1 Restrictions

The following restrictions apply to this assignment:
• You must not use lambdas and streams

The use of any third party libraries, except JUnit5, is not permitted

2 Introduction

StarTravel is a small travel agency trying hard to make a comeback now that restrictions have eased and trips are allowed again. They need to deploy their new travel planning system by the end of the week and have contracted you to help finish the project on time. Luckily, you are not alone - your teammates are Anantha the data engineer, Kimame the designer, Alina the software engineer, Luke the product manager, and Eleonora the engineering manager.

3 Project tasks

Eleonora has split up the project into separate tasks. Anantha will be taking on the tasks of obtaining schedule data for planes, buses and trains. Alina will be working on the user experience of the website and project integration. Your task will be to develop the back-end processing for the data from Anantha, so that it will be ready for Alina's components to use. Your tasks have been prioritised as "basic", "intermediate", and "advanced". Eleonora stressed that it is key to get the basic tasks done well before moving on to intermediate ones, and the same applies to intermediate and advanced ones.

3.1 Basic tasks

The highest priority work items assigned to you from the team project board are:

Implement transport classes

Alina has written a developer specification detailing that back-end code must include the following classes for modes of transport: `Plane`, `Bus`, `Taxi`, `Train`, and `Walk`. These all represent a single journey from point A to point B using that mode of transport. Planes, buses, and trains are scheduled modes of transport that go from departure point A to destination point B at specific times according to their schedules. Taxis and walking can be used to travel at anytime. They all need to:

- include journey departure and arrival places, dates, and times (as `String`s, wrapped in the provided `PlaceAndMoment` class)
- include journey distance (as an `int` — the provided values will always be rounded to the nearest km)
- include journey cost (as a `double` in British pounds)
- have a method to calculate carbon emissions per passenger for a given distance (as a `double` in kg of effective carbon dioxide emissions according to the following formulae:

Mode of transport	Carbon emissions (kg CO2e)
Plane	$distance * 92.2452/574 * 2$
Train	$distance * 27.9218/632 * 2$
Bus	$distance * 21.7060/642 * 2$
Taxi	$distance * 83.8332/642$
Walk	0

  (if you are curious where these formulae come from, see the appendix subsection 6.2. Carbon Emissions, but this is not relevant for completing the tasks)

- override the built-in `toString()` method to produce output in the following format (for some examples, see `TEST_OUTPUT_BASIC` in the `TransportData.java` file):

```
[date(s) of travel DD/MM/YYYY(-DD/MM/YYYY)]: [transport] from [departure point]
([departure time HH:MM]) to [destination point] ([arrival time HH:MM]), [price with
fixed 2 decimal places]GBP, [carbon emissions with fixed 4 decimal places]kg CO2
```

Tips from Alina: make use of inheritance to avoid code duplication and ensure conformance to the specification. If you are having trouble importing JUnit5 for the unit tests to compile, you can just delete the `test` *folder, it is not required.*

Implement a main class and add a confidence check with test data

Anantha has sent you some test data and expected output that you can use to test your class implementations. They are included in the project, see the `TransportData.java` file in the `data` package.

The project needs to have a main class called `TravelPlanner` with an entry point main method.
If `java app.TravelPlanner` is run without any program arguments, the program should perform a confidence check. This can be done by creating objects of each transport class with imported test data from the file provided by Anantha and comparing their `toString()` method output to the corresponding provided test output. If all the output matches, it should print "Confidence check passed!" to `stdout`, otherwise, it should print "Confidence check failed!" and terminate with exit code 1.

For this task, you need to only implement a confidence check using the `TEST_DATA_BASIC` dataset.

Write a short (300-500 word) report about your design decisions

To help Anantha and Alina quickly understand your code, add a description about where, how, and why you used inheritance in the project.

3.2 Intermediate tasks

Once all the basic tasks are done, the next tasks from the team project board are:

Extend the confidence check

Extend the confidence check to also use the provided TEST_DATA_INTERMEDIATE dataset. This extended data set includes data rows for walking and taxis, which do not have a specific start or end date and time.
Alina has come up with a workaround for this: you can set the departure and arrival dates and times to null for new Walk and Taxi objects. When these fields are null, the toString() override should adapt to display in this format:

```
flexible ([travel time H:MM]): [Transport] from [departure point] to [arrival point],
[price with fixed 2 decimal places]GBP, [carbon emissions with fixed 4 decimal
places]kg CO2
```

Eventually these objects will need to be tied to concrete departure and arrival moments. If departure and arrival dates/times are specified, then the toString() output should conform to the general rule from the basic tasks.

Read and parse location, schedule, and price data

The program needs to be able to dynamically load and process data about available transport connections between locations. For simplicity, we can assume all prices are fixed and never change.

Anantha's work-in-progress data scrapers have produced an aggregated-transport-data.txt file for testing, however he added a warning when sending the file — some errors have likely gotten mixed in the data and your program needs to be able to handle them. Eleonora has made the executive decision, that if a row of data is in invalid format, the program should silently ignore that whole row (this includes rows with fully valid data that have an unexpected character at the end and similar. Whether this is good design or not remains a question for another day).

The provided aggregated-transport-data.txt file has a similar structure to CSV (Comma Separated Values) files. Every line in the file is a row of data, and columns are separated by commas. The difference is that the columns are named key-value pairs and could be in any order.
For example, mode=plane,distance=355 and distance=355,mode=plane are equivalent rows of data.
Data files can have comments. Every line starting with # is a comment and should be ignored when loading data.
For planes, trains, and buses, the data format is:

```
mode=plane/train/bus, departurePlace=city1, departureDate=DD/MM/YYYY,
departureTime=HH:MM, arrivalPlace=city2, arrivalDate=DD/MM/YYYY, arrivalTime=HH:MM,
price=x.x, distance=x
```

For taxis, the data format is very similar, but dates and times are flexible:

```
mode=taxi, departurePlace=city1, arrivalPlace=city2, travelTime=H:MM, price=x.x,
distance=x
```

And walking does not include any cost (we assume the user can walk 24 hours a day and that no costs are involved. Tough!):

```
mode=walk, departurePlace=city1, arrivalPlace=city2, travelTime=H:MM, distance=x
```

If something takes less than 1 hour, e.g. 25 minutes, the data will be travelTime=0:25. For simplicity, we will not use days, or any other units, only hours and minutes. If it takes more than 1 hour, e.g. 65 minutes, the data will be travelTime=1:05, minutes never go over 60. Hours are unlimited (within int range).

Introduce new program argument for configurable data loading

To complete the updated program that can handle runtime data loading, it needs to be configurable. The main class will need to support a new program argument: `--load` followed by all the data files that should be loaded **instead of** running a confidence check with test data.
When data is loaded, print out every parsed row to `stdout` (the `verbose` parameter will be set to `true` to indicate that output should be printed).
The order of the provided files to load should only influence the order that data is displayed, but not affect what data is loaded. This means that running `java app.TravelPlanner --load aggregated-transport-data.txt some-more-data.txt` should first load `aggregated-transport-data.txt` and then `some-more-data.txt`, but overall the whole loaded dataset should be the same as when running `java app.TravelPlanner --load some-more-data.txt aggregated-transport-data.txt`.
If any files provided cannot be found or there is an error with opening the file / reading its contents, this should output an error message to `stderr`: `Could not open "[filename.txt]"!`, but the program should continue running and attempt to load any other provided data files.

Write a short (300-500 word) report about your design decisions

Anantha is interested in seeing how the data loading of the the files he produced works. Write a short report for Anantha, explaining how you implemented the intermediate tasks, and what design decisions you made. You can include details even if it didn't fully work, but you attempted the tasks.

3.3 Advanced tasks

Implement a Journey class to encapsulate multiple transport links and extend the confidence check

The `Journey` class should take in an array of transport links and calculate the total price, duration, waiting time, and CO2 emissions for the whole trip.
All the legs of a journey will need to be sorted in chronological order. The `Moment` class provides a `compareTo` method that will help to do this. For `Moments` A and B, `A.compareTo(B)` will return < 0 if A comes before B, 0 if they are equal, and > 0 if B comes
`Journey` objects should also override the `toString()` method and produce output in the following format:

```
[date(s) of travel DD/MM/YYYY(-DD/MM/YYYY)]: [comma separated modes of transport] from
[departure point] ([departure time HH:MM]) to [final destination point] ([final arrival
time HH:MM]), with [number of changes except walking] changes and [distance]km walking,
[total price with fixed 2 decimal places]GBP, [total carbon emissions with fixed 4
decimal places]kg CO2
```

Some examples are included in the `TEST_DATA_ADVANCED` and `TEST_OUTPUT_ADVANCED` fields of `TransportData`. The program confidence check should be extended to include these too.

3.3.1 User Requirements

Luke has been conducting surveys and interviews with potential customers. Having identified students as ScotTravel's ideal target audience, he summarised the most common customer archetypes as 4 fictional personas:

Hai-Jun This year I started a bachelor's degree in computer science. When applying to study here, I hoped to see many of the famous Scottish sights, but due to Covid restrictions, I haven't been able to even come to Scotland! I have been studying remotely from home, but now that things are opening up, I could finally come to Edinburgh. My first goal is to see the ancient universities in Edinburgh, Glasgow, and St. Andrews, but the trains there are too expensive for my budget... I would be willing to take slower connections, if I could save money this way.	**Shivam** I am a PhD student in veterinary studies at the Easter Bush campus. I studied here for my bachelor's and master's degrees, and have seen many things around, but Loch Lomond is still on my bucket list. I would love to go, but I am concerned about the CO2 impact of travel. Hiking there would be ideal (I usually walk everywhere in the city), but Loch Lomond is a little too far even for an experienced hiker like me. It would be great if it was possible to compare different travel options by their carbon emissions, so I could pick an environmentally friendly journey plan.	**Tudor** I have just arrived to study my master's degree in English literature in Edinburgh, and would love to explore the country! I have already seen most of the local attractions and now want to go visit other cities, like Glasgow, Aberdeen, and Inverness. However, due to a childhood injury, it is difficult for me to walk far. Whenever I travel, I need to plan everything in advance and take public transport or a taxi whenever possible.	**Anastasia** I'm in my fourth undergraduate year of mechanical engineering studies. The years passed by very quickly while studying hard for my degree and now that it is my last year, I realised I haven't visited the Isle of Skye yet and this may be my last chance. I always wanted to go, but there are no direct journeys and it is difficult to find trips without long layovers. Minimising the total travel time is very important to me, because I can only afford to go away for a few days without studying.

These personas form the basis for implementing heuristics in the next task of finding the best journey for a particular user.

Implement algorithms to find the best Journey for a user

The program should be able to find the best journey for a particular user, when run with the `--find` flag. `--find` will be followed by the user desired departure date (`DD/MM/YYYY`), time (`HH:MM`), departure place, arrival place, and one of 4 possible predefined heuristics: *cheapest*, *shortest*, *greenest*, *leastWalking*.

- *cheapest* is the journey that costs the least
- *shortest* is the journey with the earliest arrival date and time
- *greenest* is the journey with the least carbon emissions
- *leastWalking* is the journey with the least walking distance

The `--find` flag will always come after `--load`, if provided. When the `--find` flag is provided, the program should not output anything when data is being loaded (and `verbose` will be set to `false`), but instead, print out only the one best journey according to the chosen heuristic.

For example: `java app.TravelPlanner --find 01/09/2021 08:00 Edinburgh Glasgow cheapest` should display only one journey, the one that has the lowest total price.

Because ScotTravel are only developing the first stage of the program, Eleonora has suggested that you should only implement direct journeys for now. Searching for connecting journeys will be a challenge for another time.

For flexible walk and taxi options, the program will need to set the appropriate departure and arrival dates and times. The `Moment.getMomentAfter` method may come in useful for this.

If there are multiple equally good journeys according to a heuristic, then you can print any one of them. But print only one.

The journey does not have to begin immediately at the user provided time, but the maximum wait should be 4 hours. The destination will always be different from the departure place; Alina will ensure this is handled in the front-end user interface.

If no journeys can be found for the given arguments, the program should print out `No journeys found!` and terminate with exit code 2. This should always happen if `--find` is provided but no data is loaded.

Tip: you can use quotes (or double quotes if running from terminal) to pass a string with spaces as a single program argument. You can assume the departure and arrival places will always be quoted when the `--find` flag is used.

Write a short (300-500 word) report about your design decisions

Eleonore, Kimame and Luke will want to know how you implemented the new program flags, and how you went about sorting journeys to find the best one according to a given heuristic. Write up a short report about what choices you made in your design and why, even if it didn't fully work.