Lab 03: Fleet Management Example

Aims:

1. Exercise on transferring requirement description into OOP project with a fleet management case study.
2. Figure out code smells and refactor them accordingly.

The fleet management example

Today's task is to build a backbone model for a delivery company. The company's business contains the following components:

- The company is responsible of delivering cargos. A cargo can be a fluid cargo that is usually transported by a truck tank, or a pallet cargo that is usually placed and supported by a pallet. Each type of cargo features different load and unload methods. A cargo should attach a RFID tag which is used to identify it. Thus, each RFID tag has to be unique.
- The company owns a variety of carriers, such as lorries, trains, and ships, which are used to transport cargos. Each carrier owns an id, and carries a set of cargos.
- Specifically, the lorries should operate under a set of states, namely IN-USE, AVAILABLE, MAINTENANCE, and LOCKED-AWAY. The state diagram is shown in Fig. 1. Note that the state transition conditions have not been clearly defined currently in the company, so only a `setState(newState)` method is available in this design. When a *newState* is set, it means the *newState* should replace the *currentState* in the system. That is, a state transition may happen. Of course, you need to be careful that the *currentState* cannot directly jump to the *newState* if they are not adjacent in the state diagram. In that case, the `setState()` method should return `false`.
- Both the cargos and the carriers should be trackable. The company track them using their information of (1) last updated time and (2) latest location.
- A data manager is needed in the system to contain the carrier and cargo information. It should also be able to load the information of the carriers,

e.g., lorries, into the system from a text file that contains the information, as shown in Fig. 2.

- The system is used by various personnel, such as customers and employees of the company. Employees can be clerks, managers, and drivers. Those users' information should also be maintained in the data manager.
- The system also contains a RoutePlanner, which is used to plan the route of the carriers. The RoutePlanner contains the set of all routes, and the set of all route nodes. A route node is a geographic concept that is an element of a route, such as a street, a city, etc. For the RoutePlanner, for now, the planning functionality is trivial: it only adds the route and the associated route nodes into its sets of routes and route nodes, respectively. Finally, the RoutePlanner should have a method named `routeChanged(routeInstance, routeNodeInstance)`, which returns true if the routeInstance or routeNodeInstance is in the route set or route node set. It is used to indicate that a route or a route node is added to the set – the set is changed.

===

Task:

- Now, according to the above description, draw a class diagram that best represents the system. Carefully design the classes and methods, as well as relationships of composition, aggregation, inheritance, implements, etc.
- According to the class diagram, try to write your first version of code.
- Compare your code with the reference code [FleetManagement_NoRefactoring.zip] provided in Moodle. Specially, in this reference code, find the FIVE locations of 'Code Smells' marked in their comment lines, and figure out how to refactor them. Hint: Check through the items listed here to identify the types of code smell, and clean them.
- Compare with the reference code [FleetManagement_Basic.zip] to check whether you have properly refactored the code smells.

===

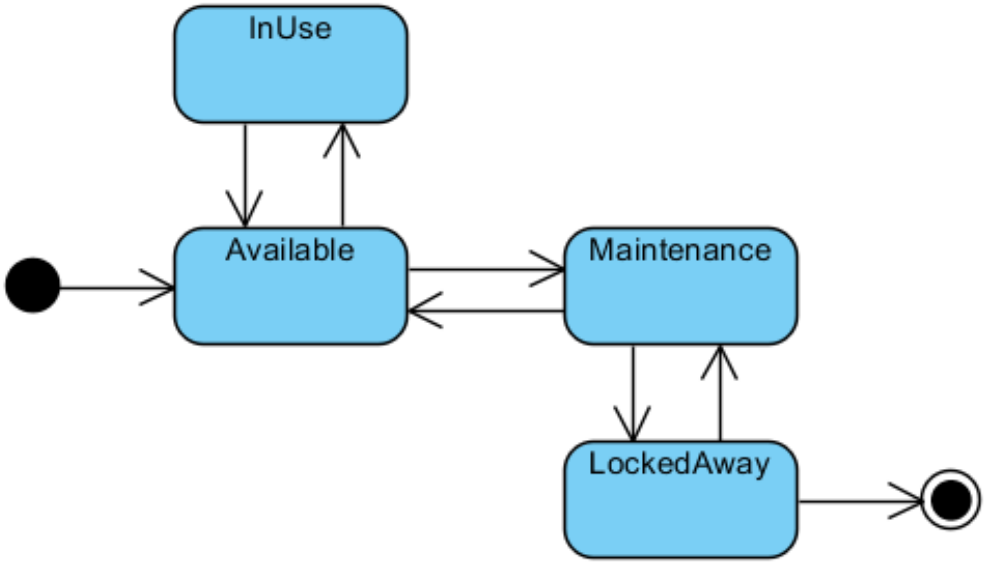

Fig. 1. State transition diagram for a lorry.

```
1    Lorry 4
2    Lorry 6
3    Lorry 8
4    Lorry 10
```

Fig. 2. Format of the text file containing lorry information. In this file, ‘Lorry’ is the carrier type, and the number following is the carrier ID.