Devon Blank
11/21/2025
CMSC 215

# Project 3 Documentation

## Approach

I began by creating a new project in IntelliJ with 2 files per the project instructions:

-Project3.java - contains the **main** method and GUI - JavaFX and Maven - Programmed 2nd

-TripCost.java - contains the TripCost class - Programmed 1st

By following the instructions of the 'Project 3 – Step by step JavaFX Setup Instructions Using IntelliJ – trip cost estimator' I created a new project, chose the JavaFX template, selected Maven as my build system. I then used umgc.assignments as my GroupId and Project3 as my ArtifactId. I skipped adding additional libraries. After the project was generated, I then removed the default hello-view.fxml file and the additional unnecessary controller class. Since the assignment requires a programmatically constructed GUI rather than an FXML-based one. I renamed HelloApplication to Project3 and replaced the contents with the given starter code. I then edited the pom.xml file and updated the module-info.java to properly configure the JavaFX modules, ensuring that the application would build and run without dependency issues.

## TripCost Class Design

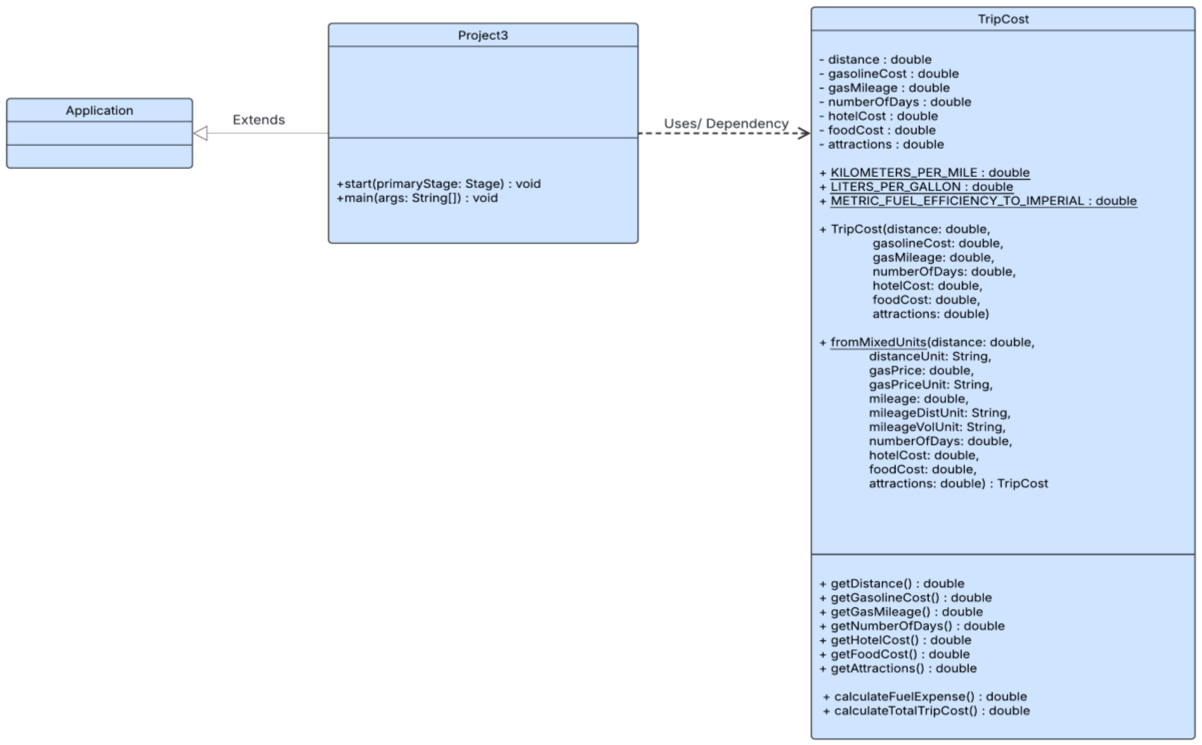
I began by defining the seven fields needed to store user-provided trip parameters. These were declared private final to satisfy the project requirements for immutability. I then included two unit-conversion constants: KILOMETERS_PER_MILE and LITERS_PER_GALLON. I created a 3rd constant for converting metric fuel efficiency to imperial fuel efficiently. This constant is a ratio of the first two. METRIC_FUEL_EFFICIENCY_TO_IMPERIAL = LITERS_PER_GALLON/ KILOMETERS_PER_MILE. Using a calculated constant avoided additional hardcoding conversion values and makes the design cleaner and easier to understand. Next, I implemented a constructor method for TripCost that also handled invalid inputs like 0 or negative numbers for certain inputs. I then wrote a static factory method that takes user inputs from imperial or metric and normalizes them to imperial units for internal usage. The helper then calls the constructor to build the immutable object. Next, I wrote the getter methods for each of the private fields. Finaly, I implemented mathematical equations in two methods, the first is calculateFuelExpense() and the second is calculateTotalTripCost() to handle needed calculations.

## Project3 – Main Program Design

The Project3 class contains the JavaFX user interface and collects input, invokes the TripCost model, and displays the resulting total trip cost. I used a GridPane layout to align all labels, text fields and combo boxes into two columns, matching the example given in the assignment handout. Each text field is pre-populated with the default values provided in the instructions, and the unit-selection combo boxes have all the required options and imperial units default. A “Calculate” button triggers an event handler that reads all user inputs, handles invalid values with exception catching, determines the correct unit types, and then calls TripCost.fromMixedUnits() to build the model. The program then displays the computed total trip cost in an un-editable text field. To improve usability, I also programmed the calculate button to ‘fire’ or automatically calculate the default example values when the program starts, so the window opens with a complete example. This improves on the original version that showed example input but a blank output.

**UML**

Figure 1. UML diagram illustrating relationship between Project3 and TripCost.


Application
Extends
Project3
+start(primaryStage: Stage) : void
+main(args: String[]) : void
Uses/ Dependency
TripCost
- distance : double
- gasolineCost : double
- gasMileage : double
- numberOfDays : double
- hotelCost : double
- foodCost : double
- attractions : double
+ KILOMETERS_PER_MILE : double
+ LITERS_PER_GALLON : double
+ METRIC_FUEL_EFFICIENCY_TO_IMPERIAL : double
+ TripCost(distance: double, gasolineCost: double, gasMileage: double, numberOfDays: double, hotelCost: double, foodCost: double, attractions: double)
+ fromMixedUnits(distance: double, distanceUnit: String, gasPrice: double, gasPriceUnit: String, mileage: double, mileageDistUnit: String, mileageVolUnit: String, numberOfDays: double, hotelCost: double, foodCost: double, attractions: double) : TripCost
+ getDistance() : double
+ getGasolineCost() : double
+ getGasMileage() : double
+ getNumberOfDays() : double
+ getHotelCost() : double
+ getFoodCost() : double
+ getAttractions() : double
+ calculateFuelExpense() : double
+ calculateTotalTripCost() : double

The Unified Modeling Language diagram above outlines the driver class Project3, which depends on the TripCost model class. Project3 extends the JavaFX Application class(white triangle), indicating it is a specialized type of Application, and it has a dependency relationship with TripCost (dashed arrow), meaning the GUI in Project 3 uses TripCost objects to perform unit conversion and trip-cost calculations.

**<u>Test Plan</u>**

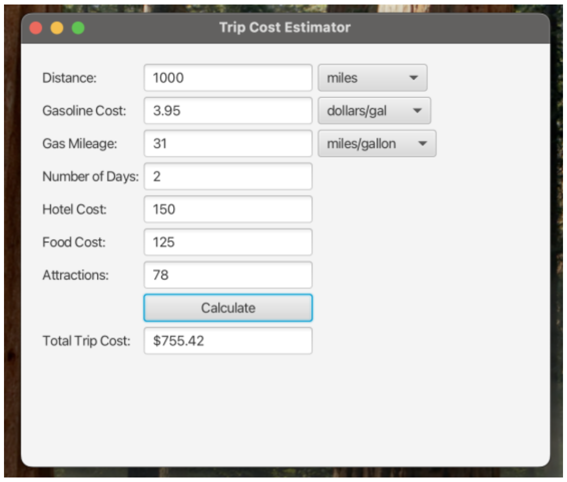
This test plan validates all essential functionality of the Trip Cost Estimator program, (Project 3) including unit conversion handling, numeric unput processing, GUI interaction, error detection, and calculation accuracy. Tests were executed incrementally to verify the correctness of the TripCost model in isolation, followed by integration testing with the Project3 JavaFX interface. Edge case, invalid inputs, and mixed-unit combinations were included to ensure the program behaves reliably under all required conditions.

| Test | Purpose | User input | Expected output | Actual output | Pass / Fail |
|---|---|---|---|---|---|
| 1.All imperial inputs | Verify program functions properly | Screenshot1 | 755.42 | Screenshot 1 | Pass |
| 2.All metric inputs | Verify km, dollars/liter and km/l all work correctly | Screenshot2 | 899.21 | Screenshot 2 | Pass |
| 3. Mixed units 1 | Ensure program functions with mixed imperial and metric units – miles, dollars/ liter and km/l | Screenshot3 | 1013.01 | Screenshot 3 | Pass |
| 4. Mixed units 2 | Ensure program functions with mixed imperial and metric units – km, dollars/ gallon and miles/gal | Screenshot4 | 854.56 | Screenshot 4 | Pass |
| 5. Long road trip | Ensure large distance values function correctly | Screenshot5 | 4300.00 | Screenshot 5 | Pass |
| 6.Very efficient car | Ensure highly efficient cars create reasonable output | Screenshot 6 | 122.00 | Screenshot 6 | Pass |
| 7.No hotel cost | Ensure hotel cost of 0 (day trip) function as intended | Screenshot 7 | 92.50 | Screenshot 7 | Pass |
| 8. Zero distance | Confirms that zero distance trips function | Screenshot 8 | 510.00 | Screenshot 8 | Pass |
| 9.Very long trip | Ensure long trip times function correctly | Screenshot 9 | 188,750.00 | Screenshot 9 | Pass |

| | | | | | |
|---|---|---|---|---|---|
| 10. Negative distance | Ensure program throws error with negative distances | Screenshot 10 | Error: Distance cannot be negative | Screenshot 10 | Pass |
| 11.  Negative gas cost | Ensure program throws error with negative gas cost | Screenshot 11 | Error: Gasoline cost cannot be negative | Screenshot 11 | Pass |
| 12. Zero Gas mileage | Ensure gas divide by 0 error is handled | Screenshot 12 | Error: Gas mileage must be positive | Screenshot 12 | Pass |
| 13. Negative hotel cost | Ensure negative hotel cost error is handled | Screenshot 13 | Error: Hotel cost cannot be negative | Screenshot 13 | Pass |
| 14.Non-numberic distance input | Ensures non-numeric input cause error | Screenshot 14 | Invalid input | Screenshot 14 | Pass |
| 15. Empty gasoline cost field | Ensure error is thrown for empty field | Screenshot 15 | Invalid input | Screenshot 15 | Pass |

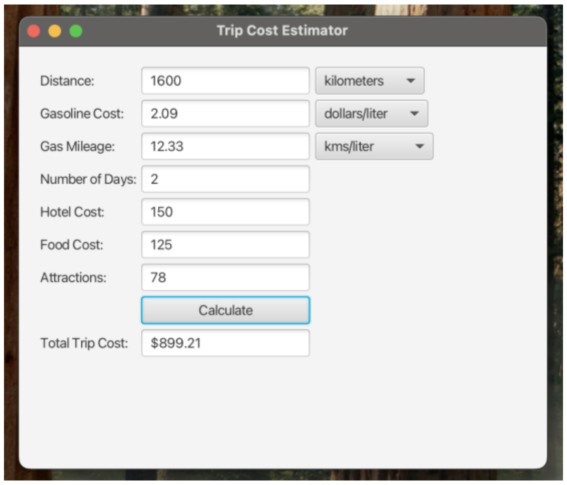
## Screenshots 1-4: Basic Functionality of Units and Mixed Inputs

Screenshot1



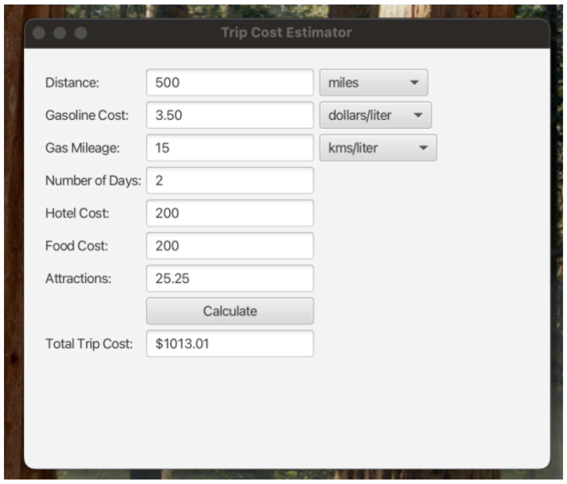
| | | |
|---|---|---|
| Distance: | 1000 | miles |
| Gasoline Cost: | 3.95 | dollars/gal |
| Gas Mileage: | 31 | miles/gallon |
| Number of Days: | 2 | |
| Hotel Cost: | 150 | |
| Food Cost: | 125 | |
| Attractions: | 78 | |
| Total Trip Cost: | $755.42 | |

Screenshot 2



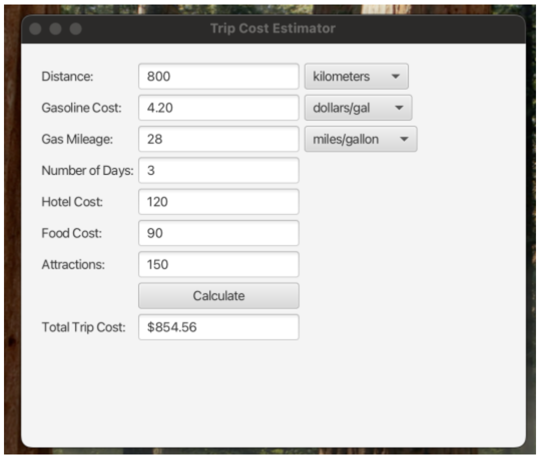
| | | |
|---|---|---|
| Distance: | 1600 | kilometers |
| Gasoline Cost: | 2.09 | dollars/liter |
| Gas Mileage: | 12.33 | kms/liter |
| Number of Days: | 2 | |
| Hotel Cost: | 150 | |
| Food Cost: | 125 | |
| Attractions: | 78 | |
| Total Trip Cost: | $899.21 | |

Screenshot 3



| | | |
|---|---|---|
| Distance: | 500 | miles |
| Gasoline Cost: | 3.50 | dollars/liter |
| Gas Mileage: | 15 | kms/liter |
| Number of Days: | 2 | |
| Hotel Cost: | 200 | |
| Food Cost: | 200 | |
| Attractions: | 25.25 | |
| Total Trip Cost: | $1013.01 | |

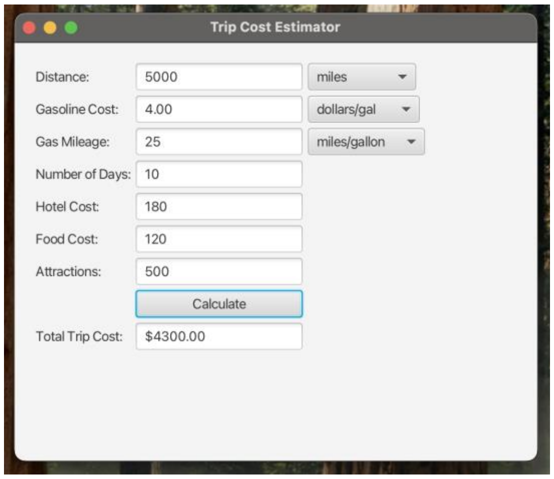
Screenshot 4



| | | |
|---|---|---|
| Distance: | 800 | kilometers |
| Gasoline Cost: | 4.20 | dollars/gal |
| Gas Mileage: | 28 | miles/gallon |
| Number of Days: | 3 | |
| Hotel Cost: | 120 | |
| Food Cost: | 90 | |
| Attractions: | 150 | |
| Total Trip Cost: | $854.56 | |

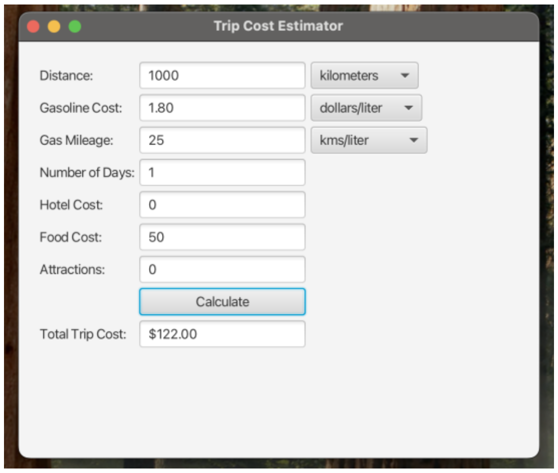
## <u>Screenshots 5-7: Long Trip, Very Efficient Car, No Hotel Cost</u>

Screenshot 5



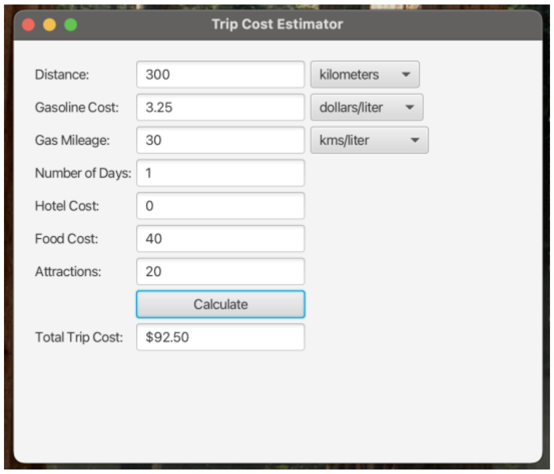
| | | |
|---|---|---|
| Distance: | 5000 | miles |
| Gasoline Cost: | 4.00 | dollars/gal |
| Gas Mileage: | 25 | miles/gallon |
| Number of Days: | 10 | |
| Hotel Cost: | 180 | |
| Food Cost: | 120 | |
| Attractions: | 500 | |
| Total Trip Cost: | $4300.00 | |

Screenshot 6



| | | |
|---|---|---|
| Distance: | 1000 | kilometers |
| Gasoline Cost: | 1.80 | dollars/liter |
| Gas Mileage: | 25 | kms/liter |
| Number of Days: | 1 | |
| Hotel Cost: | 0 | |
| Food Cost: | 50 | |
| Attractions: | 0 | |
| Total Trip Cost: | $122.00 | |

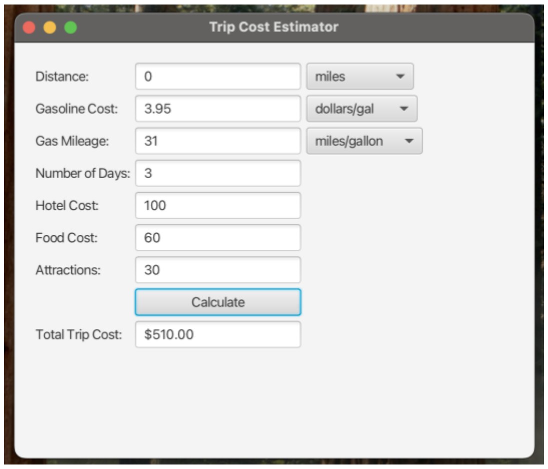
Screenshot 7



Distance: 300 kilometers
Gasoline Cost: 3.25 dollars/liter
Gas Mileage: 30 kms/liter
Number of Days: 1
Hotel Cost: 0
Food Cost: 40
Attractions: 20
Calculate
Total Trip Cost: $92.50

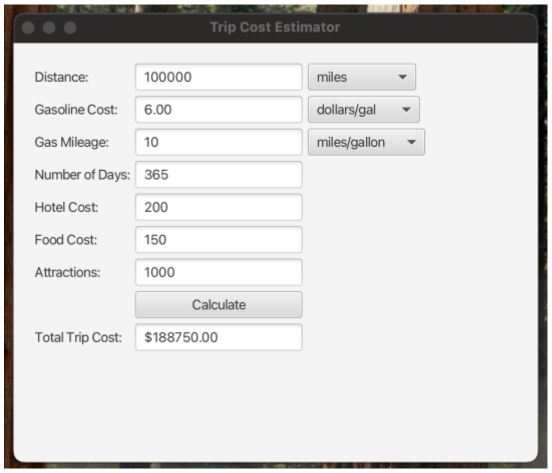
<u>**Screenshots 8-10: Zero distance, long trips, negative distance**</u>

Screenshot 8



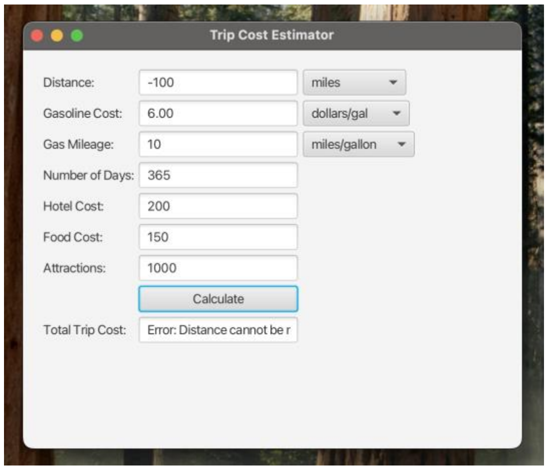
Distance: 0 miles
Gasoline Cost: 3.95 dollars/gal
Gas Mileage: 31 miles/gallon
Number of Days: 3
Hotel Cost: 100
Food Cost: 60
Attractions: 30
Calculate
Total Trip Cost: $510.00

Screenshot 9



Distance: 100000 miles
Gasoline Cost: 6.00 dollars/gal
Gas Mileage: 10 miles/gallon
Number of Days: 365
Hotel Cost: 200
Food Cost: 150
Attractions: 1000
Calculate
Total Trip Cost: $188750.00

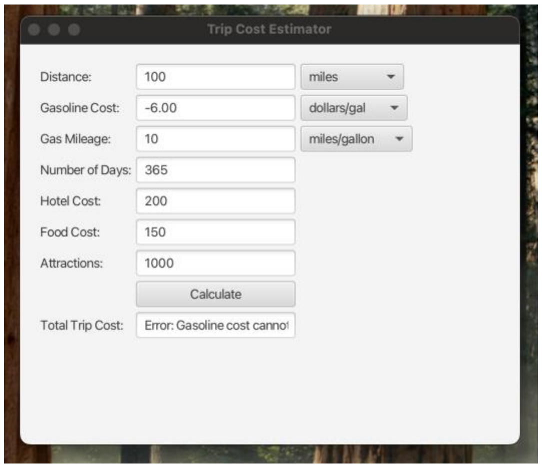
Screenshot 10


Trip Cost Estimator

Distance: -100 miles
Gasoline Cost: 6.00 dollars/gal
Gas Mileage: 10 miles/gallon
Number of Days: 365
Hotel Cost: 200
Food Cost: 150
Attractions: 1000
Calculate
Total Trip Cost: Error: Distance cannot be r

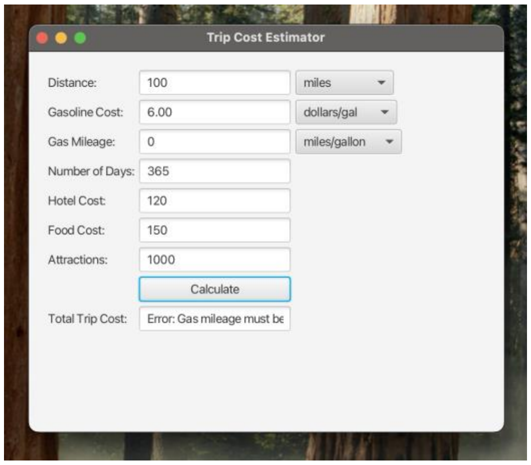
## Screenshots 11-13: Negative and Zero Input Error Handling

Screenshot 11


Trip Cost Estimator

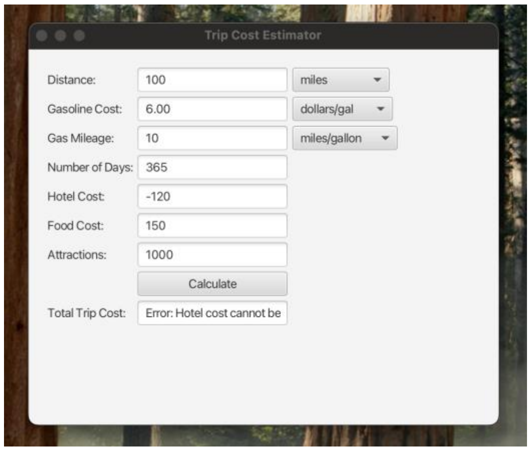
Distance: 100 miles
Gasoline Cost: -6.00 dollars/gal
Gas Mileage: 10 miles/gallon
Number of Days: 365
Hotel Cost: 200
Food Cost: 150
Attractions: 1000
Calculate
Total Trip Cost: Error: Gasoline cost cannot

Screenshot 12


Trip Cost Estimator

Distance: 100 miles
Gasoline Cost: 6.00 dollars/gal
Gas Mileage: 0 miles/gallon
Number of Days: 365
Hotel Cost: 120
Food Cost: 150
Attractions: 1000
Calculate
Total Trip Cost: Error: Gas mileage must be

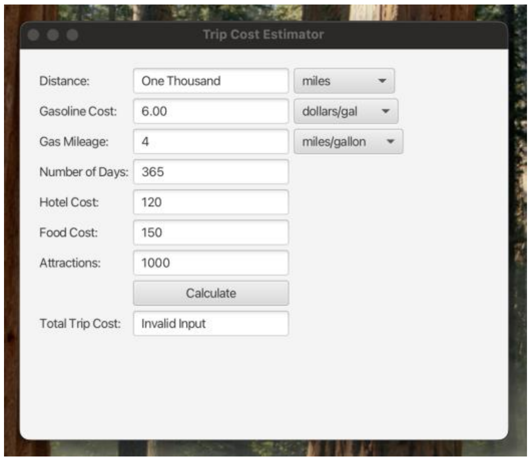
Screenshot 13



Distance: 100 miles
Gasoline Cost: 6.00 dollars/gal
Gas Mileage: 10 miles/gallon
Number of Days: 365
Hotel Cost: -120
Food Cost: 150
Attractions: 1000
Total Trip Cost: Error: Hotel cost cannot be

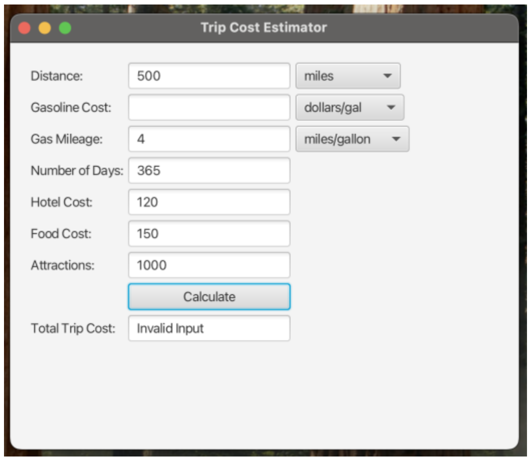
## Screenshots 14-15: Non-numeric and empty input

Screenshot 14



Distance: One Thousand miles
Gasoline Cost: 6.00 dollars/gal
Gas Mileage: 4 miles/gallon
Number of Days: 365
Hotel Cost: 120
Food Cost: 150
Attractions: 1000
Total Trip Cost: Invalid Input

Screenshot 15



Distance: 500 miles
Gasoline Cost: dollars/gal
Gas Mileage: 4 miles/gallon
Number of Days: 365
Hotel Cost: 120
Food Cost: 150
Attractions: 1000
Total Trip Cost: Invalid Input

## Summary of Testing Results

All fifteen test cases were executed to verify the correctness and reliability of the Trip Cost Estimator application. The tests covered normal usage, extreme values, invalid inputs, imperial units, metric units and mixed units. The program produced accurate results across all valid cases. This confirmed that the formulas for fuel cost and total trip cost were implemented correctly and that the internal unit conversion performed by the fromMixedUnits method worked consistently and correctly.

The graphical user interface also behaved as expected. The calculate button correctly triggered the computation, the output field remained non-editable per project requirements, and the program successfully auto computed the default example on startup. Error handling responded correctly in all negative or invalid input scenarios. Non-numeric input cases had a clear 'invalid input' message, illegal values (such as negative distance of zero gas mileage) triggered the TripCost class's validation logic and displayed the correct error messaged in the GUI. No unexpected crashes or incorrect outputs were encountered at any point during testing.

## Lessons Learned

This project provided valuable experience in applying object-oriented design principles to a real application. Separating the program into a GUI class (Project3) and a dedicated model class (TripCost) made the code cleaner and easier to test. Implementing the TripCost class as immutable reinforced the benefits of creating objects that cannot be altered after construction, reducing potential bugs and making the behaviors of the program more predictable.

This project also deepened my understanding of JavaFX layouts, event handling and user-interface structure. Designing the GridPane required attention to spacing, alignment, and consistent control sizing so that layout matched the example in the assignment. Additionally, catching errors in the GUI and validating data within the TripCost constructor demonstrated the importance of well thought out programming. By preventing illegal states and providing immediate feedback to the user, the application became more robust and user-friendly.

Overall, this project strengthened my skills in Java programming, unit conversion logic, GUI design, and software documentation. It showed how thoughtful organization, clear validation rules, and the modular design led to a reliable and maintainable application.