COSC427: Advanced OO Design

Exam 2007

Time allowed:	**2 hours**.
Number of pages:	**5**.
Number of questions:	**6**.
Total marks:	**100**.
Resources:	**Open book**: You may use any printed or electronic resources. You may not communicate, directly or electronically, with other people. If you want to print anything, ask a supervisor to collect it from the printer for you.
Answers:	Write all answers in your **answer book**.

1. **[26 marks for whole question]** For each part (a)-(m), name the idea (or ideas) that best match the description and briefly explain your answer. If possible, include WikiNames.

   (a) [2 marks] Keep the number of fields in a class small.

   (b) [2 marks] Methods should use most of the fields of the method's object.

   (c) [2 marks] When several methods cooperate to do a job, give them the same name.

   (d) [2 marks] Objects should invoke methods of contained objects.

   (e) [2 marks] A CodeSmell that supports one of RielsHeuristics.

   (f) [2 marks] A pattern that conflicts with TellDontAsk.

   (g) [2 marks] A pattern that supports TellDontAsk.

   (h) [2 marks] A pattern that employs DoubleDispatch.

   (i) [2 marks] A pattern based on ModelTheRealWorld.

   (j) [2 marks] A pattern that conflicts with one of RielsHeuristics.

   (k) [2 marks] A pattern that exhibits a CodeSmell.

   (l) [2 marks] An architectural pattern that contains a design pattern.

   (m)[2 marks] A design problem that the Waterfall Process was intended to fix.

2. **[10 marks]** Which of RielsHeuristics can be traced back to ideas described in JohnsonAndFoote1988? For each heuristic you identify, explain how it is based on the earlier idea.

3. **[16 marks for whole question]** For each of the following maxims (a)-(d) name other maxims (as many as you can) that support it. Briefly explain each answer.

   (a) [4 marks] ProgramToTheInterfaceNotTheImplementation.

   (b) [4 marks] SeparationOfConcerns.

   (c) [4 marks] SoftwareReuse

   (d) [4 marks] OpenClosedPrinciple

4. **[8 marks for whole question]** Describe a design flaw in a 427 project produced by one of your classmates. Choose the most serious flaw you can find, clearly describe where it occurs (including where it can be found in the wiki) and support your argument, where possible, with maxims etc.

5. **[5 marks]** Imagine it was your job to interview an applicant for an OO design job. If you were allowed to ask only one question, and from the answer you had to judge whether the person was a skilled OO designer, what would you ask? Explain your reasoning.

6. **[35 marks for whole question**] The following questions refer to the UML class diagram in Figure 1 and explanatory notes in Figure 2. Some getters and setters and other details are omitted from the diagram, but may be assumed where necessary. Document any non-trivial assumptions you need to make.

   (a) [18 marks] Find as many Gang of Four design patterns as you can in the `Trains` design. Name each pattern and describe where and how it is used in this design. Provide just enough information to make it clear how and why the pattern is applied here. Note any important variations from the standard pattern. There is no need to comment here on the value of the pattern (but see next question).

   (b) [17 marks] Criticise and illuminate the design. Find as many weaknesses or issues as you can, and wherever possible name relevant maxims, smells, refactorings, etc, to reinforce your arguments. Also explain how each problem could be fixed.

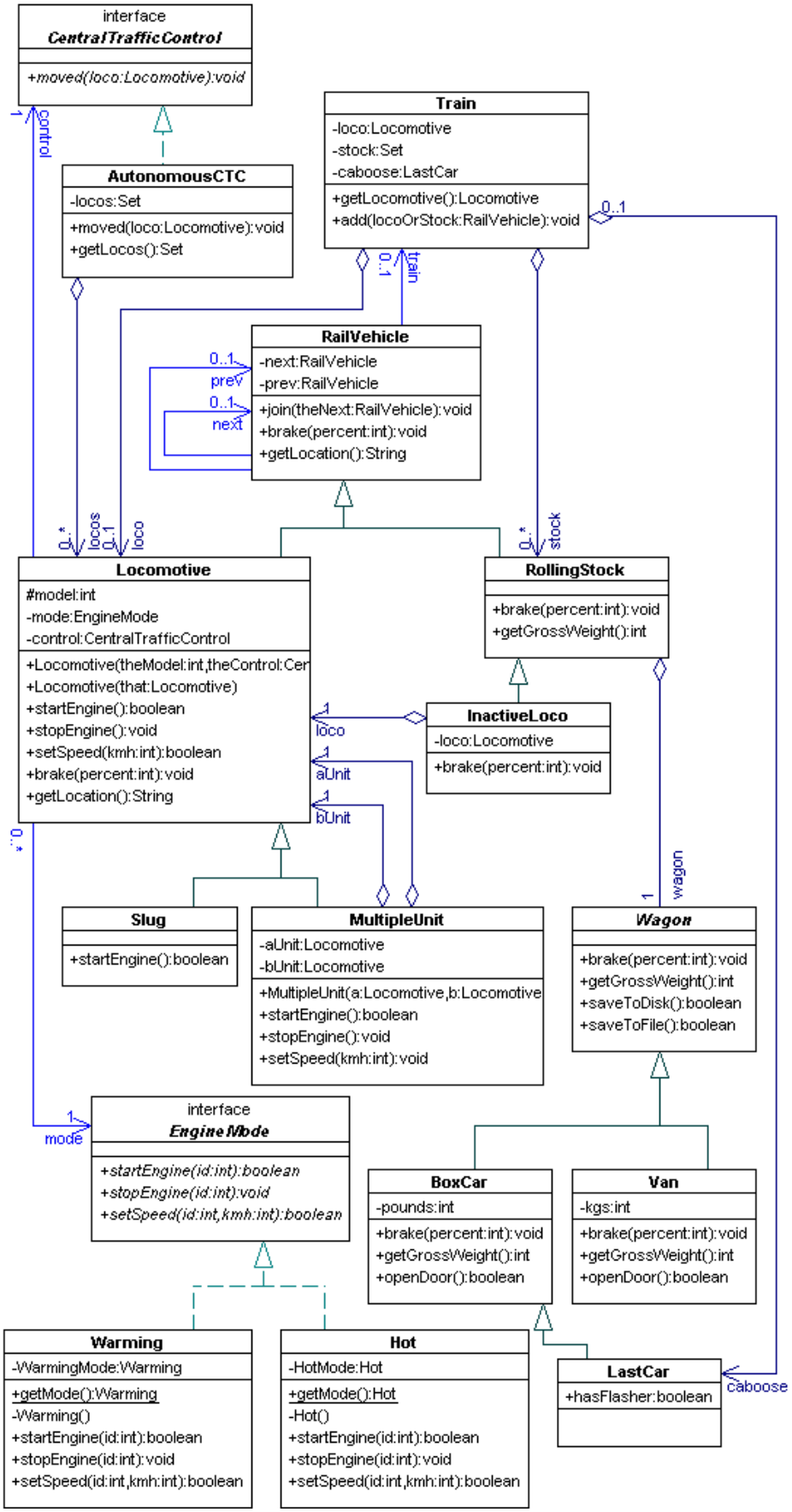

Figure 1: Trains class diagram

- This is a preliminary design for managing railway `Train`s. Each `Train` consists of an optional `Locomotive` and some `RollingStock` (i.e. railway vehicles that can't move themselves). In North American installations, a `Train` may also have a `caboose`, which is just the last wagon fitted with a flashing light.
- `Locomotive`s and `RollingStock` are `RailVehicle`s. Each `RailVehicle` knows the `RailVehicle` coupled immediately before and after it. These `next` and `prev` references are set by the `join()` method.
- When a `RailVehicle` is told to `brake()`, it applies its own brakes and also passes the message to its `next` `RailVehicle`, if one exists.
- Similarly, when a `RailVehicle` is asked to `getLocation()`, it forwards the request to its `prev` `RailVehicle`, if one exists. When the request reaches the `Locomotive` at the front of the train, it returns its location.
- A `Locomotive` pulls or pushes a `Train`. The `startEngine()` method must be called, then `setSpeed()` tells it what speed to accelerate to. A negative speed indicates reverse.
- `Locomotive`s that have not fully warmed up to operating temperature should be handled more gently. This is achieved by forwarding engine commands to an `EngineMode`, which is either `Warming` or `Hot`. Other `Locomotive` behaviours depend on what kind of `Locomotive` it is, as indicated by the `model` number.
- `Locomotive`s regularly report their changes of location to a `CentralTrafficControl` object, which can slow or stop trains to avoid collisions and congestion.
- When two `Locomotive`s are used together, they form a `MultipleUnit`. The front `Locomotive` is known as the `aUnit` and the second as the `bUnit`.
- A `Slug` has no engine, so overrides `startEngine()` to do nothing. It can only be used as a `bUnit`, because it needs a power source from an `aUnit`.
- `RollingStock` implementations differ depending on user preference. North American installations typically use `BoxCar`s, while elsewhere `Van`s are used.
- Sometimes an unused or broken `Locomotive` is pulled as if it were `RollingStock`. `InactiveLoco` caters for this situation.

Figure 2: `Trains` notes

END OF PAPER