Com S 362 Practice Mid-Term Solution

The answers offered here are “good enough” to have received full credit. They are intended to give you an indication of what is generally expected. In some cases (especially in the domain model for the ride-sharing app) we have included more detail than required so that there is a better chance that your answer will overlap with some of the presented answer.

In many cases (the Domain Model is a perfect example), there is not one single correct answer. Thus, generally you will be wasting time if you try to divine exactly why the given answer is better than yours; they might very well both be good enough. If you answer is substantially different, you might benefit from reviewing the definitions and criteria related to the topic and ask if your answer meets those as well as the given answer. If the answer is yes, you should move on.

Com S 362 Practice Mid-Term Solution

1. (20 pts, 2 pts each) For each term on the left, select the best description on the right. Put the letter associated with the description in the blank adjacent to the term.

_i_ accidental complexity

_m_UML association

_n_duplicate code

_f_functional design paradigm

_k_CRC Cards

_a_UML

_d_ scenario

_l_product owner

_h_EARS

_c_Noun-phrase analysis

a. A methodology-independent graphical modelling language.
b. A design paradigm that focuses on sequential blocks of instructions.
c. A technique for identifying candidate domain concepts.
d. A specific example of system usage.
e. The long term consequence of fixing bugs.
f. Related to mathematical composition.
g. An analytic technique that simplifies complex systems by focusing on important aspects while obscuring or ignoring less important details.
h. A specific syntax for writing clear requirements.
i. Problems created or aggravated by technological choices and, thus, which are possible to 'fix' with improved technology.
j. A retail customer.
k. Introduced in a paper by Kent Beck.
l. A scrum role – the person who determines what features will be included in the software under development.
m. A line between two boxes.
n. A code smell.

2. (20 pts, 4 pts each) Complete these statements by filling in each of the blanks with the appropriate word or phrase.
   a. Dijkstra introduced _____Separation of concerns________ to the vocabulary of software engineers in his essay "On the Importance of Scientific Thought."

   b. A __domain model____ contains conceptual classes, associations between conceptual classes, and attributes of a conceptual class.

   c. The goal of design is to create systems that can be easily _modified___. (also maintained, changed, enhanced, updated, etc.)

   d. The ____(primary) actor_____________ in a use case initiates an interaction with the system.

e. __expert____________________ is one of the five responsibility assignment patterns included in GRASP. (also creator, high cohesion, low coupling, controller)

3. (20 pts, 4 pts each) Compare and contrast each of the following terms. You will *not* receive credit for attempts to define or describe the meaning of the terms. You *must* identify (succinctly) some meaningful commonality they share (both start with "s" doesn't count) and some difference(s) that distinguish(es) them.

a. Object Oriented Analysis vs Object Oriented Design
Similarity: Both employ/construct models that are framed in terms of concepts/classes/objects.

Difference: Analysis is an investigation of the problem space. Analysis models describe the system as it or the problem.

Design models describe potential solutions; it is a directed exploration of the solution space and focused on the "system to be."

b. Procedural Design Paradigm vs. Modular Design Paradigm

Similarity: Both are in the imperative design class.

Difference: procedural design focuses on identifying cohesive sequences of actions or steps and "wrapping" them in subroutines … thus subroutines tend to be the focus of procedural designs. Encourages the designer to think in terms of tasks. Modular collects both data and actions into coherent functions with well defined interfaces. Encourages the designer to think in terms of required functionalities.

c. Refactoring code vs. rewriting code

Similarity: Both are approaches to revising an existing body of code … usually to add or change functionality.

Difference: Refactoring aims to adjust design decisions without changing functionality, until the design is such that desired changes in functionality can be accomplished more easily. Refactoring focuses on keeping the change well-controlled by working in a sequence of small changes, each of which must pass the pre-existing test suite before further change is made. Refactoring attempts to achieve necessary design changes with minimal change to existing code. A technique for managing risk.

Rewriting often attempts to implement the full change, regardless of scope or difficulty, in a single pass. Rewriting implicitly invalidates existing tests, meaning that it is more difficult to verify the correctness of the change. Rewriting often aims to achieve sweeping changes in design and coding style. An approach that adds significant risk to a project.

d. Use Case vs. Scenario

Similarity: Both are concepts used early in the analysis stage to explore and represent key interactions between external actors and the software system.

Differences. A Use Case tends to be mostly a label used to enumerate key interactions. The scenario is a narrative story describing what happens in the course of satisfying/completing the interaction.

Scenarios are finer grained than Use Cases; a Use Case can encompass multiple scenarios.

e. Object vs. Class

Similarity Both are used to describe or represent certain aspects of an conceptual element – for example a square figure or a desk.

Differences: a class defines a category of such elements. For example all square figures. An object identifies a specific instance of a class, such as a square figure drawn at coordinate 0,0 with sides 50 pixels long.

Refer to the following set of scenarios to answer problems 4 through 10. These scenarios are related to use cases in a partially complete design for a ride sharing app.

Scenario 1: Happy path scenario for use-case "Registered Driver schedules trip"

A driver, who has previously registered with the ride sharing service, authenticates with the system, and supplies a trip plan. The trip plan describes the driver's vehicle, the intended start location and destination, the intended departure time, the driver's required arrival time, the amount of time the driver is willing to add to the trip to accommodate a rider, and which (of the pre-defined cost sharing arrangements) the driver prefers. The trip is now eligible to be matched with potential riders.

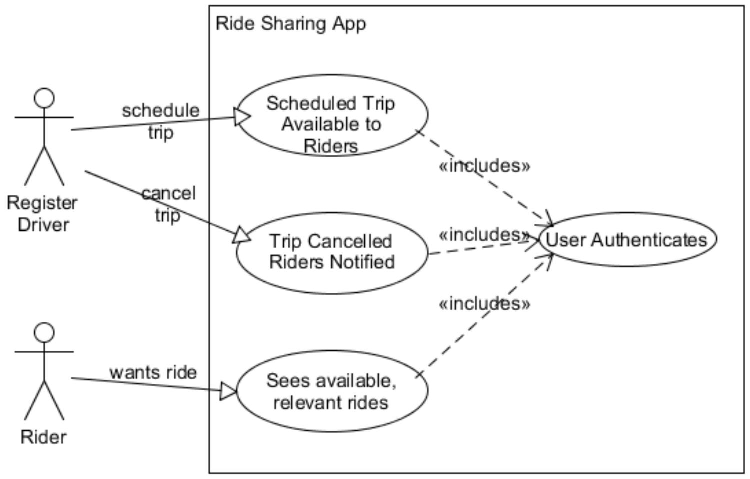
Scenario 2: Happy path scenario for "Rider looks for ride."

The rider, who has previously registered with the ride sharing service, authenticates with the system, and supplies a trip plan. The trip plan describes the intended start location and destination, the intended departure time, required arrival time, the rider's degree of flexibility in departure time (in terms of time before and time after). The system displays general information about drivers with potentially matching travel plans, including type of vehicle, departure and time en route, cost sharing preferences, and historic rider ratings for the driver.

Scenario 3: Happy path scenario for "Driver cancels trip."

A driver who has previously scheduled a trip, authenticates with the system and requests to cancel a scheduled trip. The system displays a list of the driver's currently scheduled trips. The driver selects one of more trips from the list, and confirms that the selected trips are to be cancelled. The system marks the associated trips cancelled and notifies any riders who had expressed interest in the trip of the cancellation.

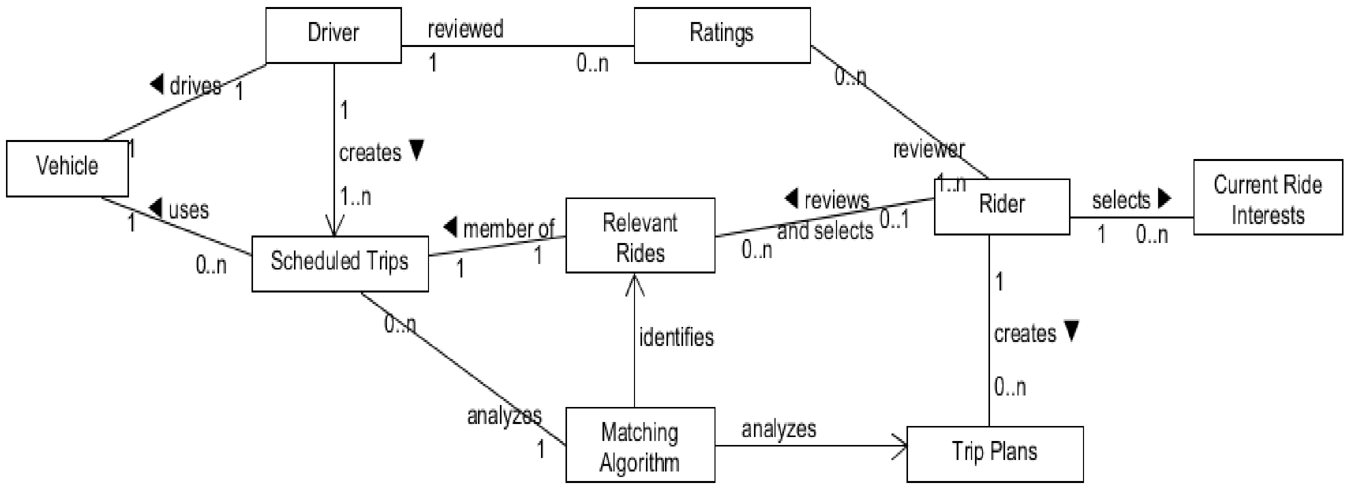
4. (10 pts) Create a use case diagram based on the three brief format scenarios above. You are not expected to complete the design – just create the use case drawing as if these scenarios were the only functions to be included in the app.



5. (7 pts) From the ride sharing scenarios, use noun phrase analysis to identify at least seven potential domain objects/concepts.

   Registered Driver
   Rider
   Start Location
   Destination
   Departure time
   Arrival Time
   Scheduled Trip
   Trip Plan
   Cost Sharing
   Potential Match
   Vehicle
   Time en route
   Driver Ratings
   Authentication Credentials

6. (20 pts) Create a domain model for the partial ride sharing design include at least five domain objects and at least three annotated associations among them.



Note: this is much more complete than required by the question, but still leaves much unspecified.

7. (15 pts) Find three requirements in the scenarios (at least one that is event triggered, and at least one that is state-triggered) and write them in EARS syntax.

When a driver cancels a trip, the system shall notify all riders who have selected the trip (expressed an interest) of the cancellation.

While a trip is still scheduled, the matching algorithm shall consider that trip when seeking potential matches to rider plans.

When the matching algorithm (the matcher) identifies a potentially relevant ride, the matcher shall display driver, vehicle, rating, and schedule information to the associated rider.

8. (20 pts) Find five responsibilities in the scenarios. Assign a name (BAD) to each responsibility and write a brief description of each.

   1. Track Scheduled Trips: keep track of all trips and related trip details scheduled by drivers.
   2. Rider Interests: keep track of all trips currently planned by riders.
   3. Remember Rider Interests: Remember which scheduled trips riders have selected as rides they are interested in.
   4. Notify Riders of cancellation: Notify interested riders when a drive cancels a trip they have selected.
   5. Track Ratings: Remember ratings riders have given to drivers in the past.

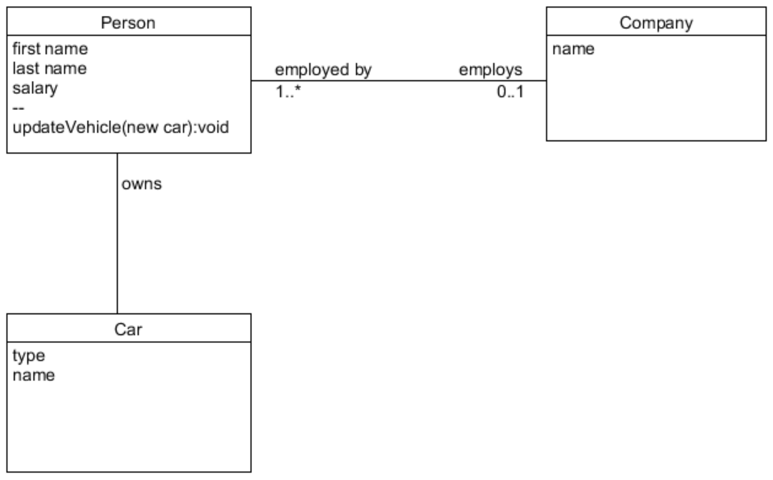
9. (6 pts) Pick three of the named responsibilities in 5 and tell which object in 3 you would assign that responsibility to. Justify each choice in terms of one or more of the GRASP responsibility patterns.

   Track Scheduled Trips → Scheduled Trips: Expert
   Rider Interests → Current Ride Interests: Expert
   Track Ratings → Ratings: Expert

10. (5 pts) Draw and fill in a CRC card for one of the objects to which you assigned a responsibility in problem 9.

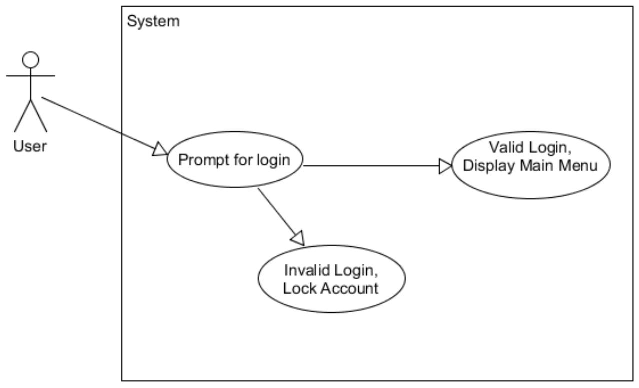
Class: Ratings	
Responsibilities: • Remember Ratings previously given to drivers by riders • Compute driver statistics	Collaborations: • Rider: to receive new ratings • Driver to review current statistics and new ratings • Relevant Rides – to supply driver rating information for use in trip description • Scheduled Trips – to get details about rated trip as part of recording a new rating.

11. (4 pts) What is wrong with this Domain Model?



Specifics of an object programming interface are not part of a domain model. (i.e., “updateVehicle(…)” should not be included.

12. (4 pts) What is wrong with this Use Case Drawing?



It is showing steps, not use cases. It is focused on steps and not on the user intent and the goal of the interaction. It gives no information about what the user trying to accomplish.