# Taxi Fare Calculator

This project is a taxi fare calculator implemented in Go. It processes records of distances traveled and calculates the total fare based on specified rates. The fare is calculated with different rates for different distance ranges.

## Table of Contents



## Requirement

1. Given the input data

```
00:00:00.000 0.0
00:01:00.123 480.9
00:02:00.125 1141.2
00:03:00.100 1800.8
```

2. Workflow

- Processing line: `00:00:00.000 0.0`

```
Step 1: Initial fare: 400 yen for up to 1 km.
```

- Processing line: `00:01:00.123 480.9`

```
Step 2: Current Distance: 480.9 meters
Still within the first 1 km, no additional fare. Fare remains: 400 yen
```

- Processing line: `00:02:00.125 1141.2`

```
Step 3: Current Distance: 1141.2 meters
Additional distance beyond 1 km: 141.2 meters
Number of 400m units: 0.35
```

```
  Additional fare: 14.12 yen (0.35 * 40.00)
  Total fare after this step: 414 yen
```

- Processing line: `00:03:00.100 1800.8`

```
  Step 4: Current Distance: 1800.8 meters
  Additional distance beyond 1 km: 659.6 meters
  Number of 400m units: 1.65
  Additional fare: 65.96 yen (1.65 * 40.00)
  Total fare after this step: 480 yen
```

- Total Fare: 480 yen

## Project Structure

```
├── Dockerfile              # Dockerfile to containerize the application
├── README.md               # This README file
├── go.mod                  # Go module file
├── go.sum                  # Go module dependencies
├── main.go                 # Main entry point of the application
├── main_test.go            # Unit tests for the main entry point
├── input.txt               # Sample input file for testing
├── docker-compose.yml      # Docker Compose configuration file
├── meter                   # Package for handling taxi meter logic
│   ├── meter.go
│   └── meter_test.go       # Unit tests for the meter package
├── record                  # Package for handling record parsing and data
structure
│   ├── record.go
│   └── record_test.go      # Unit tests for the record package
└── utils                   # Utility package for logging and other helper
functions
    ├── log.go              # Logging utility functions
    └── log_test.go         # Unit tests for the logging utility
```

## Prerequisites

Before running this project, make sure you have the following installed:

- Go (version 1.20 or higher)
- Docker (optional, if you want to run the application in a Docker container)

## Installation

1. Clone the repository (or extract zip file) to your local machine

```
git clone https://github.com/kemul/taxi-fare.git
cd taxi-fare
```

2. Install the Go modules:

```
go mod tidy
```

# Running the Application

## With Docker

In the root application

1. Build the Docker image:

```
docker build -t taxi-fare-app .
```

2. Run the application inside the Docker container:

```
docker run --rm taxi-fare-app
```

3. The application will output the calculated fare and the sorted records inside the container.

To add Docker Compose documentation to your README, you can include the following section:

---

## Or With Docker Compose

You can build and run the application using Docker Compose. Follow these steps in the root:

1. **Build and Run the Application**:

```
docker-compose up --build
```

2. **Run in Detached Mode**:

```
docker-compose up --build -d
```

3. **Stop and Remove Containers**:

```
docker-compose down
```

## Or With Go(lang) command

1. Navigate to project directory

```
cd taxi-fare
```

2. Tidy Up Go Modules

```
go mod tidy
```

3. Tidy Up Go Modules

```
go run main.go
```

## Output

```
PS E:\Workspace\taxi-fare> docker run --rm taxi-fare-app
2024/08/26 14:34:28 Process Calculation
================================================
2024/08/26 14:34:28 Processing Input: {0000-01-01 00:00:00 +0000 UTC 0 0}
2024/08/26 14:34:28 Step 1: Initial fare: 400 yen for up to 1 km.
2024/08/26 14:34:28 Process Calculation
================================================
2024/08/26 14:34:28 Processing Input: {0000-01-01 00:01:00.123 +0000 UTC 480.9
480.9}
2024/08/26 14:34:28 Step 2: Current Distance: 480.9 meters
2024/08/26 14:34:28 Still within the first 1 km, no additional fare. Fare remains:
400 yen
2024/08/26 14:34:28 Process Calculation
================================================
2024/08/26 14:34:28 Processing Input: {0000-01-01 00:02:00.125 +0000 UTC 1141.2
660.3000000000001}
2024/08/26 14:34:28 Step 3: Current Distance: 1141.2 meters
2024/08/26 14:34:28 Additional distance beyond 1 km: 141.2 meters
2024/08/26 14:34:28 Number of 400m units: 0.35
2024/08/26 14:34:28 Additional fare: 14.12 yen (0.35 * 40.00)
2024/08/26 14:34:28 Total fare after this step: 414 yen
2024/08/26 14:34:28 Process Calculation
================================================
2024/08/26 14:34:28 Processing Input: {0000-01-01 00:03:00.1 +0000 UTC 1800.8
659.5999999999999}
2024/08/26 14:34:28 Step 4: Current Distance: 1800.8 meters
```

```
2024/08/26 14:34:28 Additional distance beyond 1 km: 659.6 meters
2024/08/26 14:34:28 Number of 400m units: 1.65
2024/08/26 14:34:28 Additional fare: 65.96 yen (1.65 * 40.00)
2024/08/26 14:34:28 Total fare after this step: 480 yen
{"event":"fare_calculation","fare":480.08,"level":"info","msg":"Calculated fare
successfully","time":"2024-08-26T15:02:31Z"}
(iv) Output ============================
480
00:02:00.125 1141.2 660.3
00:03:00.100 1800.8 659.6
00:01:00.123 480.9 480.9
00:00:00.000 0.0 0.0
(iv) JSON Output ============================
{"event":"output","fare":480,"level":"info","msg":"(iv) Output Json","time":"2024-
08-26T15:02:31Z"}
{"diff":660.3000000000001,"distance":1141.2,"fields.time":"00:02:00.125","level":"
info","msg":"Processed record","time":"2024-08-26T15:02:31Z"}
{"diff":659.5999999999999,"distance":1800.8,"fields.time":"00:03:00.100","level":"
info","msg":"Processed record","time":"2024-08-26T15:02:31Z"}
{"diff":480.9,"distance":480.9,"fields.time":"00:01:00.123","level":"info","msg":"
Processed record","time":"2024-08-26T15:02:31Z"}
{"diff":0,"distance":0,"fields.time":"00:00:00.000","level":"info","msg":"Processe
d record","time":"2024-08-26T15:02:31Z"}
```

## Testing

1. Input file exist in the root project, with file name `input.txt`

```
├── input.txt               # Sample input file for testing
```

2. To run the unit tests for the project, use the following command:

```
go test ./... -coverprofile=coverage
```

Result Test Coverage

```
PS E:\Workspace\taxi-fare> go test ./... -coverprofile=coverage
ok      taxi-fare       0.424s  coverage: 77.8% of statements
ok      taxi-fare/meter 0.424s  coverage: 79.5% of statements
ok      taxi-fare/record        0.413s  coverage: 100.0% of statements
ok      taxi-fare/utils 0.413s  coverage: 100.0% of statements
```

2. To Preview Code Coverage

```
go tool cover -html=coverage
```

Example Output : https://github.com/kemul/taxi-fare/blob/main/coverage.html