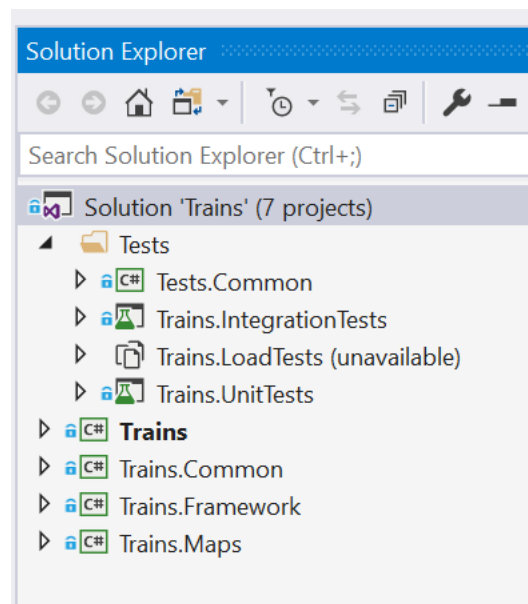


## Contents

Solution Structure .....	1
Running the application .....	2
Design Explanation.....	3
Entities .....	3
Services .....	3
Business Rules .....	3

## Solution Structure

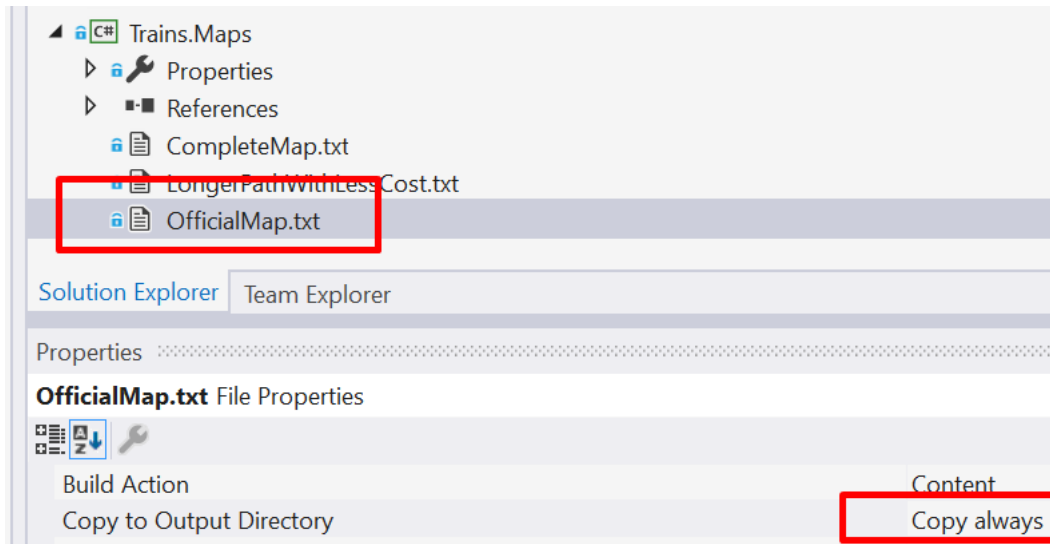
Application is written in C# and you would need Visual Studio to compile and then run it. Solution contains following 8 projects, purpose of each of them is described below:



- **Trains** – This is a console application, you need to build this project only in order to run the application.
- **Trains.Framework**- This project contains all the classes and the business logic
- **Trains.Common** – This project contains common code that is needed across all other project, like extension methods.
- **Trains.Maps** – This contains a few sample map text files. When you build console application or unit testing projects, all of these files get copied to the relevant bin folder. If you want to add more maps to the application for testing purposes, add them here and then build the solution. Make sure you chose 'Copy always' or 'Copy if newer' option for 'Copy to output directory' in properties window as shown in image below. 'OfficialMap.txt' file has got the map that was provided by ThoughtWorks.
- **Trains.UnitTests** – Contains unit tests against classes.

- **Trains.IntegrationTests** – Contains integrations tests for different parts of the system.
- **Trains.LoadTests** – Contains a few load tests to test the performance and system failure points. I have unloaded this project. Loading this project will add a couple of minutes to the unit tests so please **only load this project** if you want to do load testing.
- **Tests.Common** – Contains code that supplies in-memory maps to other testing projects and

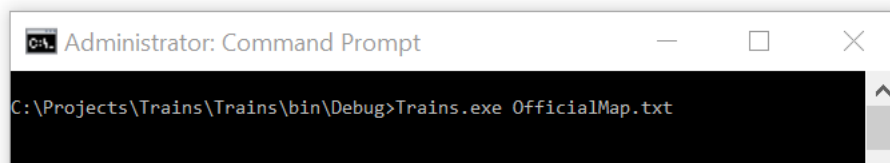
an attribute class that help testing exception message.



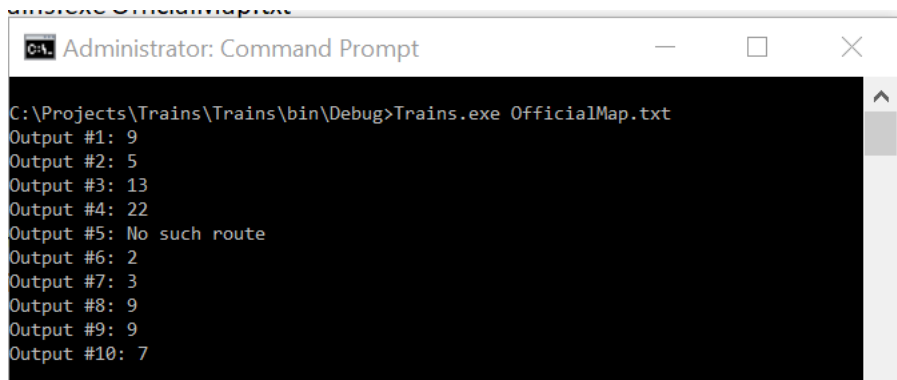
## Running the application

To run the application, do following:

1. Build 'Trains' project. This will create Console Application on the hard disk.
2. Open command prompt go to the 'bin/debug' folder under Trains.
3. Copy following command in the command prompt  
Trains.exe OfficialMap.txt



4. Hit Enter and if there is a 'OfficialMap.txt' file in 'bin/debug' folder, it will show result like this:



```
Administrator: Command Prompt
C:\Projects\Trains\Trains\bin\Debug>Trains.exe OfficialMap.txt
Output #1: 9
Output #2: 5
Output #3: 13
Output #4: 22
Output #5: No such route
Output #6: 2
Output #7: 3
Output #8: 9
Output #9: 9
Output #10: 7
```

## Design Explanation

Classes are mainly categorised in following three areas:

### 1. Entities

These map to real life business objects and are extracted from business domain.

**Map** - Contains list of all towns in the map and those towns then contain link between them

**Town** – Represents a town in real word, has got its name and list of neighbouring town that are directly connected to it

**Route** - represents a direct link between two towns

**TravelCard** – Maintains travel history across one specific route like how much distance is covered so far and how many stops are visited

### 2. Services

These classes perform different calculations and analysis over entities to compute the result.

Purpose of some of them is below:

**DistanceCalculator** - Calculates the distance along a specific path.

**ShortestPathFinder** - Calculates the shortest path between two towns by maintaining minimum distance from source against each town. This algorithm is inspired by Dijkstra's algorithm. It maintains a list of every town's shortest distance from the source – initially set to a very large number. If coming from another route the distance is less than previously calculated distance, it will be updated. If distance is greater or equal, that path will simply be ignored.

**RouteFinder** - This class finds all the routes in the map using depth first approach.

**JourneyPlanner** – This is the core class that is kind of orchestrator in the system. It takes in other service classes and map as parameter, performing calculations and then returns the result

### 3. Business Rules

This business rules are passed to services to make some decision making dynamic. For example the rule for how deep in the hierarchy system should go is provided from the caller of the service.