Goals & Objectives

The goal of this program is to test the relationships between two time intervals and check the existence of one Time within the two intervals. The program will identify if two time intervals overlap and if the provided time is contained within the two time intervals. The program uses user input in the form of a GUI and has two outputs.

Lessons Learned

The lessons I learned in this project were 1) learning some of the column constraints and general formatting of the GUI to make it even across the three columns. I learned a better build process within the source code to set up the GUI to minimize the amount of code within the start method, with some trade off on the variables and nodes. 2) Generic classes and methods were difficult to understand at first and I had to go through the lessons in Revel a few times and my own research before I had a better understanding of how to use the Generic classes and methods. Once I got the grasp of it, writing the rest of the program became a little bit easier. 3) Comparing time intervals didn’t work well. When I compared hours 11 vs 10, I would return a 1 in the corresponding method, but when comparing a 1 vs a 12, I would not. I decided to convert all times to a 24-hour time and convert them back. It took a little extra code to get it to work, but the end product worked well.

Functional Requirements

1. Present a User Interface for input in the program.
2. Create Time objects for each time-related textfield.
3. Create two Interval objects to store Time objects.
4. Compare Interval relationships
5. Display results of interval comparison in Textfield.
6. Compare input time to Time Intervals
7. Display results of Time check in Textfield.

Pseudocode

Import JavaFX Application

Import JavaFX Geometry

Import JavaFX Scene

Import JavaFX Stage

Class Project4 extends Application {

Declare Buttons btnCompareInts and btnCheckTime

Declare TextFields tfStart1, tfStart2, tfEnd1, tfEnd2, tfCheck, tfOutput

Public String method getCompareIntervals(Requires Interval<Time> i1, Interval<Time> i2) {

Declare message as String

IF (i1.subinterval(i2)) {

message = “Interval 1 is a sub-interval of interval 2.”

Else IF (i2.subinterval(i1)) {

message = “Interval 2 is a sub-interval of interval 1.”

Else IF ( i1.overlaps(i2)) {

message = “The intervals overlap”

Else {

message = “The intervals are disjoint”

End

Return Message

End

Public String method getCompareIntervals(Requires Interval<Time> i1, Interval<Time> i2, Time t)

Declare message as String

IF (i1.within(t) && i2.within(t)) {

message = “Both intervals contain the time ” + t.toString()

Else IF (i1.within(t) && !i2.within(t)) {

message = “Interval 1 contains the time ” + t.toString()

Else IF (!i1.within(t) && i2.within(t)) {

message = “Interval 2 contains the time ” + t.toString()

Else {

message = “Neither interval contains the time ” + t.toString()

End

Return Message

End

Public void Method start(Requires Stage primaryStage) {

Declare GridPane gpMain as buildGridPane()

btnCompareInts.setOnAction ( e -> {

Try {

Declare startTime1 as new Time(tfStart1 text)

Declare startTime2 as new Time(tfStart2 text)

Declare endTime1 as new Time(tfEnd1 text)

Declare endTime2 as new Time(tfEnd2 text)

Declare i1 as new Interval containing startTime1 and endTime1

Declare i2 as new Interval containing startTime2 and endTime2

Declare intervalMessage as getCompareIntervals(i1, i2)

Output intervalMessage to tfOutput

Catch (InvalidTime it) {

Output it.getMessage to tfOutput

End

btnCheckTime.setOnAction ( e -> {

Try {

Declare startTime1 as new Time(tfStart1 text)

Declare startTime2 as new Time(tfStart2 text)

Declare endTime1 as new Time(tfEnd1 text)

Declare endTime2 as new Time(tfEnd2 text)

Declare i1 as new Interval containing startTime1 and endTime1

Declare i2 as new Interval containing startTime2 and endTime2

Declare checkTime as new Time(tfCheck text)

Declare intervalMessage as getCompareIntervals(i1, i2, checkTime)

Output intervalMessage to tfOutput

Catch (InvalidTime it) {

Output it.getMessage to tfOutput

End

Declare scene as new Scene(gpMain, 550px wide, 250px height)

Set primaryStage title as “Time Interval Checker”

Set primaryStage scene as scene

Show primaryStage

End

Public GridPane Method buildGridPane() {

Declare gpMain as new GridPane

Declare lblStartTime, lblEndTime, lblTimeInt1, lblTimeInt2, lblTimeCheck as new Labels

Set lblStartTime text as “Start Time”

Set lblEndTime text as “End Time”

Set lblTimeInt1 text as “Time Interval 1”

Set lblTimeInt2 text as “Time Interval 2”

Set lblTimeCheck text as “Time to Check”

Declare btnCompareInts as new Button with label “Compare Intervals”

Set btnCompareInts minimum width as 50px

Declare btnCheckTime as new Button with label “Check Time”

Set btnCheckTime minimum width as 50px

Declare tfStart1 as new TextField

Set tfStart1 alignment to Center

Set tfStart1 style to gray text

Set tfStart1 text to “HH:MM AM/PM”

Set tfStart1 mouse clicked action to clear text if “HH:MM AM/PM” and set style to black text

Declare tfStart2 as new TextField

Set tfStart2 alignment to Center

Set tfStart2 style to gray text

Set tfStart2 text to “HH:MM AM/PM”

Set tfStart2 mouse clicked action to clear text if “HH:MM AM/PM” and set style to black text

Declare tfEnd1 as new TextField

Set tfEnd1 alignment to Center

Set tfEnd1 style to gray text

Set tfEnd1 text to “HH:MM AM/PM”

Set tfEnd1 mouse clicked action to clear text if “HH:MM AM/PM” and set style to black text

Declare tfEnd2 as new TextField

Set tfEnd2 alignment to Center

Set tfEnd2 style to gray text

Set tfEnd2 text to “HH:MM AM/PM”

Set tfEnd2 mouse clicked action to clear text if “HH:MM AM/PM” and set style to black text

Declare tfCheck as new TextField

Set tfCheck alignment to Center

Declare tfOutput as new TextField

Declare col1 as new ColumnConstraints(100px wide)

Declare col2 as new ColumnConstraints(200px wide)

Declare col3 as new ColumnConstraints(200px wide)

Add col1, col2, col3 to gpMain

Set col2, col3 Horizontal Alignment to Center

Set gpMain padding to 20px Insets

Set gpMain Horizontal Gap to 5px

Set gpMain Vertical Gap to 5px

Add lblStartTime to gpMain

Add lblEndTime to gpMain

Add lblTimeInt1 to gpMain

Add tfStart1 to gpMain

Add tfEnd1 to gpMain

Add lblTimeInt2 to gpMain

Add tfStart2 to gpMain

Add tfEnd2 to gpMain

Add btnCompareInts to gpMain

Add lblTimeCheck to gpMain

Add tfCheck to gpMain

Add btnCheckTime to gpMain

Add tfOutput to gpMain

Return gpMain

End

End

Class Interval<E extends Comparable<E>> {

Declare start as private E

Declare end as private E

Public Constructor Interval(Requires E start, E end) {

Store this.start as start

Store this.end as end

End

Public E Method getStart() {

Return this.start

End

Public E Method getEnd() {

Return this.end

End

Public Boolean Method within(Requires E o) {

Return o.compareTo(this.start) >= 0 AND o.compareTo(this.end) <= 0

End

Public Boolean Method subinterval(Requires Interval<E> o) {

Return o.start.compareTo(this.start) >= 0 AND o.end.compareTo(this.end) <= 0

End

Public Boolean Method overlaps(Requires Interval<E> o) {

Return this.start.compareTo(o.end) <= 0 AND this.end.compareTo(o.start) >= 0

End

End

Class Time implements Comparable<Time> {

Declare hours as private int

Declare minutes as private int

Declare meridian as private String

Public Constructor Time(Requires int hours, int minutes, String meridian) throws InvalidTime {

If (hours < 0 or hours > 12 or minutes < 0 or minutes > 59 {

Throw new InvalidTime with message “Invalid Time Format”

End

Store this.hours as hours

Store this.minutes as minutes

Store this.meridian as meridian

End

Public Constructor Time(Requires String time) throws InvalidTime {

Declare parts as String Array and store time split on whitespace in String

If Length of parts is not 2 {

Throw new InvalidTime with message “Invalid Time Format”

End

Declare timeParts as StringArray and store time[0] split on colon (“:”) in String

If Length of timeParts is not 2 {

Throw new InvalidTime with message “Invalid Time Format”

End

Declare hours as int and store Integer.parseInt(timeParts[0])

Declare minutes as int and store Integer.parseInt(timeParts[1])

Declare meridian as String and store parts[1]

If (hours < 0 or hours > 12 or minutes < 0 or minutes > 59 {

Throw new InvalidTime with message “Invalid Time Format”

End

IF meridian equals “AM” {

If hours equals 12 {

Store this.hours as 0

ELSE {

Store this.hours as hours

End

Store this.minutes as minutes

Store this.meridian as meridian

ELSE IF meridian equals “PM” {

If hours equals 12 {

Store this.hours as hours + 12

ELSE {

Store this.hours as hours

End

Store this.minutes as minutes

Store this.meridian as meridian

ELSE {

Throw new InvalidTime with message “Invalid Meridian”

End

End

Public int Method getHours() {

IF this.hours equals 0

Return this.hours + 12

ELSE IF this.hours equals 12

Return this.hours

ELSE

Return this.hours mod 12

End

End

Public int Method getMinutes() {

Return this.minutes

End

Public String Method getMeridian() {

Return this.meridian

End

Public int Method compareTo(Requires Time o) {

IF (meridian.compareTo(o.meridian) >= 0 && hours does not equal o.hours)

Return Integer.compare(hours, o.hours)

ELSE IF (meridian.compareTo(o.meridian) >= 0 && hours does not equal o.hours) {

Return Integer.compare(o.hours, hours)

End

IF (meridian.compareTo(o.meridian) >= 0 && minutes does not equal o.minutes) {

Return Integer.compare(minutes, o.minutes)

ELSE IF (meridian.compareTo(o.meridian) >= 0 && minutes does not equal o.minutes) {

Return Integer.compare(o.minutes, minutes)

End

Return meridian.compareTo(o.meridian)

End

Override Public String Method toString() {

Return String.format(%02d:%02d %s, getHours(), getMinutes(),

getMeridian().toUpperCase())

End

End

Class InvalidTime extends Exception {

Declare message as private String

Public Constructor InvalidTime (Requires message as String) {

Store this.message as message

End

Override Public String Method getMessage() {

Return this.message

End

End

UML Diagram

|  |
| --- |
| **Project4** |
| -btnCompareInts : Button |
| -btnCheckTime : Button |
| -tfStart1 : TextField |
| -tfEnd1 : TextField |
| -tfStart2 : TextField |
| -tfEnd2 : TextField |
| -tfCheck : TextField |
| -tfOutput : TextField |
| +start(primaryStage : Stage)  +getCompareIntervals(i1: Interval<Time>, i2 : Interval<Time>) : String  +getTimeCheck(i1 : Interval<Time>, i2 : Interval<Time>, t : Time) : String  +buildGridPane() : GridPane |

|  |
| --- |
| **Interval<E>** |
| -start : E |
| -end : E |
| +Interval(start : E, end : E) |
| +getStart() : E |
| +getEnd() : E |
| +within(o : E) : Boolean  +subinterval(o : Interval<Time>) : Boolean  +overlaps(o : Interval<Time>) : Boolean |

|  |
| --- |
| **Time** |
| -hours : int |
| -minutes : int  -meridian : String |
| +Time(hours : int, minutes : int, meridian : String) |
| +Time(time : String) |
| +getHours() : Int |
| +getMinutes() : Int  +getMeridian() : String  +compareTo(o : Time) : int  +toString() : String |

|  |
| --- |
| **InvalidTime** |
| -message : String |
| +InvalidTime(message : String) |
| +getMessage() : String |

Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input/Output** | **Expected Result** | **Actual Result** | **Outcome (Pass/Fail)** |
| 1a | Present a User Interface for input in the program. | GUI launches with empty textfields for user input | GUI launches with textfields ready for user input and displays acceptable format | Pass |
| 2a | Create Time objects for each time-related textfield. | On button click, Time objects are made for each time textfield | On button click, Time objects are created for each time in textfields tfStart1, tfStart2, tfEnd1, tfEnd2, and tfCheck | Pass |
| 3a | Create two Interval objects to store Time objects. | On button click, two Interval objects are created with Time objects | On button click, Interval<Time> i1 and i2 are created to start startTime1, startTime2, endTime1, and endTime2 | Pass |
| 4a | Compare Interval relationships | On button click, intervals are passed to method to check subinterval and overlap | On button click, Interval<Time> i1 and i2 are passed to getCompareIntervals and checked if they overlap or are subintervals | Pass |
| 5a | Display results of interval comparison in Textfield. | Results of method to compare intervals in output to textfield | Results of getCompareIntervals method is output to tfOutput | Pass |
| 6a | Compare input time to Time Intervals | On button click, intervals and time objects passed to method to check if time is in intervals | On button click, Time t and Interval<Time> i1 and i2 are passed to getTimeCheck method and t checked if it is within i1 and i2 | Pass |
| 7a | Display results of Time check in Textfield. | Results of method to check time against intervals output to textfield | Results of getTimeCheck method is output to tfOutput | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input/Output** | **Expected Result** | **Actual Result** | **Outcome (Pass/Fail)** |
| 1 | Time Interval 1: 10:30 AM, 12:30 PM  Time Interval 2: 11:05 AM, 01:00 PM  Check Time: 10:45 AM | 4a: The intervals overlap  6a: Interval 1 contains time 10:45 AM | 4a: The intervals overlap  6a: Interval 1 contains time 10:45 AM | Pass |
| 2 | Time Interval 1: 08:30 AM, 04:30 PM  Time Interval 2: 11:00 AM, 12:00 PM  Check Time: 11:30 AM | 4a: Interval 2 is a sub-interval of interval 1  6a: Both intervals contain the time 11:30 AM | 4a: Interval 2 is a sub-interval of interval 1  6a: Both intervals contain the time 11:30 AM | Pass |
| 3 | Time Interval 1: 09:00 PM, 05:00 AM  Time Interval 2: 08:30 AM, 04:30 PM  Check Time: 06:30 AM | 4a: The intervals are disjoint  6a: Neither interval contains the time 06:30 AM | 4a: The intervals are disjoint  6a: Neither interval contains the time 06:30 AM | Pass |
| 4 | Time Interval 1: 08:30 AM, 09:00 AM  Time Interval 2: 08:00 AM, 11:30 AM  Check Time: 10:15 AM | 4a: Interval 1 is a sub-interval of interval 2  6a: Interval 2 contains the time 10:15 AM | 4a: Interval 1 is a sub-interval of interval 2  6a: Interval 2 contains the time 10:15 AM | Pass |

Test 1:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Test 2

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Test 3

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Test 4

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated