Devon Blank
12/3/2025
CMSC 215

Project 4 Documentation

Approach

For this project, I began by creating the 4 Java files that would become the four classes that make up this program:

- Project4.java
- Time.java
- Interval.java
- InvalidTime.java

I began by creating the InvalidTime class, which worked as a checked exception used to report invalid time inputs. Creating this class first enabled the Time class to handle errors such as out-of-range values, non-numeric inputs, or incorrect meridian format.

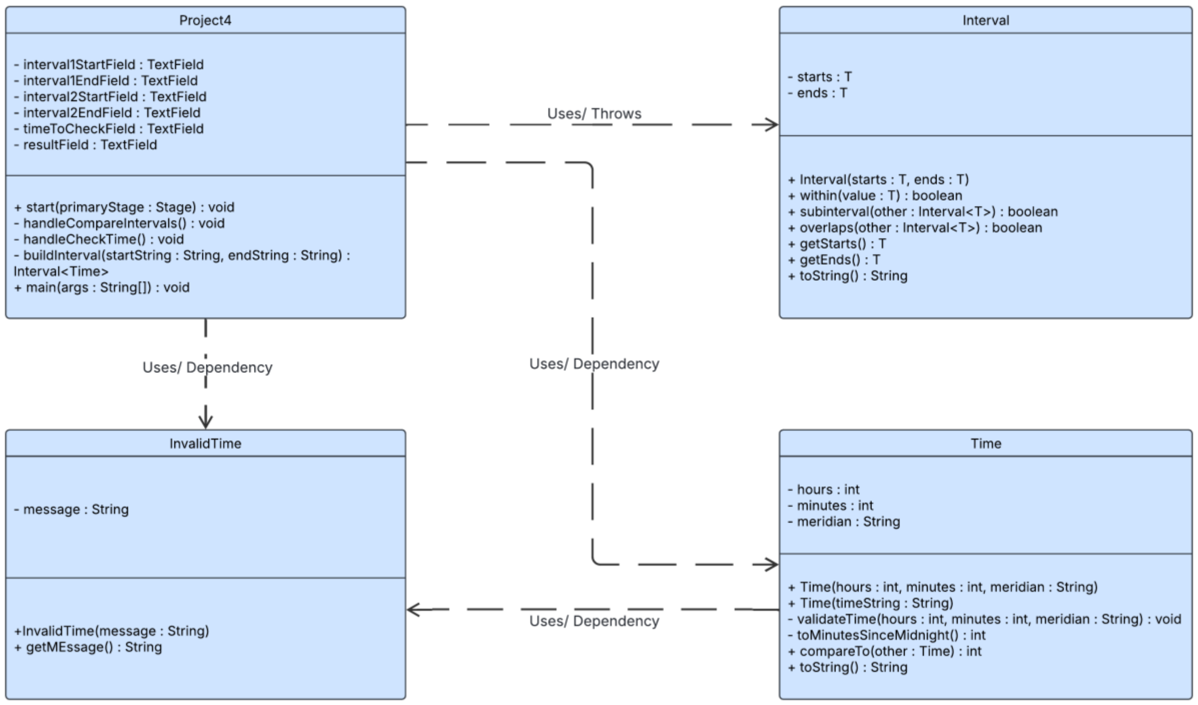
Next, I implemented the Time class. The goal with this class was to build an immutable object that represents a single time in the form "HH:MM AM/PM." I made two constructors, one that accepts numeric values and one that parses a string. I used a private validation method to ensure consistency. The Time class also implements Comparable<Time>, which allows Time objects to be ordered chronologically. This comparison logic was essential for comparing the intervals.

After Time was complete, I began working on the Interval class. The Interval class is immutable and includes methods to check if two intervals overlap or are disjoint. These methods encapsulate the comparison logic used by the GUI.

Lastly, I built the Project4 JavaFX class, which implements the GUI. This class creates the input fields, buttons, and output fields that the user interacts with. When the user clicks the "Compare Intervals" or "Check Time" buttons, Project 4 constructs the necessary Time and Interval objects and uses their methods to determine the correct output message. All exceptions are caught and displayed to the user in a readable format. This approach kept the GUI focused strictly on input/output, while the model classes handled logic and validation.

<u>UML</u>

Figure 1. UML diagram illustrating the relationship between Project4, Time, Interval, and InvalidTime.


Project4
- interval1StartField : TextField
- interval1EndField : TextField
- interval2StartField : TextField
- interval2EndField : TextField
- timeToCheckField : TextField
- resultField : TextField
+ start(primaryStage : Stage) : void
- handleCompareIntervals() : void
- handleCheckTime() : void
- buildInterval(startString : String, endString : String) : Interval<Time>
+ main(args : String[]) : void
Interval
- starts : T
- ends : T
+ Interval(starts : T, ends : T)
+ within(value : T) : boolean
+ subinterval(other : Interval<T>) : boolean
+ overlaps(other : Interval<T>) : boolean
+ getStarts() : T
+ getEnds() : T
+ toString() : String
Uses/ Throws
Uses/ Dependency
Uses/ Dependency
InvalidTime
- message : String
+InvalidTime(message : String)
+ getMEssage() : String
Time
- hours : int
- minutes : int
- meridian : String
+ Time(hours : int, minutes : int, meridian : String)
+ Time(timeString : String)
- validateTime(hours : int, minutes : int, meridian : String) : void
- toMinutesSinceMidnight() : int
+ compareTo(other : Time) : int
+ toString() : String
Uses/ Dependency

The Unified Modeling Language diagram above outlines the driver class Project4, which depends on Time, Interval, and InvalidTime classes. Project4 extends the JavaFX Application class, meaning that it is a specialized type of graphical application. It has dependency relationships with Time and Interval because the GUI creates and compares Time and Interval objects to evaluate user-entered time intervals. Project4 also depends on InvalidTime, which is thrown by the Time class during input validation and caught in the GUI to report errors to the user.

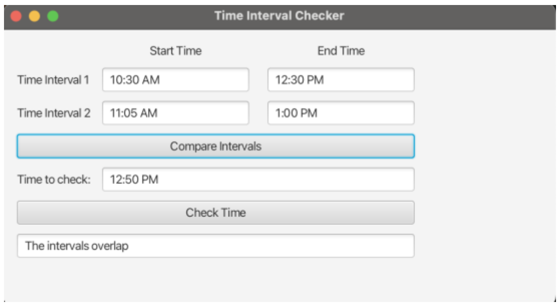
Test Plan

Test	**Purpose**	**User input**	**Expected output**	**Actual output**	**Pass / Fail**
1.Basic Overlap	Ensure overlap message functionality	Screenshot1	The intervals overlap	Screenshot 1	Pass
2.Interval 1 is subinterval of interval 2	Ensure subinterval functionality	Screenshot2	Interval 1 is a sub-interval of interval 2	Screenshot 2	Pass
3. Interval 2 is subinterval of invtreval1	Ensure subinterval functionality	Screenshot3	Interval 2 is a sub-interval of interval 1	Screenshot 3	Pass
4. Disjoint intervals	Ensure disjoint functionality	Screenshot4	The intervals are disjoint	Screenshot 4	Pass
5.intervals touching at endpoint	Does it count as overlap	Screenshot5	The intervals overlap	Screenshot 5	Pass
6. Time is inside both intervals	Test functionality of both intervals	Screenshot 6	Both intervals contain the time 12:00 PM	Screenshot 6	Pass
7.Time only in interval 1	Ensure functionality of interval 1	Screenshot 7	Only Interval 1 contains the time 11:00 AM	Screenshot 7	Pass
8. Time only in interval 2	Ensure functionality of interval 2	Screenshot 8	Only Interval 2 contains the time 10:00 AM	Screenshot 8	Pass
9. Time in neither interval	Ensure correctly communicates neither interval	Screenshot 9	Neither interval contains the time 09:00 AM	Screenshot 9	Pass
10. Time exactly at start of interval 1, outside of interval 2	Test edge case and outside of another interval	Screenshot 10	Only Interval 1 contains the time 09:00 AM	Screenshot 10	Pass
11. Invalid hour in interval	Hours out of range test	Screenshot 11	Error: Hours must be between 1 and 12	Screenshot 11	Pass
12. Invalid minutes in interval	Minutes out of range	Screenshot 12	Error: Minutes must be between 0 and 59	Screenshot 12	Pass
13. Invalid meridian interval	Neither AM or PM	Screenshot 13	Error: Meridian must be either 'AM' or 'PM'	Screenshot 13	Pass

14. Non-numeric time input	Ensure Non Numeric input error functionality	Screenshot 14	Error: Hours and minutes must be numeric	Screenshot 14	Pass
15. Interval with start after end	Ensure correct error is thrown	Screenshot 15	Error: Start of interval cannot be greater than the end	Screenshot 15	Pass

Screenshots 1-4:

Screenshot1



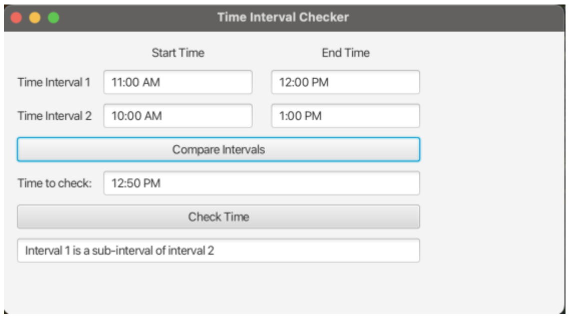
Time Interval 1: 10:30 AM | 12:30 PM

Time Interval 2: 11:05 AM | 1:00 PM

Time to check: 12:50 PM

The intervals overlap

Screenshot 2



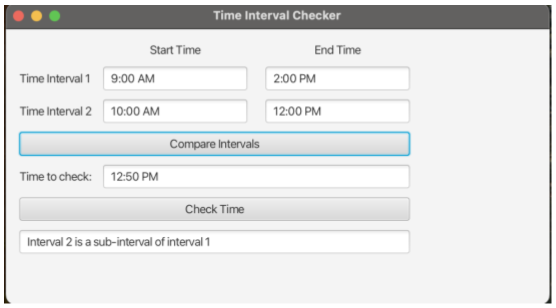
Time Interval 1: 11:00 AM | 12:00 PM

Time Interval 2: 10:00 AM | 1:00 PM

Time to check: 12:50 PM

Interval 1 is a sub-interval of interval 2

Screenshot 3



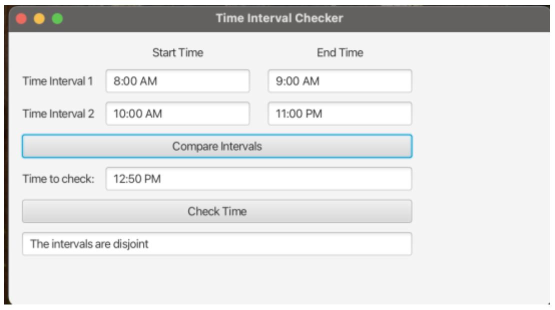
Time Interval 1: 9:00 AM | 2:00 PM

Time Interval 2: 10:00 AM | 12:00 PM

Time to check: 12:50 PM

Interval 2 is a sub-interval of interval 1

Screenshot 4



Time Interval 1: 8:00 AM | 9:00 AM

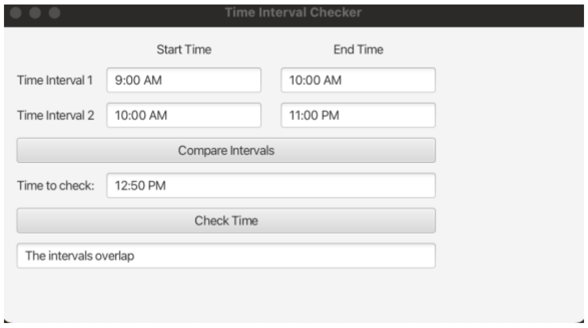
Time Interval 2: 10:00 AM | 11:00 PM

Time to check: 12:50 PM

The intervals are disjoint

Screenshots 5-7

Screenshot 5



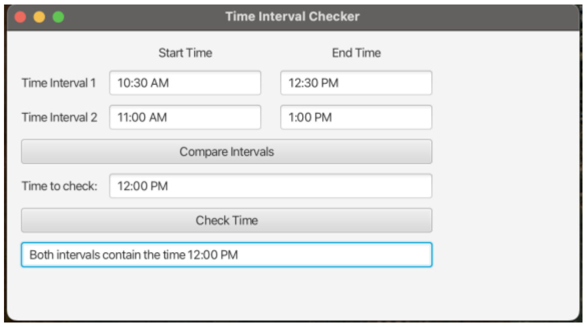
Time Interval 1 9:00 AM 10:00 AM

Time Interval 2 10:00 AM 11:00 PM

Time to check: 12:50 PM

The intervals overlap

Screenshot 6



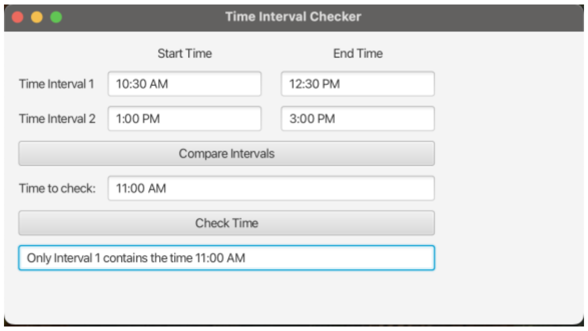
Time Interval 1 10:30 AM 12:30 PM

Time Interval 2 11:00 AM 1:00 PM

Time to check: 12:00 PM

Both intervals contain the time 12:00 PM

Screenshot 7



Time Interval 1 10:30 AM 12:30 PM

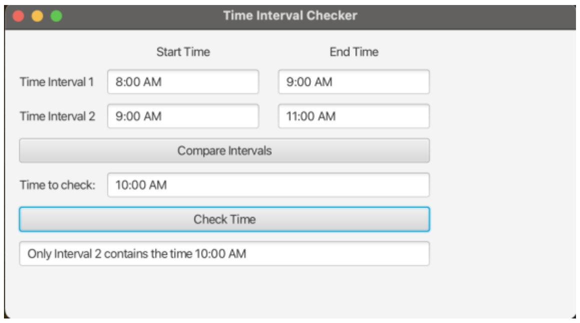
Time Interval 2 1:00 PM 3:00 PM

Time to check: 11:00 AM

Only Interval 1 contains the time 11:00 AM

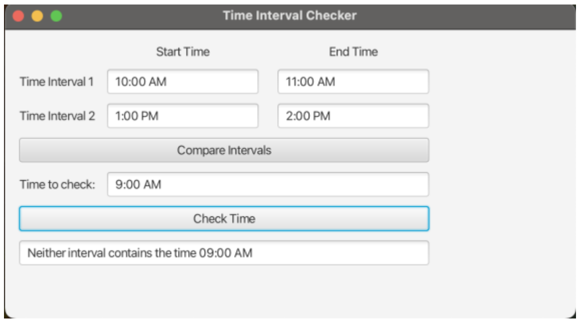
Screenshots 8-10:

Screenshot 8



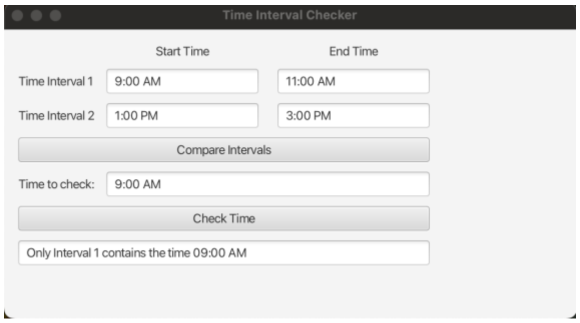
Time Interval 1: 8:00 AM – 9:00 AM
Time Interval 2: 9:00 AM – 11:00 AM
Time to check: 10:00 AM
Only Interval 2 contains the time 10:00 AM

Screenshot 9



Time Interval 1: 10:00 AM – 11:00 AM
Time Interval 2: 1:00 PM – 2:00 PM
Time to check: 9:00 AM
Neither interval contains the time 09:00 AM

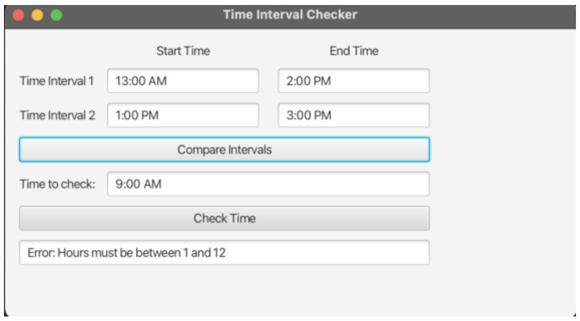
Screenshot 10



Time Interval 1: 9:00 AM – 11:00 AM
Time Interval 2: 1:00 PM – 3:00 PM
Time to check: 9:00 AM
Only Interval 1 contains the time 09:00 AM

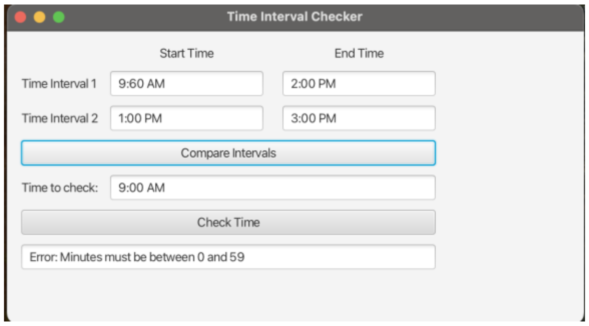
Screenshots 11-13:

Screenshot 11



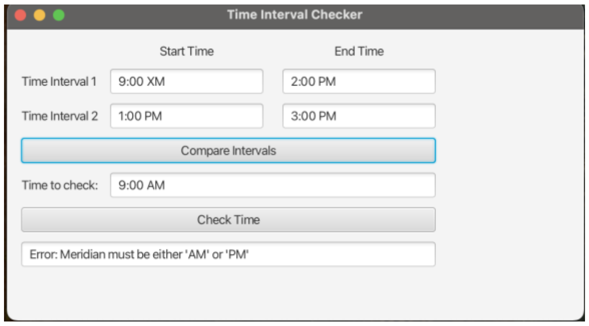
Time Interval 1: 13:00 AM — 2:00 PM
Time Interval 2: 1:00 PM — 3:00 PM
Time to check: 9:00 AM
Error: Hours must be between 1 and 12

Screenshot 12



Time Interval 1: 9:60 AM — 2:00 PM
Time Interval 2: 1:00 PM — 3:00 PM
Time to check: 9:00 AM
Error: Minutes must be between 0 and 59

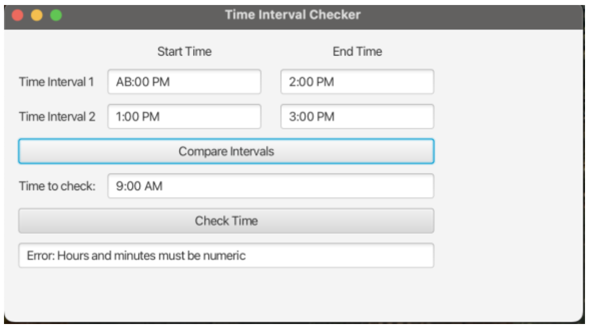
Screenshot 13



Time Interval 1: 9:00 XM — 2:00 PM
Time Interval 2: 1:00 PM — 3:00 PM
Time to check: 9:00 AM
Error: Meridian must be either 'AM' or 'PM'

Screenshots 14-15:

Screenshot 14



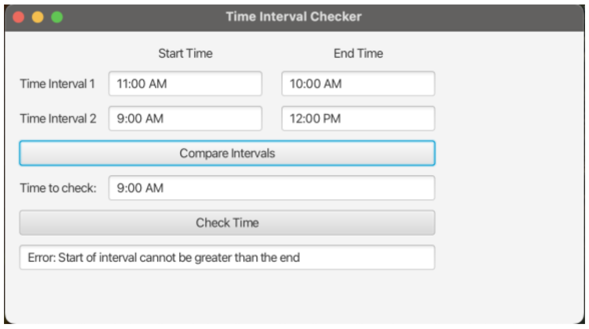
Time Interval 1: AB:00 PM | 2:00 PM

Time Interval 2: 1:00 PM | 3:00 PM

Time to check: 9:00 AM

Error: Hours and minutes must be numeric

Screenshot 15



Time Interval 1: 11:00 AM | 10:00 AM

Time Interval 2: 9:00 AM | 12:00 PM

Time to check: 9:00 AM

Error: Start of interval cannot be greater than the end

Summary of Testing Results

I began by creating a test plan comprising 15 test cases to ensure comprehensive coverage of the program's functionality. These tests included validating operations such as overlapping intervals, subinterval detection in both directions, disjoint intervals, and times that fall inside neither, one, or both intervals. I also tested boundary conditions, such as endpoints that exactly match, to ensure the program correctly treats intervals as closed sets. I also tested all error conditions to ensure they were handled correctly when encountered. I tested for invalid hour values, invalid minute values, non-numeric time inputs, and intervals where the start time comes after the end time. The program correctly handled all errors during testing. I also tested the GUI to ensure that the buttons produce the correct output and that errors thrown by the Time class are correctly caught and displayed in the Project4 interface. Each test was documented with the expected output and verified against the actual production shown in the accompanying screenshots. The results matched expectations, confirming that the Time parsing logic, Interval comparisons, and GUI interactions all functioned as intended. The testing demonstrated that the program handles both correct and incorrect input successfully.

Lessons Learned

This project reinforced the importance of class design and separating functionality across different program components. Implementing the InvalidTime, Time, and Interval classes before building the GUI made the logic easier to verify and reuse. This project also gave me experience working with generics, immutability, and the Comparable interface, which helped me understand how Java allows objects to be ordered and compared. Working with JavaFX reminded me how critical layout planning is for a user-friendly interface. Small details – such as aligning labels, pre-filling fields, and displaying error messages improve the program's usability.

Finally, this project demonstrated the importance of thorough testing. Many edge cases, such as invalid time formats or intervals that touch at endpoints, would not have been evident without deliberately constructing test scenarios. Building a structured test plan helped ensure that the final program was robust and behaved correctly under a wide range of inputs.