Inf2C Software Engineering 2017-18

Coursework 2

Creating a software design for a tour guide app

SAMPLE SOLUTION

1 Introduction

This sample solution is consistent with the guidance given in the Coursework 1 and 2 handouts. While this solution offers strong suggestions for how one could create a solution for Coursework 3, it should not be taken as precise instructions for Coursework 3; there are slight differences between what is asked for in Coursework 3 and what is described here.

2 Static model

2.1 UML class model

See Figure 1 on page 2.

2.2 High-level description

- The Note class is for what I called *annotations* in the instructions.
- Further subclasses of Note could be added. For example, an HTMLNote.
- Further subclasses of Chunk need to be added. The class diagram just shows subclasses mentioned in the Coursework 2 instructions.
- The Mode box shows how one expresses an enumerated type in UML. This is a more advanced feature of UML class diagram notation and is not a feature students on the course are required to know about.

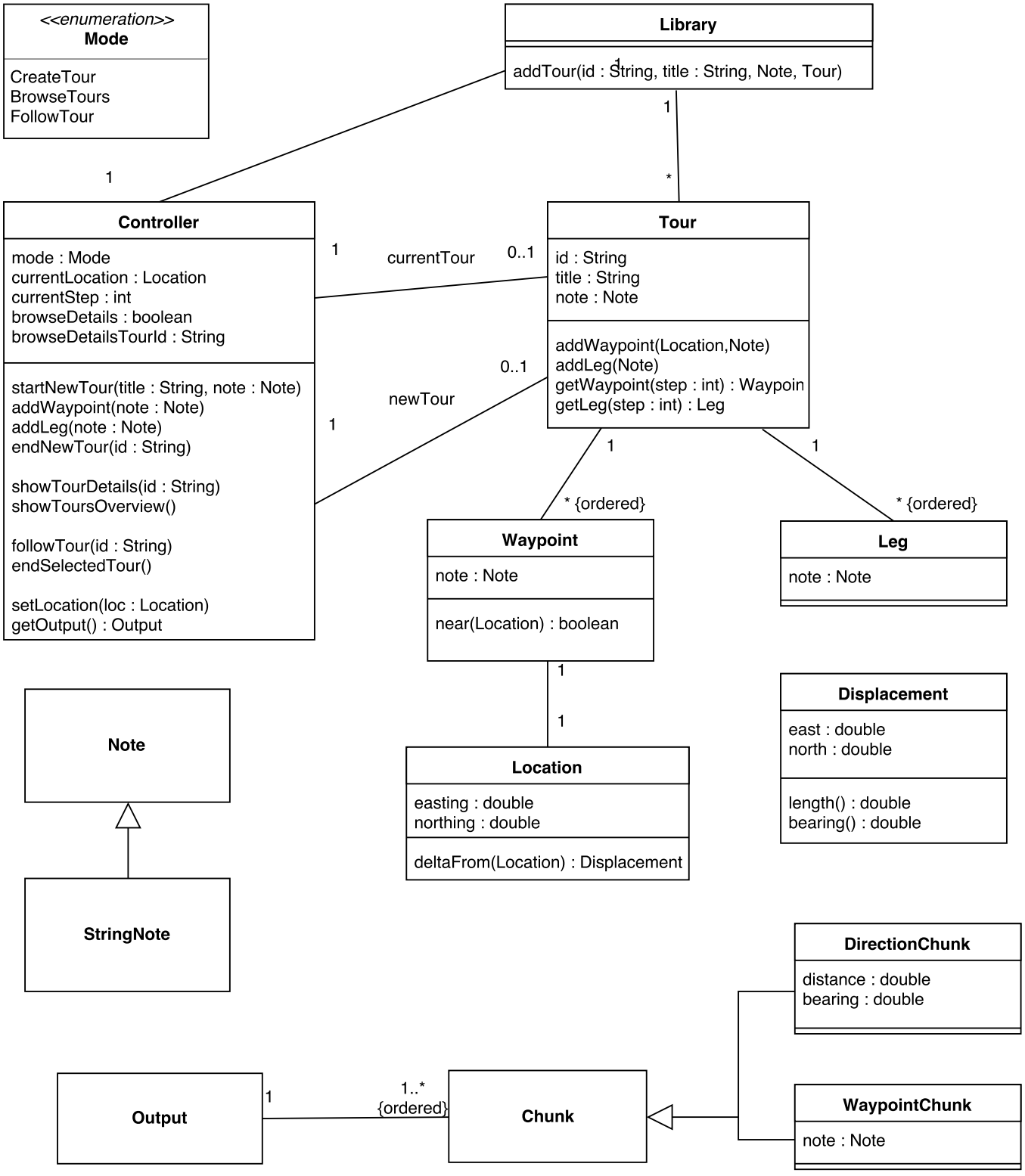

Figure 1: UML Class diagram

- A *displacement* is the position of one location relative to another. While the attributes of the Displacement and Location classes are essentially the same, the concepts are not. For example, it is meaningful to consider the size/length of a displacement, but not of a location. Two displacements can be added to give a new displacement. It does not make sense to add two locations.

- There are a number of ways a tour could be made up from waypoints and legs, each with pros and cons. With the approach shown in Figure 1, it is not obvious that the numbers of Waypoint and Leg objects for a given Tour object are closely related and code creating tours could easily create ill formed tours. The relationship between these numbers could be expressed by a class invariant. This could be recorded in a UML note associated with the Tour class in the class diagram and could later be coded in JML.

  An alternative design option would be to create a special kind of linked list with alternating Waypoint and Leg objects.

  In Figure 1 the Controller class has an integer-valued currentStep attribute, the idea being that the value of currentStep gives the index of the current waypoint and of the leg immediately after it (if it is not the last waypoint). A further design option would be to introduce a new class Step such that each Step object is associated the step's Waypoint and Leg objects. A tour then would be made up of an ordered sequence of Step objects.

- This design imagines the creation of Chunk objects to be centralised in the Controller object. An alternative is to spread this responsibility around, have the Controller object delegate Chunk object creation to the LIbrary, Tour, Waypoint, Leg and Displacement classes. Arguments can be made for the merits of either of these alternatives.

2.3 Dynamic models

2.3.1 UML sequence diagram

See Figure 4 on page 5 for a Sequence diagram showing the dynamic behaviour of the Follow Tour use case.

- No specific conditions are shown for when opt frames executed and for when loop frame exited. All these choices are non-deterministic.

- The positioning of next Waypoint and next Location object boxes in the optional frames does not imply that these objects are actually created as the flow of time in the diagram passes them. All these objects are related to the tour selected by the followTour() call at the start of the diagram and would have come into existence when that selected tour was created.

  The issue is that different iterations of the loop in general involve different such objects as the user progresses along the path of a tour, and this positioning seemed the best way of suggesting this.

- Full details are not shown for the messages computing the Note chunks for the current waypoint and leg.

2.3.2 Behaviour descriptions

For tour creation see Figure 2 on page 4. For tour browsing see Figure 3 on page 4.

```
* -- startNewTour(title, note) --> theController
    theController -- new --> Tour
    theController <- - newTour - - Tour

LOOP [For each waypoint except last]
* -- setLocation() --> theController

* -- addWaypoint() --> theController
   theController -- addWaypoint() --> newTour

* -- addLeg() --> theController
   theController -- addLeg() --> newTour
 END LOOP
* -- setLocation() --> theController

* -- addWaypoint() --> theController

* -- endNewTour(id) --> theController
```

Figure 2: Tour creation behaviour

```
* -- showTourDetails(id) --> theController

* -- showToursOverview() --> theController
```

Figure 3: Tour browsing behaviour

Paul Jackson, 15th November 2017.

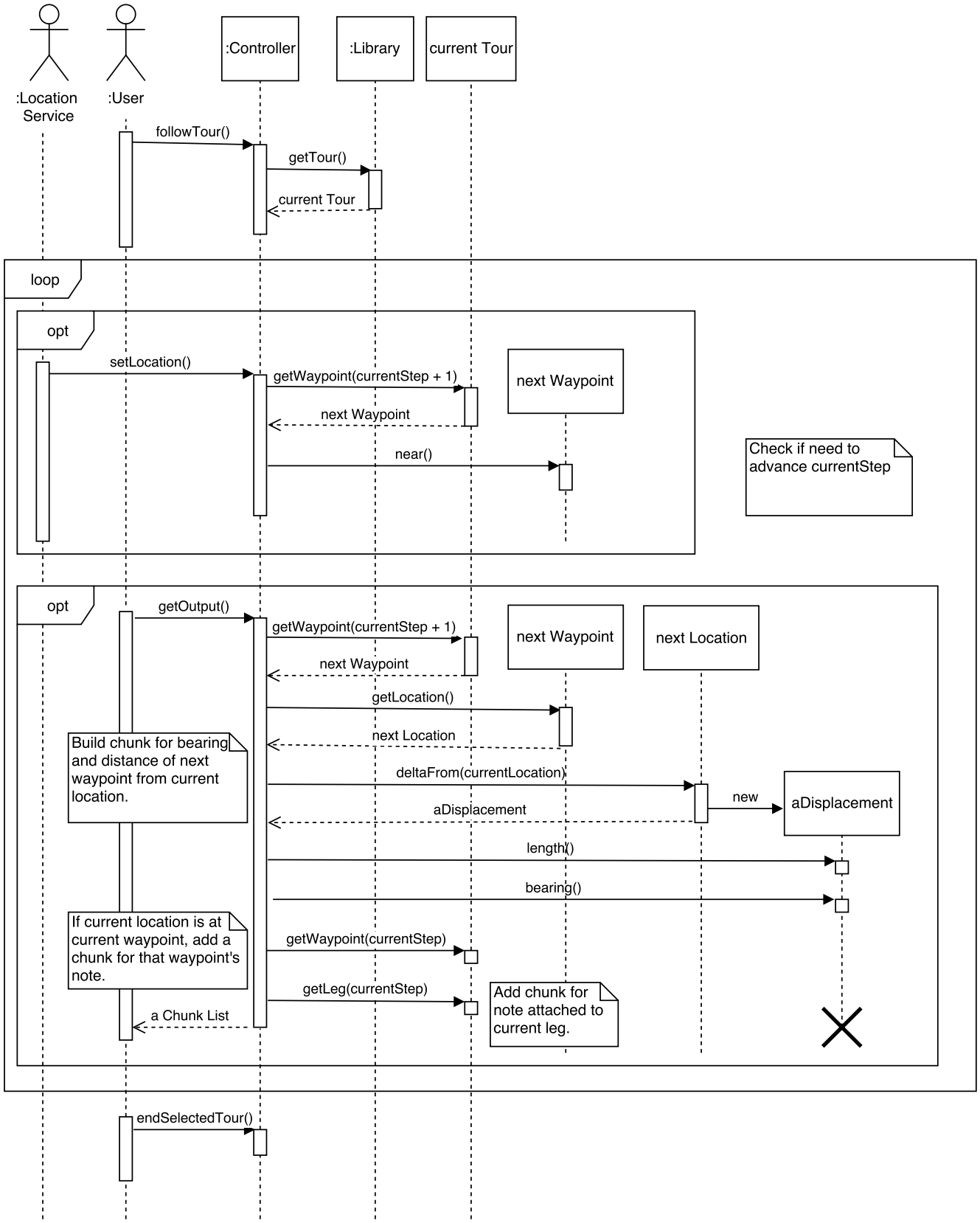

Figure 4: Sequence Diagram for Follow Tour use case