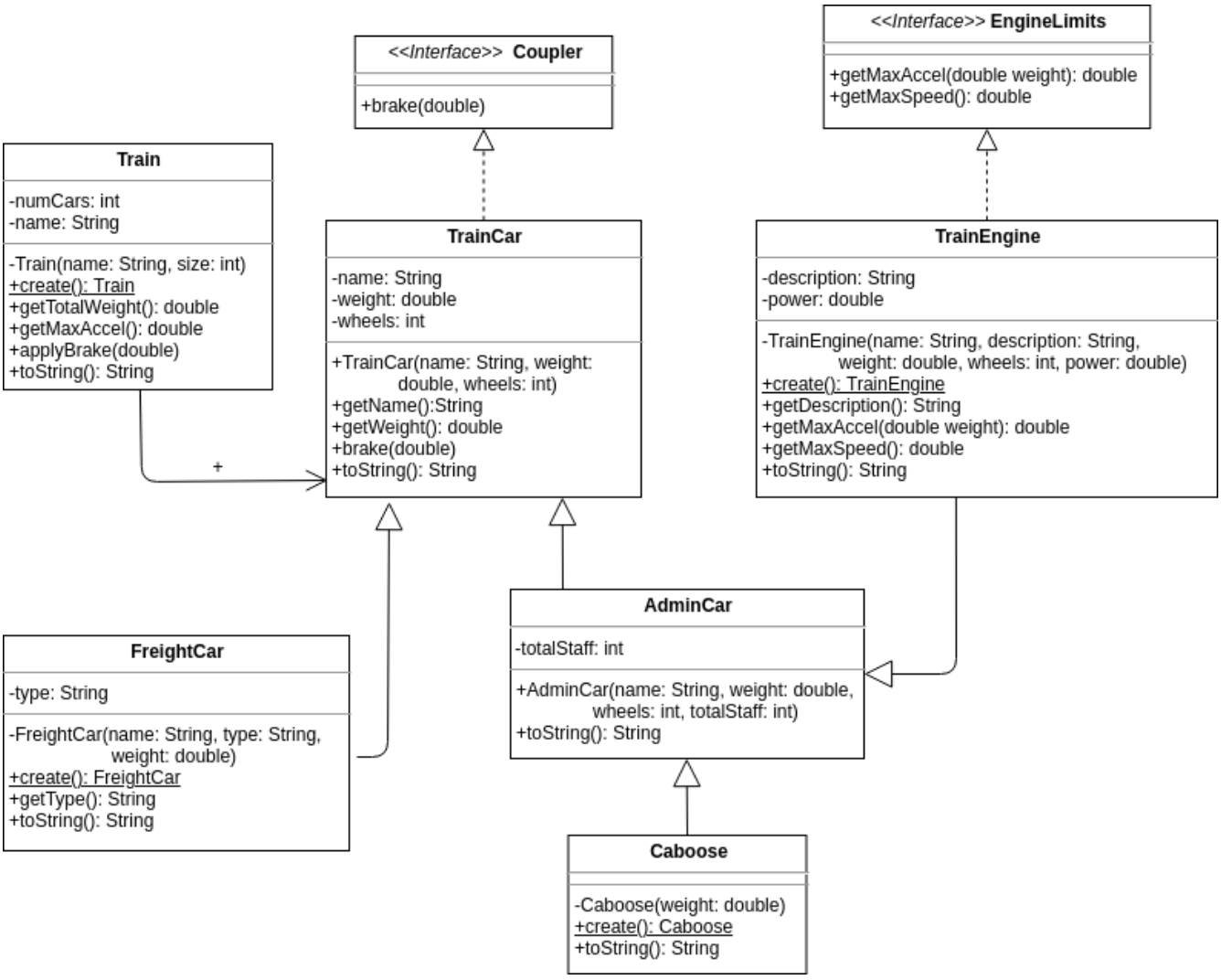
COMP10062: Assignment 7

Mohawk College, Summer 2023

The Assignment

This assignment is about polymorphism, abstract classes, and interfaces. The UML class diagram below shows an collection of polymorphic classes defined through inheritance, association and specific interfaces. Your job is to implement the classes and relationships shown here.



Description

The above class diagram represents the classes necessary for describing a freight train. There are 3 kinds of TrainCars, the TrainEngine pulls or pushes the train. A train always has at least one engine but can have more. The FreightCar is a generic TrainCar and may be a flat-bed, tanker, or box-car. The caboose is a special freight car, often built from a converted box car and includes a kitchen and small living area for train employees. It is normally on the end of a train.

Help With Freight Trains

A freight train engine weighs between 100 tonnes and 250 tonnes, has a maximum speed of around 100 km/h, and generates between 600KW and 3000KW of power.

A freight train can have 100s of cars of various types. The typical weight of a loaded freight car is about 160 tonnes. About 1/2 of the weight is the car itself.

The caboose is usually the last car on the train and is used to house workers and provides a location for administration. It also provides an observation point for workers to check that the train cargo is secure and safe.

See: https://en.wikipedia.org/wiki/Locomotive

See: https://en.wikipedia.org/wiki/Railroad_car

Implementation Notes

Not a Train Simulation

The purpose of this program is to catalogue the weight and speed details of a freight train cars and compute the total weight of the train. From the weight we can compute a trains maximum acceleration. Administrators who operate freight train companies need to know how many engines they will need to move their inventory, and whether their train is too heavy.

Train Car Types and Names

Each train car should have a unique name. The name can be a simple serial number, or include the details of the company that manufactured the train car. There are many types of freight cars, typical types include the box-car, flat-bed, and tanker, but you can include others.

brake() and applyBrake()

Every train car has brakes on the wheels on all of the cars. For the train to be able to stop, all train cars must provide the same brake interface. The method doesn't need to do any calculations. The brake method takes an argument in the range of zero to one and should just print the car details and say that the brake has been applied. The Train class should provide an applyBrake() method which calls brake method on all the cars to check that they are working and report on all the cars in the train. Use the toString() method to report details for each car.

getMaxAccel()

The `getMaxAccel()` method in EngineLimits should return a number in m/s^2. The maximum acceleration is determined by dividing the total weight of the train by its power. 10 m/s^2 is the acceleration due to gravity. This is the maximum acceleration for an engine with no cars. A train should never have so much weight that it accelerates at less than 0.1 m/s^2. Accelerating at 0.1 m/s^2 means that the train will take about 10 minutes go from 0 to 100 km/h. Train engine power is additive, so two engines can make the same group of cars accelerate twice as fast.

Computing Acceleration from Engine Power and Total Weight

Train engine power is measured in KW. A typical freight train will have an engine which generates between 600KW and 3000KW of power. You can compute the maximum acceleration of the train by dividing the engine power by the total weight of the train. Here is a sample calculations to get you started:

Assume you have 10 train cars, at 160 tonnes each, two cabooses at 75 tonnes each, and one train engine at 250 tonnes. Your **total train weight is then 160 x 10 + 75 * 2 + 250 = 2000 tonnes.** If your train engine generates **1000KW of power** then your **maximum acceleration is 1000KW / 2000 tonnes** which is **0.5 m/s^2**. This meets our minimum requirement, so this train will be good for rail travel.

Constructors vs. Create Methods[1]

Notice that many of the constructors are private. The classes with private constructors have static `create()` methods. Instead of calling a constructor, call the `create()` method, like this:

```
FreightCar c1 = FreightCar.create();
```

This method will have a dialog with the user to collect information about the object (in this case, name, type and weight), and will then create and return an object by calling the private constructor. When you add new cars to your train, you should follow this same pattern.
The `Train` class also uses a `create()` method. This method asks the user the name of their train and how long the train should be, then creates a `Train` object by calling its private constructor. For each TrainCar, the Train constructor asks what type of car to add, Engine, FrieghtCar or Caboose, then calls the appropriate `create()` method and stores the result in the array. You should use the special Java type ArrayList(), discussed in upcoming lectures.

Your constructors should enforce sensible limits. You cannot have a TrainEngine that weighs 10 tonnes, has 3 wheels, and generates 10,000 KW of power. Use the notes on the previous page which describe freight trains and ensure they are enforced.

There is no maximum length for trains. Use ArrayLists and see the notes on getMaxAccel().

Model vs. View

Note that because the `create()` methods are talking to the user, this assignment does not maintain the standard model/view separation. However, you can think of the static methods as implementing the **view**, and the instance methods and variables as implementing the **model**.

---

1 This is known in software engineering as the **Factory Design Pattern**. The create methods are called **static factory methods**. The Java FX `Color` class uses static factory methods for creating colors (`rgb()`, `web()`, etc.). Each of these methods interprets its arguments and then calls a `Color` constructor and returns the result.

TestClass.java

Use the code below to test your classes. Include this code with your submission instead of a Main.java class. This short program should initiate an elaborate dialog with the train administrators through calls to the `create()` methods, then output the brake check and the maximum acceleration rounded to two decimal places. You should be able to use this code without changing it.

```
public class TestClass {
    public static void main(String[] args) {
        Train t = Train.create();
        t.applyBrake(0.5);
        System.out.printf("Maxiumum Acceleration: $%.2f\n",
                          t.getMaxAccel());
        if (t.getMaxAccel() < 0.1)
            System.out.println("** Warning: Train is too heavy.");

    }
}
```

Evaluation

Your assignment will be evaluated for performance (40%), structure (40%), and documentation (20%) according to the standards discussed in the lectures.