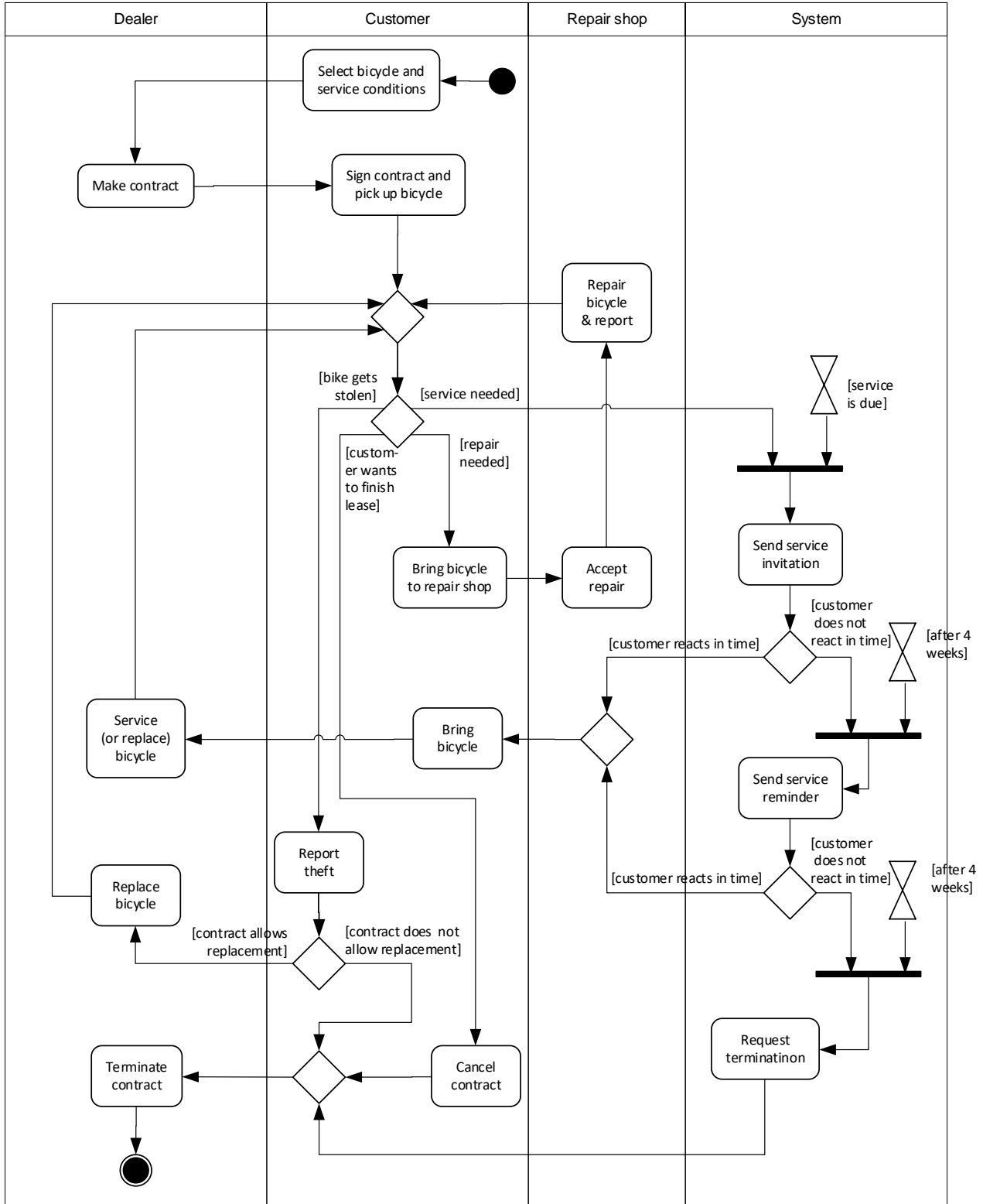


Solutions to the Software Systems Design Test

14 December 2017

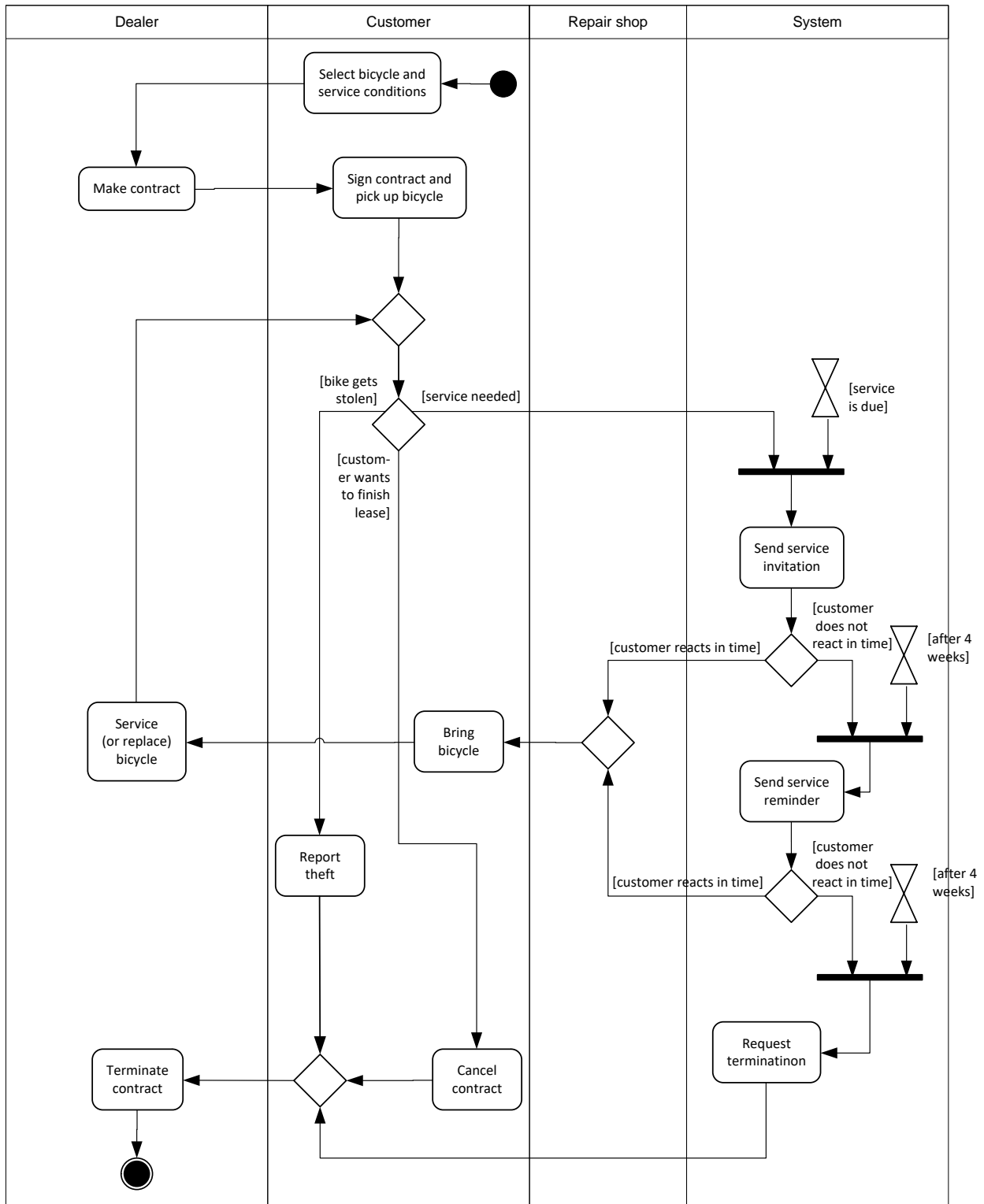
Question 1 (Activity Diagram) [2 points]

The standard solution is given below. See page 2 for an alternative solution for a possible different interpretation of the question.



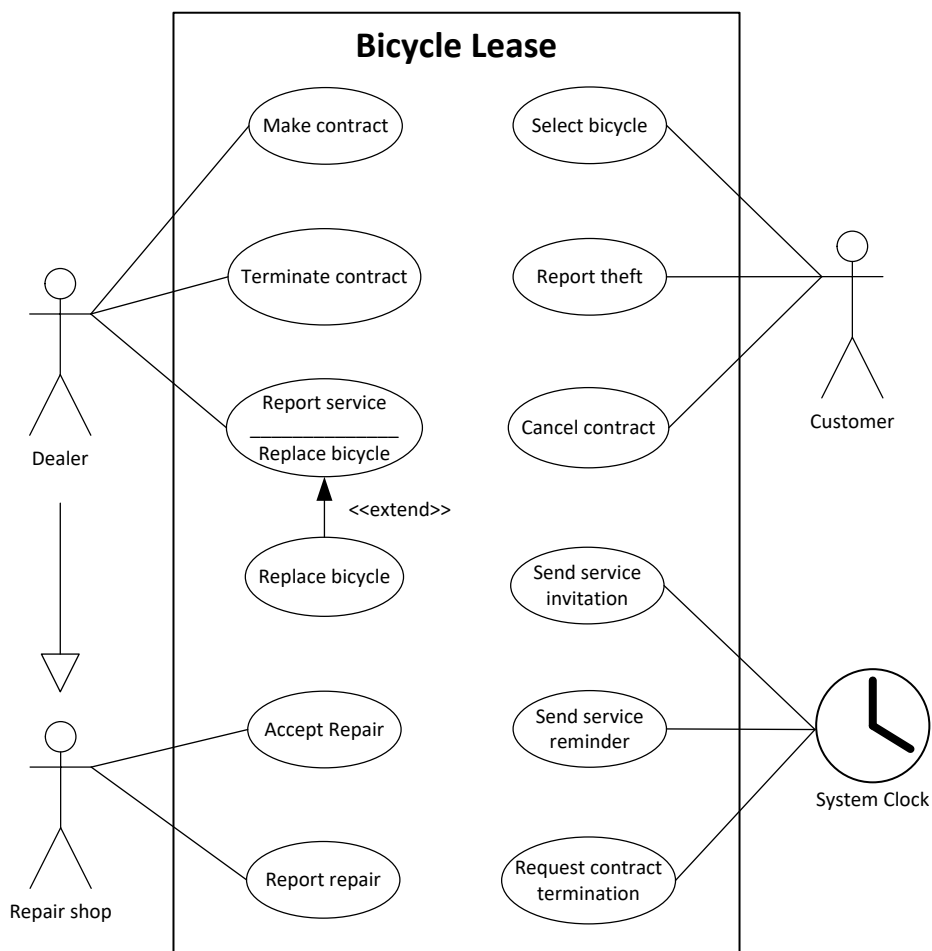
However, the question turned out to be ambiguous. The intention was to model *everything* from the selection of a bicycle to the termination of a contract. It could also be interpreted as *only activities related to leasing and servicing* (i.e., not included repairs and replacing the bicycle when it gets stolen). This interpretation yields the activity diagram below. (Stealing is still included as it may lead to termination of the contract).

All tests have been graded for both alternatives, so that solutions according to the second interpretation still could get 100 % of the points. For those who chose the first interpretation, the higher of both grades counts.



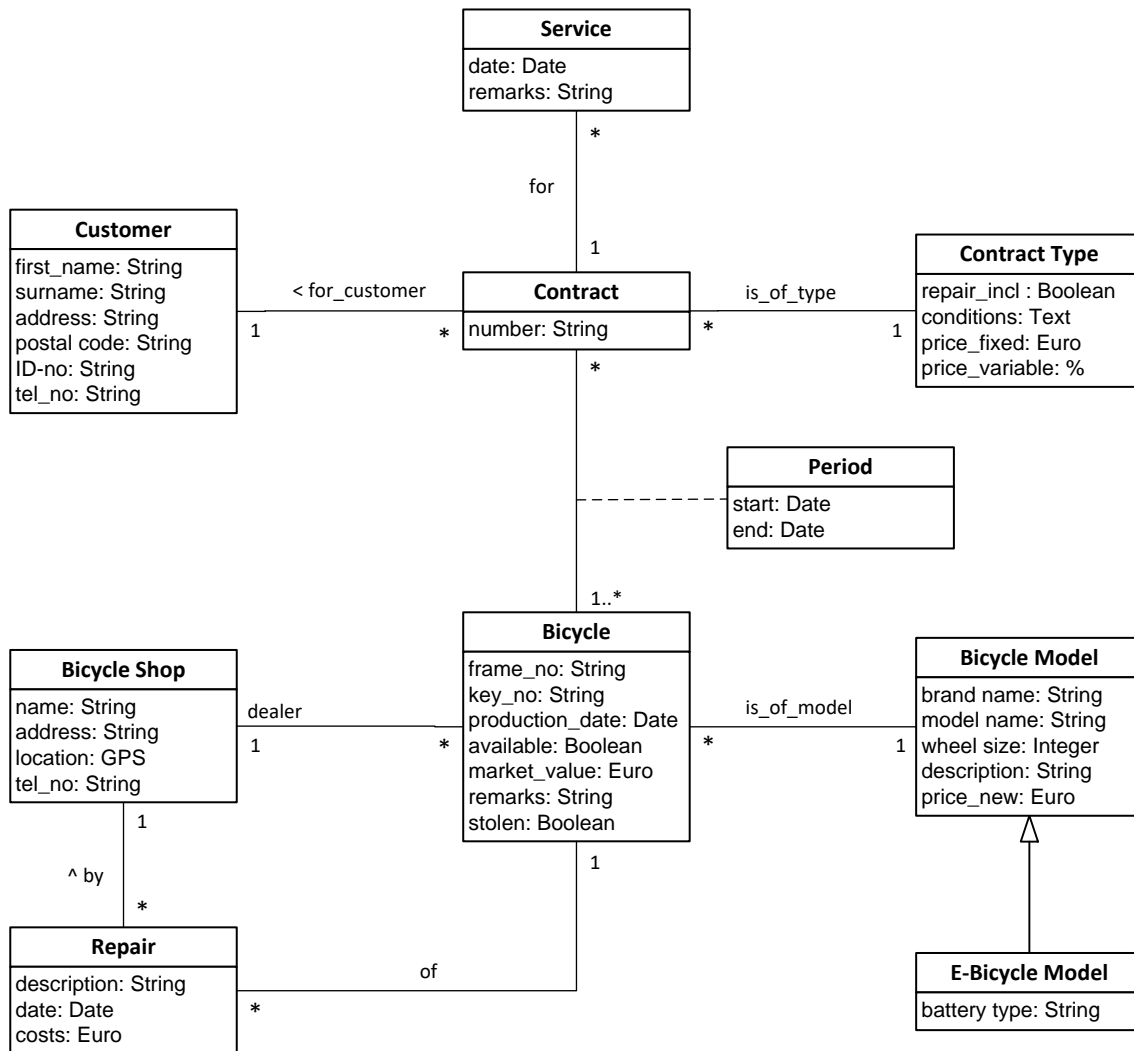
Question 2 (Use Cases) [1.5 points]**2a (Actor list)**

Actor	Description
Customer	Person who leases a bicycle
Dealer	Bicycle shop which leases the bicycle to the customer
Repair shop	Bicycle shop which carries out repair. Could be dealer or other bicycle shop
System clock	Sends notifications about service

2b (Use case diagram)**Remarks:**

- Please note the generalization: A Dealer can do all the activities of a Repair shop, but not the other way round. (Dealer and Repair shop are different roles of a bicycle shop. Not every bicycle shop who does repairs is also a dealer of lease bicycles.)
- Replace bicycle is a proper extension. High-end customers get a new bicycle every few years rather than servicing their current bicycle. This comes with some extra administrative work for the dealer.

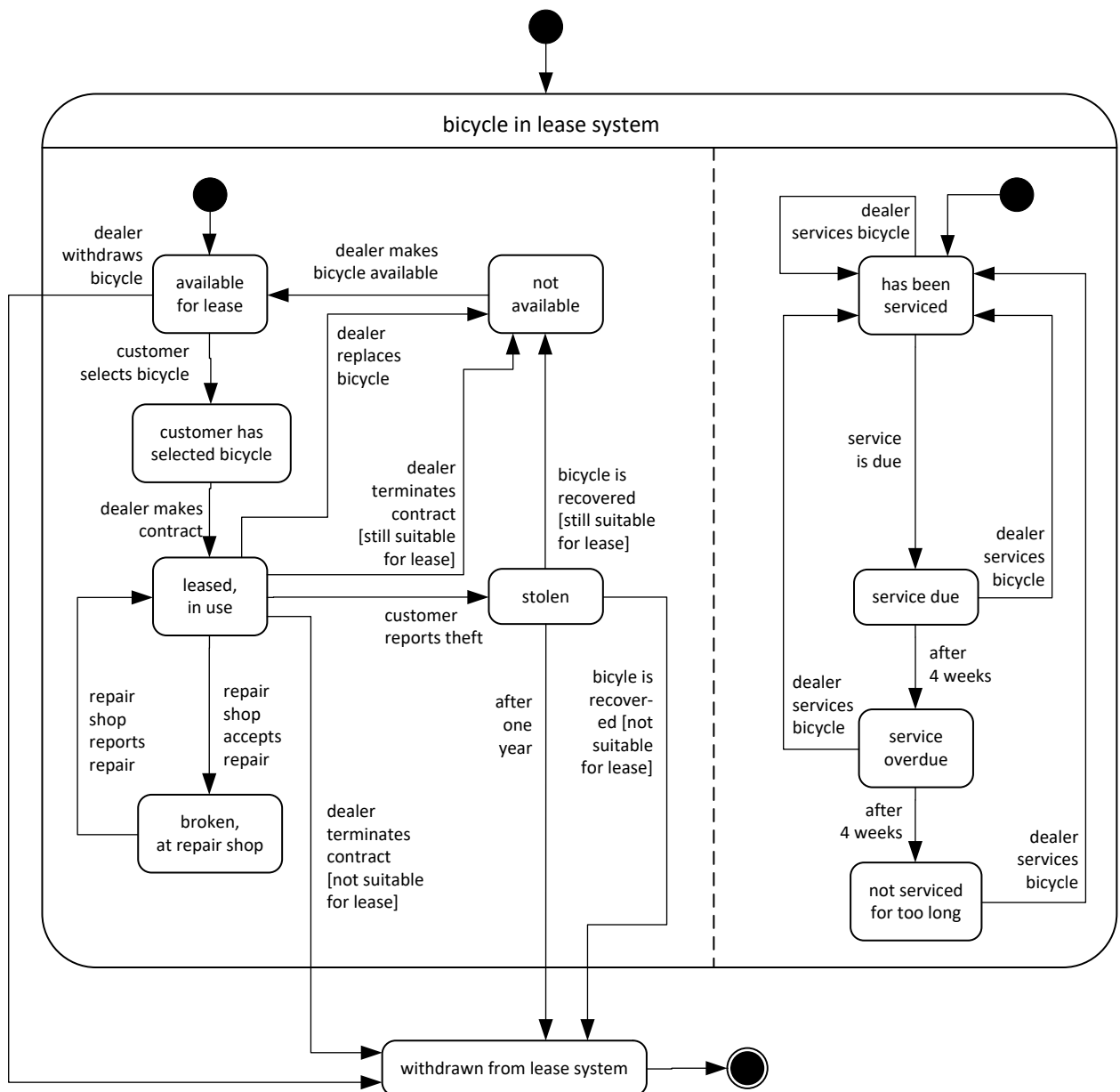
Question 3 (Class Diagram) [3 points]



Remarks:

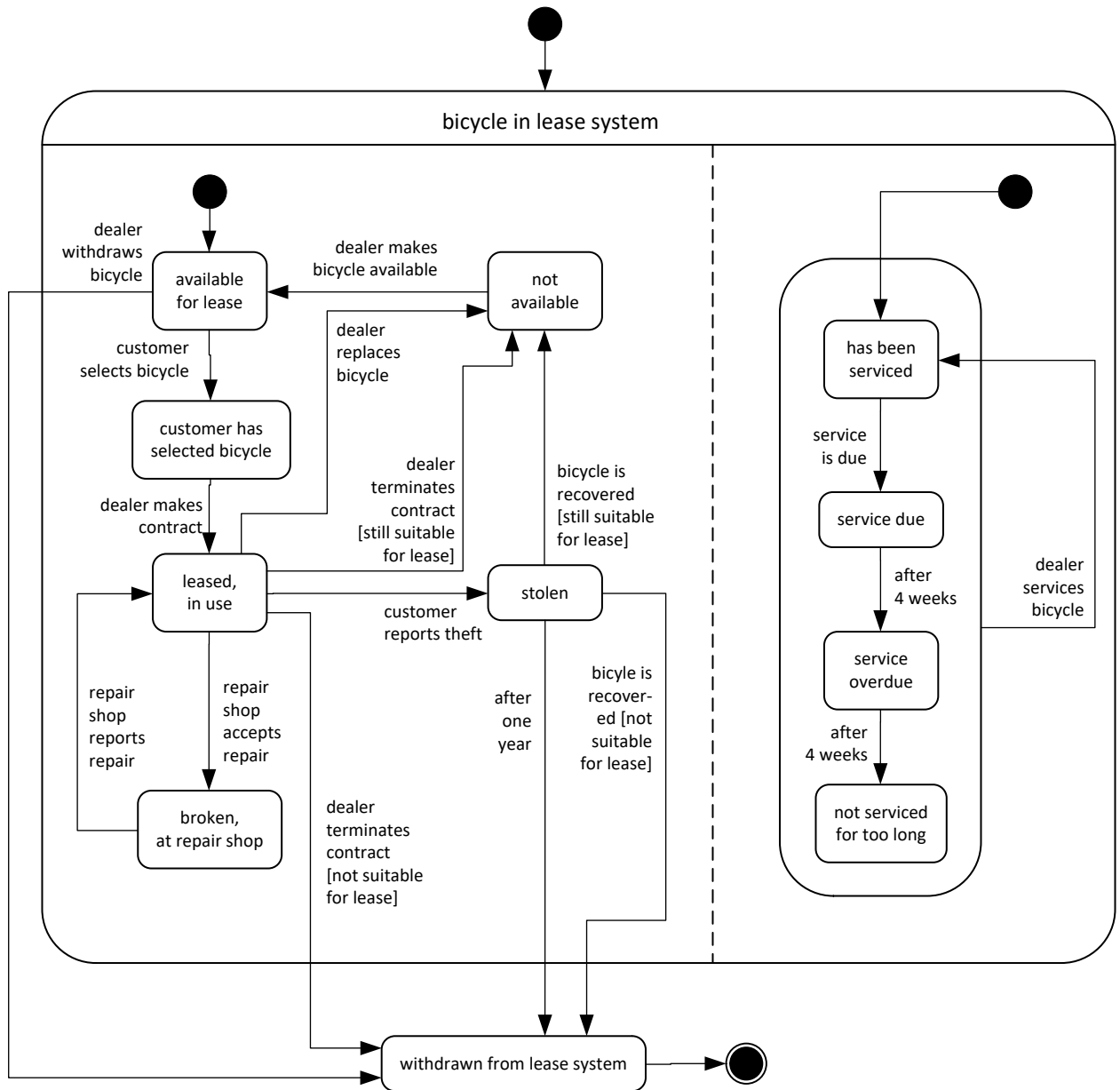
- There are *two* is-of-type relations: *Contract* vs. *Contract Type* and *Bicycle* vs. *Bicycle Model*. In both cases, hints were given in the text (e.g. "standard contracts", "individual contracts", various attributes of a bicycle model) that should be clear enough if you are aware of this kind of construct (see *Car* vs. *Car Type* in the slides).
- *Period* is an association class, because there is exactly one period for the combination of a contract and a bicycle. There can be different bicycles under one contract (high-end contract with bicycle replacement) and different contracts for one bicycle (release of a previously leased bicycle), hence modelling start and end date as attributes of contract or attributes of bicycle is not specific enough.
- If *Period* is correctly modelled, it does not matter whether *Service* and *Repair* are associated with *Contract* or with *Bicycle Service*. In either case, it is clear which object of the other class is meant; from the repair date it is known in which period the repair took place.
- There is no subclass for *Regular* (i.e., non-E) *Bicycle Model*; it is not needed because it has no attributes and no associations.
- Note that *dealer* is the role name of the *Bicycle Shop* in the association with *Bicycle*.

Question 4 (State Machine) [2.3 points]



Remarks:

- It was stated that servicing is independent from the lease status and intermediate repairs, so it makes sense to model this in parallel. If the intermediate repairs were not there, it would have been possible to model the states related to service as a substate of "leased, in use". The presence of the state "broken, at repair shop" makes this infeasible (or it should be handled in parallel within the composite state "leased, in use", service could become due according to the schedule when the bicycle is at a repair shop).
- The right part of the diagram has multiple transitions "dealer services bicycle". The number of transitions could be reduced by introduction another composite state, as in the diagram below.

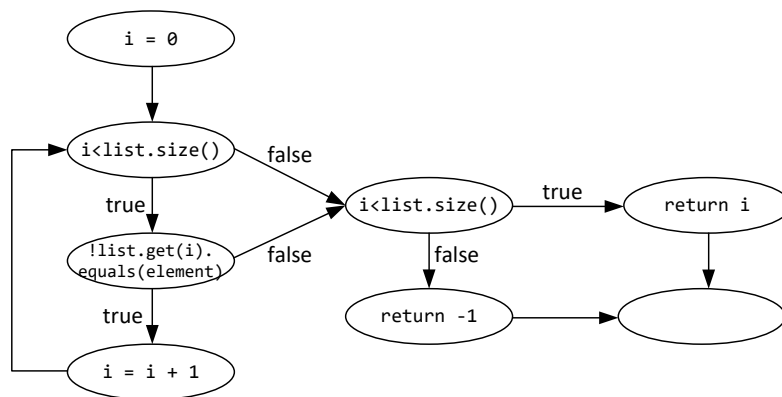


Question 5 (Software Metrics) [1.2 points]**5a (Coupling)**

1. High CA of a class C means that it is hard to know all the effects in other classes of a change in C , so you'd better not change it if you can avoid it.
2. High CE of a class C means that there is a potential lack of stability, as changes in classes on which C depends may cause errors.
Alternatively: Dependency on a lot of other classes makes C harder to test.

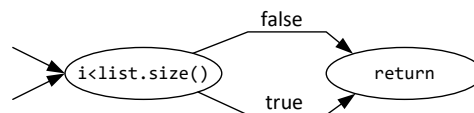
5b (Cyclomatic complexity)

3. A flow graph can be drawn as follows:



We find #edges = 10, #nodes = 8, $CC = E - N + 2 = 4$

Alternatively, the last part of the graph can be simplified as shown below, yielding #edges = 8, #nodes = 6, $CC = E - N + 2 = 4$



4. The test `i < list.size()` is included twice. The test at the end of the method can be eliminated by a return from within the loop if `list.get(i).equals(element)` evaluates to true:

```

public int getIndexOf(List<String> list, String element) {
    int i = 0;
    while (i < list.size()) {
        if (list.get(i).equals(element)) {
            return i;
        }
        i = i + 1;
    }
    return -1;
}
  
```

The resulting flow graph would have 2 binary decision nodes, rather than 3, yielding $CC = 3$.