Programming Assignment 2: Concurrent Railway Signalling

Due 8 Oct by 23:59 **Points** 200 **Submitting** an external tool

Assessment

Weighting:	20% (200 Marks)
Task description:	In this assignment, you will be designing and implementing Petri-nets to manage signalling in a rail yard.
Academic Integrity Checklist	Do ✅ Discuss/compare high level approaches ✅ Discuss/compare program output/errors ✅ Regularly commit/push your work and add comments/notes on you commits Be careful ❓ Code snippets from reference pages/guides/Stack Overflow must be attributed/referenced. ❓ Only use code snippets that do not significantly contribute to the exercise solution. Do NOT ❌ Submit code or Petri-nets not solely authored by you. ❌ Post/share code or complete Petri-nets on Piazza/online/etc. ❌ Give/show your code or Petri-net to others ❌ Set your GitHub repository to "Public"

Before you begin

This assignment will be marked using a combination of automated and manual techniques.

You must log your development progress

- During manual marking, we will look look at your development process. If we do not see a clear path to a solution
  (i.e. code changes and regular commits and comments on those commits **reflecting** your learning to develop your implementation)
  you **may forfeit up to 100 marks.**

- *An example case of forfeiting 100 marks would be the sudden appearance of working code with no prior evidence of your development process*.
- It is up to you to provide evidence of your development through regular commits and comments.

You must test your code

- Event-Driven code is especially error-prone, so testing is important.
- During marking, we will look look at your test cases and test case coverage.
  If you do not sufficiently test your code you **may forfeit up to 100 marks.**
- *An example case of forfeiting 100 marks would no test cases present*.

This assignment requires thought and planning. You need to start early to allow yourself time to think of what and how to test before writing any code. Failing to do this is likely to make the assignment take far more time than it should.

Aims

- Apply Event-Driven Techniques to manage concurrent events.
- Apply formal methods and Petri-nets to the design of a robust system.

Overview

You have been hired by a railway company to build a prototype of the software to manage an interlocking for one corridor in their rail network:

An **interlocking** (https://en.wikipedia.org/wiki/Interlocking) is the system that ensures signals and points in a section of railway do not allow trains to occupy the same section of track.

Your system needs to:

- Allow trains to pass through the area without colliding.
- Avoid deadlock & ensure all trains that enter the area are able to successfully pass through if possible.
- Make efficient use of the track

Your Task

Your task is to plan and implement the system described above.

- See **Rail Network Details** below for more information on system layout and requirements

First, plan out your system using Petri Nets

- Refer to the Notes/Readings listed in the section below for ways to use Petri-nets to represent this type of system.
- Include all of your working in your Github commit log/comments.
  - Since you won't have any code to commit at this stage, you could add your current diagram, or alternately keep a text-based log of your progress whose commits you can add comments on.

Finally, implement your code.

- You are given a Java Interface **HERE** ↓ **(https://myuni.adelaide.edu.au/courses/64843/files/9674831/download?download_frd=1)** that provides function signatures that your implementing class(es) will need to implement.
  - Review this and make sure you understand it before writing code.
- Your implementation should at minimum contain a class named `InterlockingImpl` in a file named `InterlockingImpl.java` that implements the interface above.

Event Driven systems are particularly prone to errors so you'll also need to:

1. Document your planning of the system (See **Logging/Documenting your Progress** below)
2. Write your own suite of test cases for your code (See **Testing your Code** below)

If you get stuck, ask on Piazza. Feel free to discuss and plan with your peers, but write the Petri-net and code yourself.

Programming Language/Software Requirements

Be sure to adhere to the following requirements. Not doing so may result in a **reduction or complete loss of marks**.

Architecture

- Your system must be planned and developed using a Petri-net

Version Control System

- Your work must be stored and submitted using GitHub
- You will need to log your progress by commenting on your commits

Programming Language

You will need to use Java & JUnit to complete this assignment.

- Your code will be run using JDK 11
- Your test code will be run using JUnit 4
- Your implementation may use any other libraries/classes available in the standard JDK and JUnit, but **no other external libraries/classes**.
- Your classes will be run as shown below:
  - Compiled with

    ```
    javac InterlockingImpl.java
    ```

  - Your class' methods will be run by a driver class.
    You will need to implement your own driver code for testing.
  - Tests compiled with:

    ```
    javac *_Test.java
    ```

  - Each Test run with:

    ```
    java org.junit.runner.JUnitCore TestName_Test
    ```

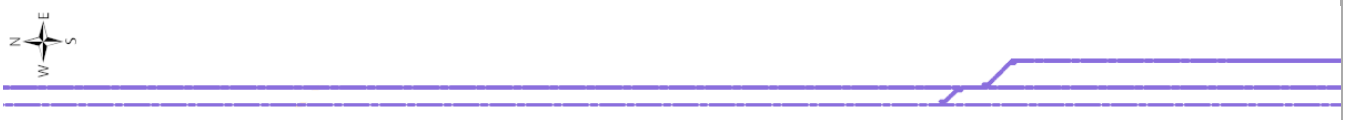
Railway Network Details

Overview

The corridor consists of 11 sections plus several junctions and crossings:

This can be split into 2 separate parts, a freight line that connects to a set of workshops:

And a passenger line:



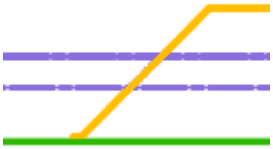
These 2 parts use different tracks and trains cannot move between them.

Junctions

There are 2 main junctions in the corridor

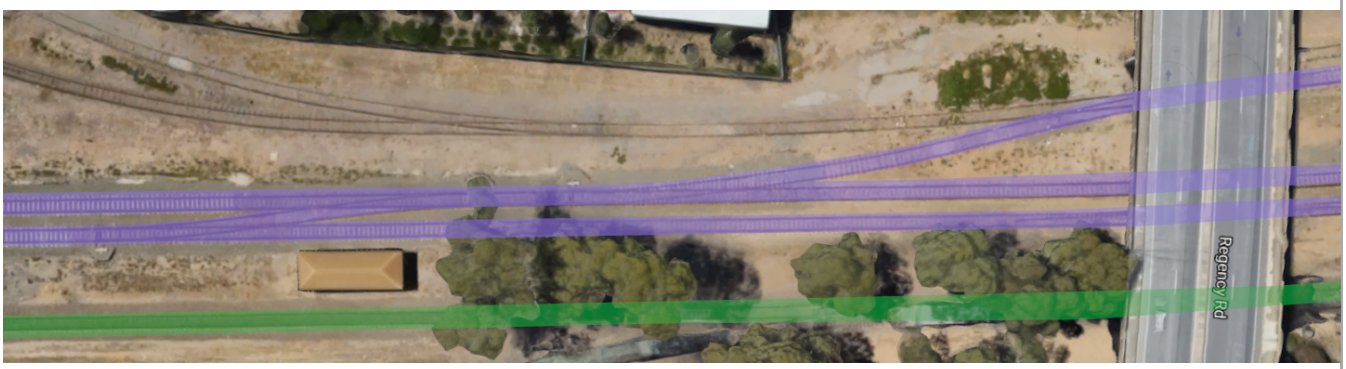
There are 2 main junctions in the corridor:

One where the freight line splits and crosses the passenger line:



Passenger trains should always have priority at this junction.

And another where the passenger line splits from 2 tracks to 3 tracks:



Entry & Exit

Trains enter and leave the corridor as follows:

- Trains travelling South (left-to-right):
  - Can enter into sections 1 & 3.
  - Can exit from sections 4, 8, 9 & 11
- Trains travelling North (right-to-left):
  - Can enter into sections 4, 9, 10 & 11
  - Can exit from sections 2 & 3

Constraints

Your system needs to:

- Allow trains to pass through the area without colliding.
  A collision happens if trains:
  - Occupy the same section of track
    i.e. a train should not be permitted to enter a section of track if another train is present
  - Cross paths as they move between sections
    i.e. 2 trains should both not be permitted to enter the same part of an intersection.
- Avoid deadlock & ensure all trains that enter the area are able to successfully pass through
  - i.e. trains should not get stuck.
  - There may be some scenarios in which avoiding deadlock is not possible, but your system should minimise these.
- Avoid mixing freight and passenger trains
  - i.e. A freight train should not be able to switch to the passenger line
- Make efficient use of the tracks
  - i.e. all sections of track can be utilised.

Fun Facts

This assignment is based on a (simplified) real section of railway around the Islington workshops in Adelaide:

Sources:

- **Google Maps Satellite View (https://www.google.com.au/maps/@-34.8702879,138.5804872,910m/data=!3m1!1e3)**
- **OpenStreetMap View (https://www.openstreetmap.org/#map=16/-34.8665/138.5798&layers=H)**
- **ARTC Network Map (https://www.artc.com.au/uploads/ARTCS3090005_EW_SA.pdf)**

Submission, Assessment & Marking

Submission

Submissions open 11:59pm Friday 10 Sept.

Before submitting

- You will need a github.com account to submit
- Be sure to set any repository you create for this assignment to **private**.
- Your source files must either be in the root of your repository, or in a `/src` folder

To submit:

1. Click the Load Programming Assignment ... in a new window button at the bottom of this page
2. Select the Submit button on that page
3. If prompted, link your Github account
4. Select the repository and branch that you want to submit
5. Submit and wait for the system to check your work.

Your submission will run through some acceptance tests and provide basic feedback. Full testing comes after the assignment deadline and will be done using a combination of automatic and manual tests.

Assessment

The assignment is marked out of 200. These 200 marks are allocated as follows using a combination of automated testing and manual review:

- Planning and Design of Petri-net **[80 marks]**
- Basic Implementation **[45 marks]**
- More complex traffic routing **[45 marks]**
- Edge cases **[30 marks]**
- Full mark details to come closer to deadline

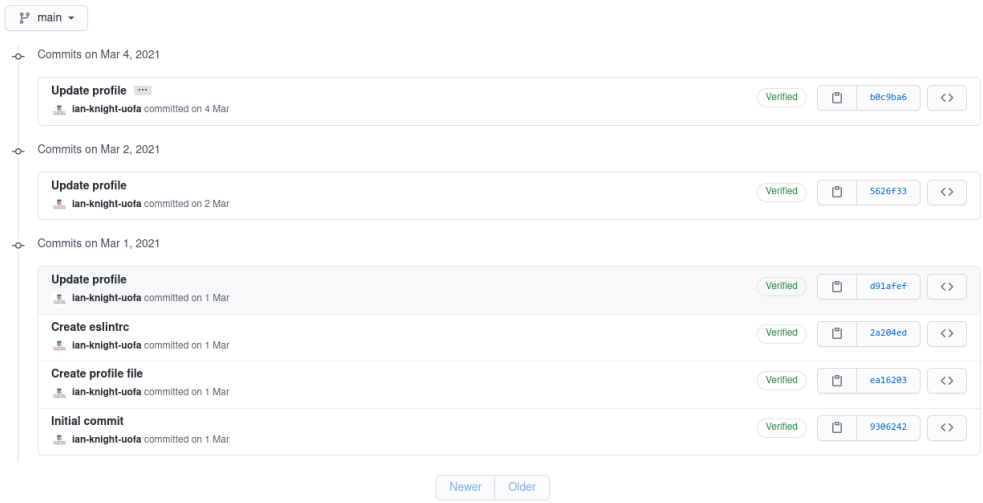
However, your marks are **scaled and reviewed** based on the following:

1. Up to **10 marks may be deducted** for poor code quality.
   - When in doubt, consider **Google's Java Code Style Guide (https://google.github.io/styleguide/javaguide.html)** .
   - Javadoc comments are always recommended.
2. Up to **100 marks may be deducted** for poor or missing testing.
   - Your test code will be reviewed against a code coverage checker. Poor test coverage will result in lost marks.
3. Up to **100 marks may be deducted** for poor/insufficient/missing evidence of development process.
   - This includes cases where you've designed your Petri-net after writing your code.

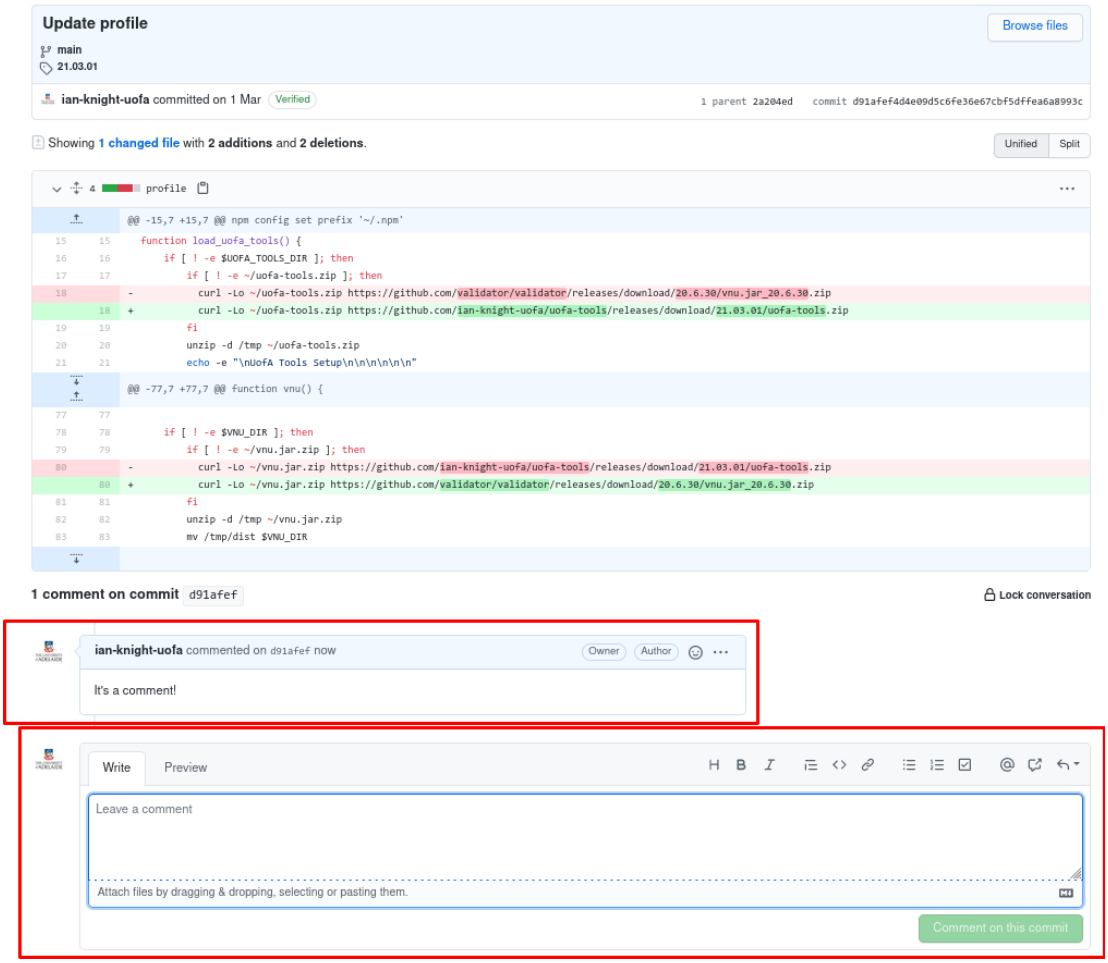
Logging/Documenting your Progress

We'll be using GitHub to log our development progress this semester.

Each time you make a commit and push those changes, they will be visible in your commit history:



Each commit can have additional notes and comments added to it:



Testing your Code

We'll be using JUnit to test our code

Take a look at their Getting Started guide: **https://github.com/junit-team/junit4/wiki/Getting-started (https://github.com/junit-team/junit4/wiki/Getting-started)**

You will be reviewed on your test coverage, which will be done using a combination of manual review and automated test coverage reports.

Additional Notes/Resources

General Petri-net structures

- Section 6.2 of **Notes on Finite State Machines**
  The buffer structure will be particularly useful here.

Modelling Railway networks with Petri-nets

- **A. M. Hagalisletto et al; Constructing and Refining Large-Scale Railway Models Represented by Petri Nets**
- **Mandira Banik et al; Railway Network Modelling Using Petri Nets (https://pdfs.semanticscholar.org/e3fb/6ea427d700818eac85fd233abe53045a4f92.pdf)**

This tool needs to be loaded in a new browser window

Load Programming Assignment 2: Concurrent Railway Signalling in a new window