txtBase converted to type intager

Set exponent to txtExponent converted to type intager

Set txtResult to base to the power of exp

End

On event txtExponent’s text being changed

Set base to txtBase converted to type intager

Set exponent to txtExponent converted to type intager

Set txtResult to base to the power of exp

End

Code

Main Class

package squareAndCube;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class Main extends Application {  
  
 @Override  
 public void start(Stage stageMain) throws Exception  
 {  
  
 //get the contents of Main.fxml to append to the scene  
 Parent root = FXMLLoader.*load*(getClass().getResource("Main.fxml"));  
  
 //set the scene with the contents of Main.fxml and default size.  
 Scene sceneMain = new Scene(root, 250, 100);  
  
  
 //get the css file and add it to the scene's stylesheets.  
 sceneMain.getStylesheets().add(getClass().getResource("css/style.css").toExternalForm());  
  
 //set title, scene and show the stage.  
 stageMain.setTitle("5.2 Square and Cube");  
 stageMain.setScene(sceneMain);  
 stageMain.show();  
  
  
 /\*IF to check OS names, as setResizable does not work with some operating systems, such as linux.  
 By default we want to do it the longer method with more set actions, and looks worse on windows. If we can just use set  
 resizable then we can do that. Linux based OS' do not work with setResizable, which is an issue because  
 that is the kernel I am using to program javafx\*/  
 String osName = System.*getProperty*("os.name");  
  
 if (osName.indexOf("Windows") != -1)  
 {  
 //any os that contains "Windows" within its name (e.g. "Windows 10"  
 stageMain.setResizable(false);  
 }  
 else  
 {  
 /\*sets resize properties. Sets current height and width at the top, then sets the max and mins  
 dependent on their current value (what was set at the top)\*/  
 stageMain.setHeight(100);  
 stageMain.setWidth(250);  
 stageMain.setMaxWidth(stageMain.getWidth());  
 stageMain.setMaxHeight(stageMain.getHeight());  
 stageMain.setMinWidth(stageMain.getWidth());  
 stageMain.setMinHeight(stageMain.getHeight());  
 }  
  
 }  
  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

Some code here was generated by the IDE, mainly the package and class (which I modified the package name of) and the two methods. I wrote the code inside the start method.

Controller Class

package squareAndCube;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
  
  
public class Controller {  
  
 @FXML private TextField txtExponent;  
 @FXML private TextField txtBase;  
 @FXML private Label lblResult;  
  
 public void changeText()  
 {  
 /\*default values for base and exponent. If a NumberFormatException is thrown (which it is whenever  
 Double.parseDouble() is ran on not a number) then these default values will be used, as this method  
 is ran on any text change, and resets base and exponent any time it is invoked.\*/  
 Double base = 1.0;  
 Double exponent = 0.0;  
  
 try  
 {  
 /\*get the values of the text inputs and set them to their respective variable, converting them to  
 a double\*/  
 base = Double.*parseDouble*(txtBase.getText());  
 exponent = Double.*parseDouble*(txtExponent.getText());  
 }  
 catch (NumberFormatException e) {  
 /\*On NumberFormatException, do nothing. Suppresses errors in the console which we do not need to  
 know about, this is an expected and solved issue.\*/  
 }  
  
 //calculate the new result, based on the current values of exponent and base, then set it to the output.  
 Double result = Math.*pow*(base, exponent);  
 lblResult.setText(result.toString());  
 }  
}

The package and the class were auto-generated by the IDE, everything else I wrote.

Scene Builder generated FXML:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.control.TextField?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.RowConstraints?>  
<?import javafx.scene.text.Font?>  
  
<GridPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="100.0" prefWidth="255.0" xmlns="http://javafx.com/javafx/9" xmlns:fx="http://javafx.com/fxml/1" fx:controller="squareAndCube.Controller">  
 <columnConstraints>  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="87.0" minWidth="10.0" prefWidth="40.0" />  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="111.0" minWidth="0.0" prefWidth="41.0" />  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="159.0" minWidth="10.0" prefWidth="23.0" />  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="210.0" minWidth="10.0" prefWidth="112.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 </rowConstraints>  
 <children>  
 <Label id="lblResult" fx:id="lblResult" text="1.0" GridPane.columnIndex="3" GridPane.rowIndex="1" />  
 <Label id="lblSymbolEquals" fx:id="lblSymbolEquals" text="=" GridPane.columnIndex="2" GridPane.rowIndex="1">  
 <font>  
 <Font size="29.0" />  
 </font>  
 </Label>  
 <TextField id="txtBase" fx:id="txtBase" alignment="CENTER" onKeyReleased="#changeText" prefHeight="30.0" prefWidth="0.0" promptText="Base" styleClass="txt" text="1" GridPane.rowIndex="1" />  
 <TextField id="txtExponent" fx:id="txtExponent" alignment="CENTER" onKeyReleased="#changeText" prefHeight="28.0" prefWidth="42.0" promptText="Exp." styleClass="txt" text="0" GridPane.columnIndex="1" />  
 </children>  
</GridPane>

All was generated using Gluon Scene Builder, which I designed in its GUI.

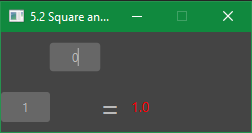
JavaFX CSS

GridPane {  
 -fx-background-color: #444444;  
}  
  
.txt {  
 -fx-text-fill: #bbbbbb;  
 -fx-background-color: #676767;  
}  
  
#lblSymbolEquals {  
 -fx-text-fill: #bbbbbb;  
}  
  
#lblResult {  
 -fx-text-fill: #ff0000;  
 -fx-font-size: 14px;  
}

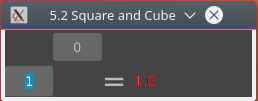
All was written by me.

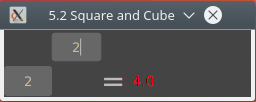
Output

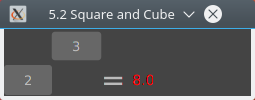
(Windows 10)

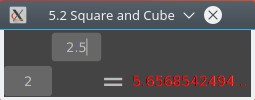


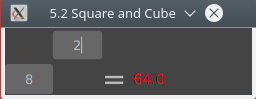
(Kbuntu)

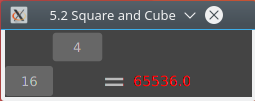


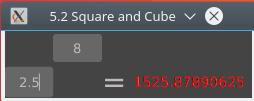












### Exercise 5.3. Payslips

* Design, write, and test a program to input 5 pairs of numbers, the first number representing hours worked and the second number representing rate of pay per hour for each of 5 workers. For each worker, calculate and output the gross pay earned.

**CLI**

Design

Pseudocode

Define a custom datatype “CalcWork” that has inHoursWorked, inRatePay and outGrossPay.

Define array “work” as type CalcWork, with a size of 5 elements.

Set i to 0;

For i to 4

Set work[i].inHoursWorked to value set by the client

Set work[i].inRatePay to value set by the user

Set work[i] outGrossPay = inHours\*inRatePay

End

For i to 4

Output work[i].outGrossPay

End

Code

import java.util.Scanner;

import java.util.InputMismatchException;

//Class defines a custom data structure for an array.

class payCalcWork

{

//define variables for the object.

public double inHoursWorked;

public double inRatePay;

public double outGrossPay;

//method for calculating the gross pay.

public void calcGrossPay()

{

//hours worked \* hourly rate, then assign it to the gross pay of the current object.

this.outGrossPay = inHoursWorked \* inRatePay;

}

}

//main class

class payslips

{

public static void main(String[] args)

{

//create an array for my class, that has 5 elements.

CalcWork[] work = new CalcWork[5];

//create a variable to input data from a terminal.

Scanner input = new Scanner(System.in);

for (int i = 0; i < 5; i++)

{

//initiate the current array position as my class.

work[i] = new CalcWork();

try

{

//Input the next hours worked and store it in the current array row, in the hours worked variable.

System.out.print("[" + i + "] Input Hours Worked: ");

work[i].inHoursWorked = input.nextDouble();

//Input next hourly rate, store in same row, in the hourly rate variable.

System.out.print("[" + i + "] Input Hourly Rate: ");

work[i].inRatePay = input.nextDouble();

}

catch (InputMismatchException e)

{

//catch InputMismatchExceptions, generated when inputted value is not the correct datatype (double)

//inform the user what went wrong and exit.

System.out.println("Inputted values must be numbers.");

System.exit(0);

}

//invoke the method calcGrossPay for the current array row.

work[i].calcGrossPay();

}

//labels following output.

System.out.println();

System.out.println("Gross Pays:");

//print each gross pay.

for (int i = 0; i < 5; i++)

{

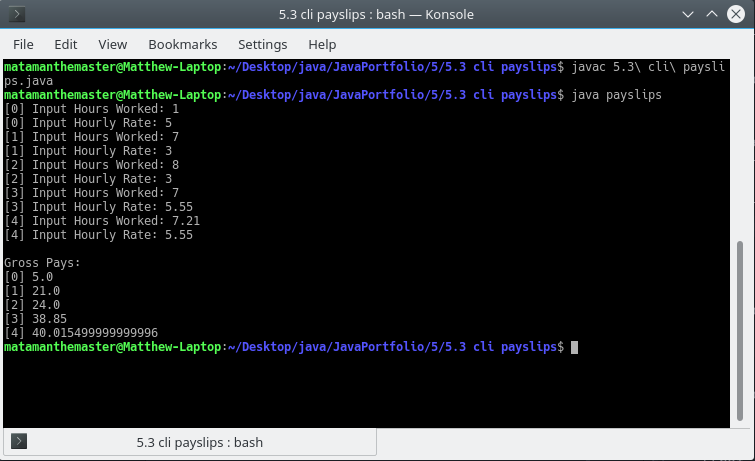
System.out.println("[" + i + "] " + work[i].outGrossPay);

}

}

}

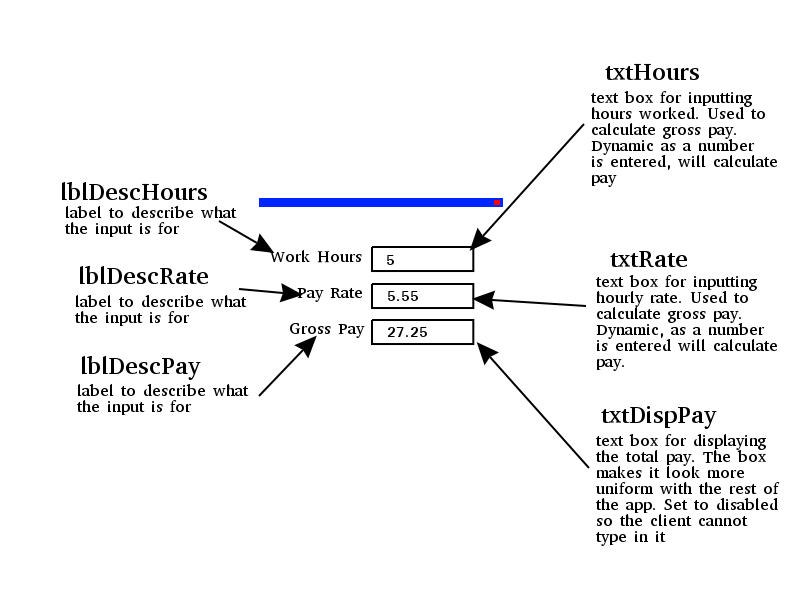
Output



**GUI**

Design

GUI Design



PseudocodeCode

On event txtHours’ text being changed

Set inHoursWorked to the text value of txtHours converted to double

Set inRatePay to the text value of txtRate converted to double

Set txtDispPay’s text to inHoursWorked \* inRatePay

End

On event txtRate’s text being changed

Set inHoursWorked to the text value of txtHours converted to double

Set inRatePay to the text value of txtRate converted to double

Set txtDispPay’s text to inHoursWorked \* inRatePay

End

Main Class

package payslips;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class Main extends Application {  
  
 @Override  
 public void start(Stage stageMain) throws Exception  
 {  
 //get the contents of Main.fxml to append to the scene  
 Parent root = FXMLLoader.*load*(getClass().getResource("payslips.fxml"));  
  
 //set the scene with the contents of Main.fxml and default size.  
 Scene sceneMain = new Scene(root, 500, 250);  
  
  
 //get the css files and add it to the scene's stylesheets.  
 sceneMain.getStylesheets().add(getClass().getResource("css/style.css").toExternalForm());  
 sceneMain.getStylesheets().add(getClass().getResource("css/colour.css").toExternalForm());  
  
 //set stage minimum sizes  
 stageMain.setMinWidth(300);  
 stageMain.setMinHeight(150);  
  
 //set title, scene and show the stage.  
 stageMain.setTitle("5.3 Payslips");  
 stageMain.setScene(sceneMain);  
 stageMain.show();  
 }  
  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

Part of this class were auto-generated by my IDE. Everything written within the method start was written by me.

Controller Class

package payslips;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.TextField;  
  
public class Controller {  
  
 @FXML private TextField txtHours;  
 @FXML private TextField txtRate;  
 @FXML private TextField txtDispPay;  
  
 public void textChanged()  
 {  
 /\* define variables to assign the values in the text boxes, which are strings. Default to 0.0 in case  
 of NumberFormatException\*/  
 Double inHoursWorked = 0.0;  
 Double inRatePay = 0.0;  
  
 try  
 {  
 /\*get the values of the text inputs and set them to their respective variable, converting them to  
 a double\*/  
 inHoursWorked = Double.*parseDouble*(txtHours.getText());  
 inRatePay = Double.*parseDouble*(txtRate.getText());  
 }  
 catch (NumberFormatException e){  
 /\*On NumberFormatException, do nothing. Suppresses errors in the console which we do not need to  
 know about, this is an expected and solved issue.\*/  
 }  
  
 //calculate the gross pay as double, then set it to the text of the display text box.  
 Double outGrossPay = inHoursWorked \* inRatePay;  
  
 txtDispPay.setText(outGrossPay.toString());  
 }  
}

Only the class definition and package definition were auto-generated. The rest was written by me.

Payslips FXML

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import java.lang.String?>  
<?import javafx.geometry.Insets?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.control.TextField?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.RowConstraints?>  
<?import javafx.scene.text.Font?>  
  
<GridPane xmlns="http://javafx.com/javafx/9.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="payslips.Controller">  
 <columnConstraints>  
 <ColumnConstraints />  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="100.0" minWidth="100.0" prefWidth="100.0" />  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="1.7976931348623157E308" minWidth="100.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 </rowConstraints>  
 <children>  
 <Label id="lblDescHours" fx:id="lblDescHours" prefHeight="16.0" prefWidth="81.0" styleClass="lblDesc" text="Work Hours" GridPane.columnIndex="1" GridPane.rowIndex="1">  
 <font>  
 <Font size="15.0" />  
 </font></Label>  
 <Label id="lblDescRate" fx:id="lblDescRate" prefHeight="16.0" prefWidth="63.0" styleClass="lblDesc" text="Pay Rate" GridPane.columnIndex="1" GridPane.rowIndex="2">  
 <font>  
 <Font size="15.0" />  
 </font></Label>  
 <Label id="lblDescPay" fx:id="lblDescPay" prefHeight="16.0" prefWidth="69.0" styleClass="lblDesc" text="Gross Pay" GridPane.columnIndex="1" GridPane.rowIndex="3">  
 <font>  
 <Font size="15.0" />  
 </font></Label>  
 <TextField id="txtHours" fx:id="txtHours" onKeyReleased="#textChanged" styleClass="txt" GridPane.columnIndex="2" GridPane.rowIndex="1">  
 <GridPane.margin>  
 <Insets right="20.0" />  
 </GridPane.margin>  
 <font>  
 <Font size="15.0" />  
 </font></TextField>  
 <TextField id="txtRate" fx:id="txtRate" onKeyReleased="#textChanged" styleClass="txt" GridPane.columnIndex="2" GridPane.rowIndex="2">  
 <GridPane.margin>  
 <Insets right="20.0" />  
 </GridPane.margin>  
 <font>  
 <Font size="15.0" />  
 </font></TextField>  
 <TextField id="txtDispPay" fx:id="txtDispPay" editable="false" text="0.0" GridPane.columnIndex="2" GridPane.rowIndex="3">  
 <styleClass>  
 <String fx:value="txt" />  
 <String fx:value="txtDisp" />  
 </styleClass>  
 <GridPane.margin>  
 <Insets right="20.0" />  
 </GridPane.margin>  
 <font>  
 <Font size="15.0" />  
 </font>  
 </TextField>  
 </children>  
</GridPane>

The interface was designed by me in Gluon Scene Builder, which generated the FXML file above.

Style.css

#txtDispPay {  
 -fx-font-weight: bold;  
}

This was all written by me.

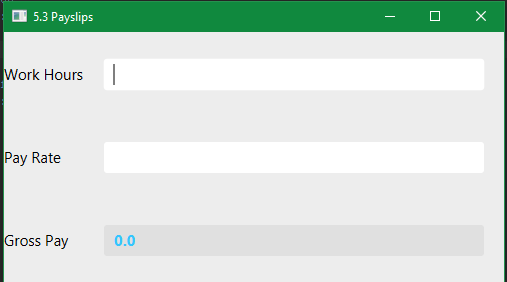
Colour.css

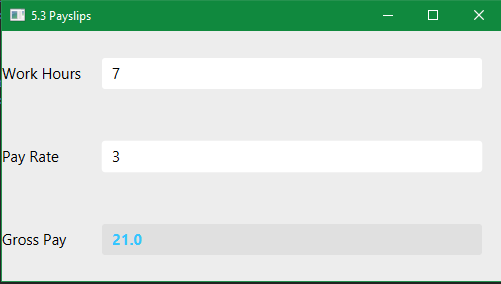
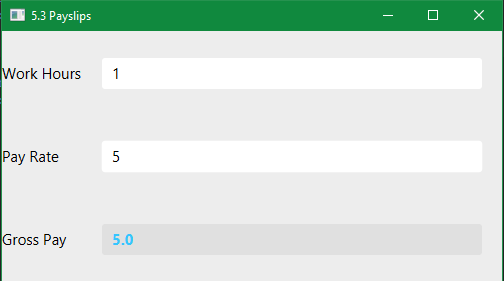
GridPane {  
 -fx-background-color: #ededed;  
}  
  
.lblDesc {  
 -fx-text-fill: #000000;  
}  
  
.txt {  
 -fx-text-fill: #000000;  
 -fx-background-color: #ffffff;  
}  
  
#txtDispPay {  
 -fx-text-fill: #35c5ff;  
 -fx-background-color: #e0e0e0;  
}

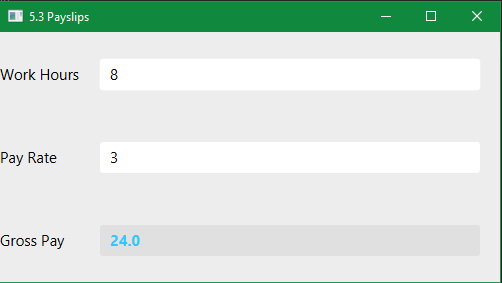
This was all written by me

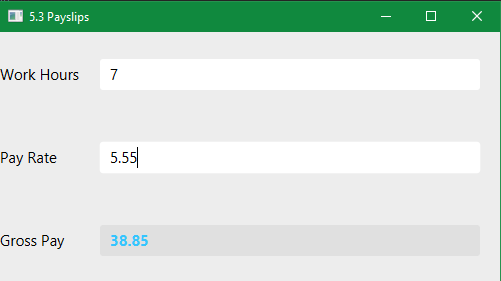
Output

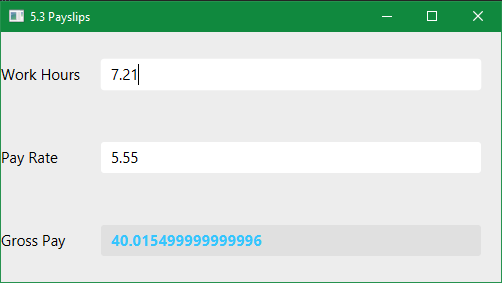
(Windows)

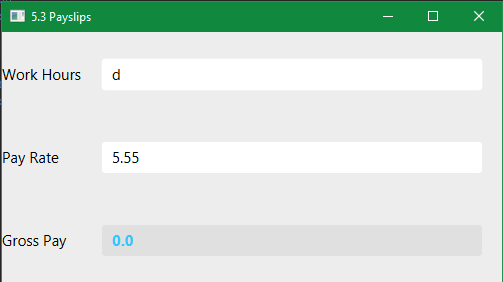


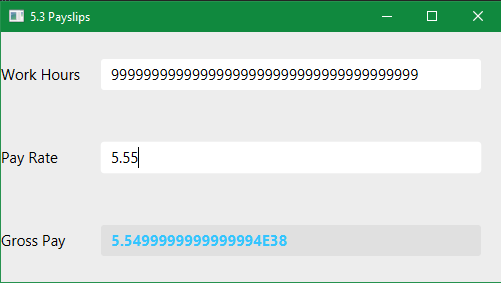


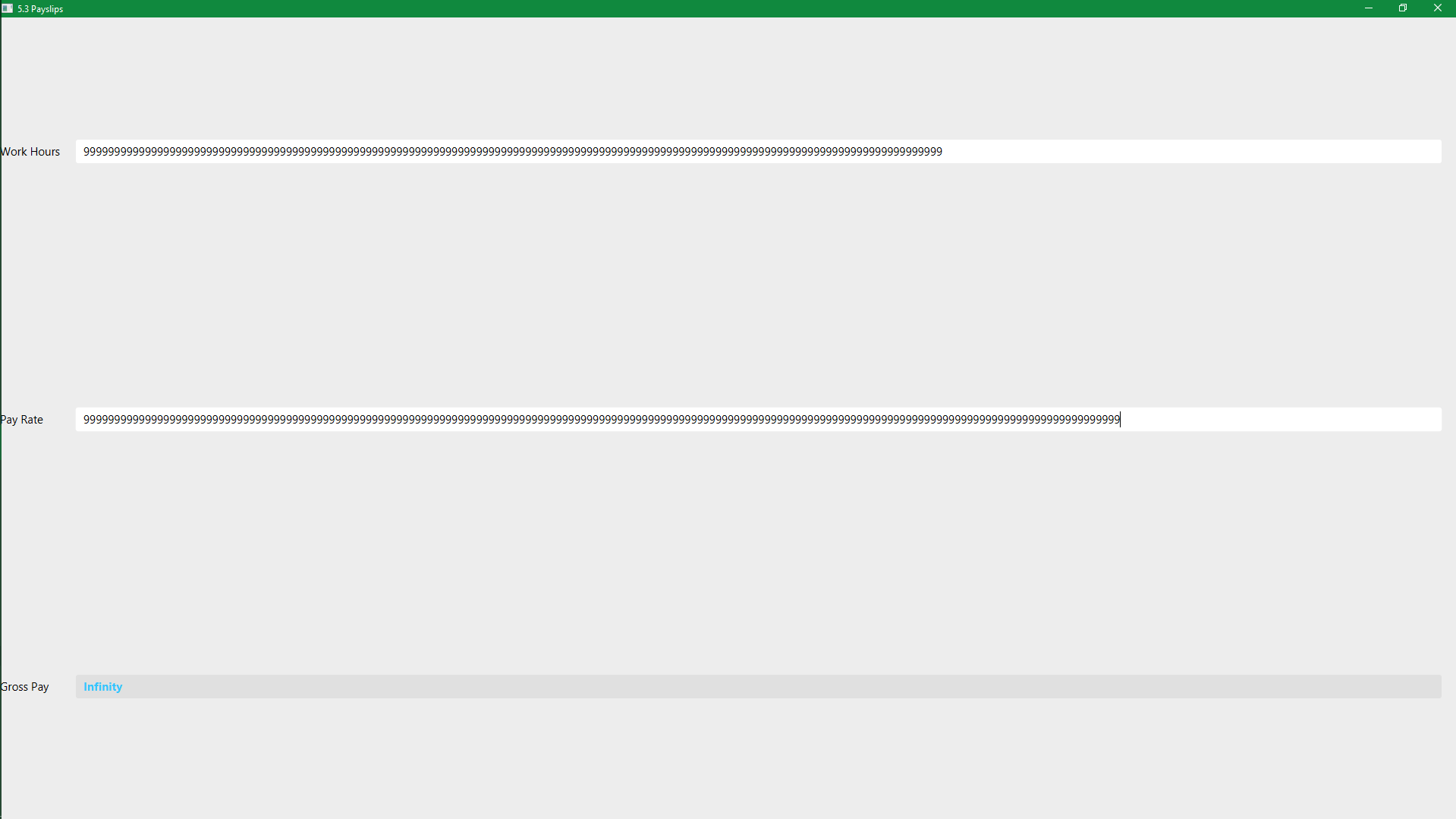












Exercise 5.4. Commission

* Design, write, and test a program to input the commission figures for 10 sales staff; calculate and output the total commission overall and the average commission.

### Exercise 5.5. Student marks

* Design, write, and test a program to input the number of students sitting an examination, followed by an exam mark for each of the students. For each student, display the mark and a message 'Pass' or 'Fail', depending on whether the mark is at least 40 (for a Pass) or less than 40 (for a Fail).

### Exercise 5.6. Multiplication table

* Input an integer between 1 and 100 and output the multiplication table for that integer

(i.e. 1  X integer = ?   
       2  X integer = ?

....

12 X integer = ?).

### Exercise 5.7. Even more coffee

* Modify your coffee survey program (which used a for loop) to use a while loop instead, as in the example. You may choose to open your original program and save it with a new name. Remember to modify the class name too though!
* A new requirement for the program is to output the number of people surveyed in the report. Use a new variable called personCount to do this, which you should update every time you run through the loop.

### Exercise 5.8. Does not exist!

### Exercise 5.9. More student marks

* Design, write and test a program to input student marks in a test. Output 'Pass' if the mark is 40 or above, otherwise output 'Fail'. Input a mark of -1 to finish the program.
* Modify the design and the code of the above program to output the number of students who took the test, the number of passes, the number of fails, and the overall average mark.
* What happens in the above program if there are no students? Make sure your program can cope with zero or more students.

### Exercise 5.10. Payroll

* Design, write and test a program to process a payroll. You should input the employee name, their hourly rate, and the number of hours each employee has worked. Output the gross pay and the net pay (with a tax rate of 30%). When there are no more employees, input the word "End", instead of the employee name.

### Exercise 5.11. Higher or lower?

* A simple two player game involves player 1 typing in a number, and player 2 guessing the number. Each time player 2 guesses the wrong answer, the computer must reply "Try higher" or "Try lower", to give the player a clue for the next try. When the correct number is guessed, the number of guesses taken will be output, and the program will end. Design and write a program to play the guessing game.

### Exercise 5.12. Module marks

* Design, write and test a program to process student module marks. The input data for each student consists of their name, followed by a group of 6 marks for each student (e.g. 56, 45, 75, 32, 68, 39). For each student, calculate the average mark and print a result depending on this average - 'Pass' if at least 40, otherwise 'Fail'. The program finishes when the user types “End” when prompted for the student’s name. Your program should process any number of students.

### Exercise 6.1. Hello again

* Design, build and test a program that asks the user to input a number, and output the message "Hello world" that number of times. Your program must have a separate method which outputs the set of messages.

### Exercise 6.2. Circle area

* Design, build, and test a program that calculates the area of a circle, based on the user inputting its radius. You must write a separate method to calculate the formula: Area = πr2

### Exercise 6.3. Volume

* Add a new method to your circle area calculation program which calculates the volume of a cylinder with the same base area as the circle. Use the formula: Volume = area \* height. Hint: there are lots of ways to solve this problem, but the best way is to use the result returned from the area calculation method as an input to the calculate volume method!

### Exercise 6.4. More pay

* Design, build and test a program to input the number of workers followed by their name, hours worked, hourly rate of pay, overtime rate of pay, status (married or single) for each. The main program must use separate methods and appropriate parameters. Suggested methods include:
  + Calculate the gross pay (overtime rate is paid for any number of hours above 40);
  + Calculate the amount of tax paid (25% if Married and 30% if Single);
  + Display all the details as a payslip on the computer screen. Hint: you can pass **String** parameters too!

### Exercise 7.1. Ice skaters

* Design and build a program to enable a user to input an array of six integers, which represent the scores allocated by judges in an ice skating competition. Once all scores have been entered into the array, use it to calculate and output the average score.
* Modify your program to omit the lowest and highest score allocated, taking the middle four marks as the basis on which to calculate the average.

### Exercise 7.2. Initialise a list

* Modify the **NameList** program so that no user input is required to enter the list of names. Instead of the user entering ten names, initialise a new list of names in the String declaration statement.

### Exercise 7.3. Saver bonus

* Finally, you will show how you can use arrays to manipulate grouped information. Every Christmas, the Famously Good Building Society awards a bonus to all its savers, expressed as a percentage of the balance of each account. Design, code and test a program to enable a user to type in four savers’ balances and the percentage bonus (which is the same for everyone). The program should then output the new balances for each saver.

### Exercise 7.4. Savers with methods

* Modify the Famously Good Building Society program to use methods. Use separate methods for the following functions:
  + enable a user to type in four savers’ balances
  + update the balances by applying the percentage bonus
  + output the new balances for each saver.

### Exercise 8.1. Author

* Add a new attribute to the **Book** class called **author**. In your main method, after comparing dates, write another test to establish if the author of the two books is the same person.

### Exercise 8.2. Music

* Draw a class diagram which describes what you know about three musical instruments. The instruments are piano, guitar, and violin. All of them have strings and all of them make a sound. Pianos have keys and violins have bows. You can strum a guitar and a violin, but you can't strum a piano - you have to hit the keys. Show some other properties and actions that can be performed with the instruments. To structure your class model, use a super class, called **Instrument**.

### Exercise 8.3. Museum

* Draw a class diagram and write the Java code to represent a vehicle museum. Vehicles have a licence plate, a year of manufacture, a value, and a colour. The museum has cars, which have a number of doors, seats, engine type (petrol or diesel), and engine size in litres. The museum also has motorbikes, which have a bike type (sports, tourer, or trails), and engine size in cubic centimetres (cc). Note that you are not expected to run the program – all you need to do is show how the class structures can be coded in Java.

### Exercise 8.4. Museum inventory

* Here's a reminder of an exercise from earlier in this chapter: *draw a class diagram and write the Java code to represent a vehicle museum. Vehicles have a licence plate, a year of manufacture, a value, and a colour. The museum has cars, which have a number of doors, seats, engine type (petrol or diesel), and engine size in litres. The museum also has motorbikes, which have a bike type (sports, tourer, or trails), and engine size in cubic centimetres (cc).*
* Using the knowledge you have gained since tackling this exercise, add the following features to the program in the form of a menu:
  + add new vehicles (bikes or cars)
  + list all vehicles manufactured in a given year
  + list all vehicles of 1 litre or more (1 litre = 1000cc)
  + list all cars
  + list all bikes
    - HINT: for the above two options, you can get a string representation of an object's class by calling the following method **getClass().toString()**, so to print object x's class, you would do this:   
      **System.out.print( x.getClass().toString() );**