CSSE2002/7023

Programming in the Large

Assignment 1 Overview

Context

Assignment 1 involves modelling a public transportation network. This model will then be built on in future assignments. The transportation network involves:

- **Passengers** — who travel in the network
- **PublicTransports** — vehicles which the passengers travel on
- **Stops** — which the vehicles stop at, and which the passengers travel to/from
- **Routes** — which vehicles travel along. Routes are effectively a collection of multiple stops
- **Exceptions** — for when things go wrong in the network

Passenger

Main class:

- `Passenger` — a standard passenger. Has a name and a destination.

One subclass:

- `ConcessionPassenger` — extends `Passenger`. Has a concessionId, which can either be valid or invalid.

PublicTransport

Main class:

- `PublicTransport` — Defines the basic vehicle functionality. This class is abstract, and thus *cannot be instantiated*. To create a new vehicle object, one of the subclasses will need to be used. A vehicle has:
  - an ID
  - a capacity (the maximum number of passengers which can be on this vehicle at a given time)
  - a route (this is the path which this vehicle will follow. A vehicle can also only travel to stops which are along its route)
  - passengers which are currently travelling on this vehicle
  - a current stop. If this vehicle is currently travelling between stops, then the current stop can be thought of as its destination (i.e. its next stop).

PublicTransport subclasses

PublicTransport has three subclasses:

- `Bus` — also has a registration number
- `Train` — also has a carriage count
- `Ferry` — also has a ferry type

These subclasses are not abstract, and can thus be instantiated.

Note that the respective subclasses can only travel along the correct Route for their type (for example, a Bus can only travel on a BusRoute).

Stop

- `Stop` — a stop has:
  - a name
  - x- and y-coordinates
  - passengers waiting at it
  - routes (these are the routes which go past this stop)
  - vehicles currently at it (vehicles stop at stops to drop off and collect passengers)
  - neighbouring stops (the stops which are next to this stop on a route)

Stop has no subclasses. In future assignments, stops will be responsible for routing passengers to their destinations — for this reason, when vehicles arrive at stops, all their passengers should be unloaded.

Route

Main class:

- `Route` — Defines the basic route functionality. This class is abstract, and thus *cannot be instantiated*. To create a new route object, one of the subclasses will need to be used. A route has:
  - a name
  - a route number
  - vehicles (which are travelling along this route)
  - a collection of stops making up the route

Route subclasses

Route has three subclasses:

- `BusRoute` — a route which a Bus can travel along
- `TrainRoute` — a route which a Train can travel along
- `FerryRoute` — a route which a Ferry can travel along

These subclasses are not abstract, and can thus be instantiated. These subclasses do not add any additional functionality to the base Route class, their purpose is to provide specific routes for the various vehicles to travel on (e.g. a BusRoute for Buses).

Exceptions

- `TransportException`
- `EmptyRouteException`
- `IncompatibleTypeException`
- `NoNameException`
- `OverCapacityException`

Things You'll Need To Know

- How to write basic Java classes (week 2 lecture, week 3 prac)
- How to write and use exceptions (week 3 lecture, week 4 prac)
- How to write JUnit tests (week 4 lecture, week 5 prac)
- SVN — *you cannot submit your assignment unless you know how to checkout, add, and commit in SVN* (week 3 prac, SVN guides)

Tips

- Write parent classes before their subclasses
- Write stubs of all classes and methods before writing code
- Start with the classes that don't depend heavily on other classes. A suggested order is as follows:
  1. `TransportException` and subclasses
  2. `Passenger` and `ConcessionPassenger`
  3. `PublicTransport` and its subclasses
  4. `Stop`
  5. `Route` and its subclasses
- Write tests for all your classes, not just the tests you have to submit
- Write tests progressively as you write code (to ensure your base functionality is correct before continuing)
- Follow the style guide from the beginning (to avoid needing to reformat all your code at the end)

SVN tips

- Commit frequently to avoid losing work (follow the guidelines laid out in the SVN guides on Blackboard)
- Ensure that you have started your assignment (and ideally stubbed all your classes) before the prechecks