Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto and Matteo Rossi

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering II

June 25th, 2019

Last Name

First Name

Id number (Codice Persona or Matricola)

Note

1. The exam is not valid if you do not fill in the above data.
2. Write your answers on these pages. Extra sheets will be ignored. You may use a pencil.
3. Incomprehensible hand-writing is equivalent to not providing an answer.
4. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden.
5. You cannot keep a copy of the exam when you leave the room.
6. The exam is composed of three exercises. Read carefully all points in the text!
7. **Total available time: 2h**

Scores of each question:

Question 1	(MAX 6)	________
Question 2	(MAX 7)	________
Question 3	(MAX 3)	________

Question 1 Alloy (6 points)

Consider the following behavior of a vending machine.

The vending machine accepts 3 types of coins: 25¢ (cents), 50¢, and 1$. It accepts coins only in ascending order, for example, after inserting a 50¢ coin, it can only accept 50¢ or 1$ coins and it cannot accept 25¢ coins until the process is restarted by asking for the remaining change.
Every time the amount of money in the machine reaches (or surpasses) 1$, it produces a bottle of milk, subtracts 1$ from the amount of money that is in the machine, and leaves the rest in the machine.
At any time, the user can ask for the money still in the machine (and not used to buy a bottle of milk) to be returned (this can occur even if there is no money remaining in the machine); the effect of this, as mentioned above, is that the process is restarted, so smaller coins can be introduced again.

You are given the following signatures:

```
abstract sig Coin {}
one sig Quarter extends Coin {} // represents a 25¢ coin
one sig Half extends Coin {} // represents a 50¢ coin
one sig Dollar extends Coin {} // represents a 1$ coin

abstract sig AmountMoney {} //represents a collected amount of money
one sig Tot0 extends AmountMoney {} // represents an amount of 0$
one sig Tot25 extends AmountMoney {} // represents an amount of 25¢
one sig Tot50 extends AmountMoney {} // represents an amount of 50¢
one sig Tot75 extends AmountMoney {} // represents an amount of 75¢
```

In addition, suppose that the function
`computeRest[ am : AmountMoney, c : Coin ] : AmountMoney`
has been defined. This takes as input an amount of money `am` and a coin `c` and, if the sum of `am` and the coin value is greater than or equal to 1$, it returns the rest of the sum with respect to 1$; otherwise, if the sum is less than 1$, it simply returns the sum.

Formalize in Alloy the following elements:

a) A signature `VendingMachineState` describing the state of the vending machine.
b) A predicate `leqCoin` that takes as input two coins, `c1` and `c2`, and returns true if `c1` is less than, or equal to, `c2`, false otherwise.
c) A predicate describing operation `insertCoin`, which, given a vending machine, takes the value of the inserted coin, and changes the state of the vending machine accordingly.
d) A predicate describing operation `getMoneyBack`, which, given a vending machine, changes its state and produces an `AmountMoney` with the amount of money that was present in the machine.

Note: you are not required to model the actual production of milk, but only to focus on the amount of money and coins handled by the vending machine.

Solution
A possible solution for the exercise is the following. Other ones are also possible, of course.

```
sig VendingMachineState {
  money: AmountMoney,
  minCoin : Coin
}

pred leqCoin[ c1, c2 : Coin ] {
  c1 = Quarter or
  c1 = Half and not (c2 = Quarter) or
  c1 = Dollar and c2 = Dollar
}
```

```
pred insertCoin[ s, s': VendingMachineState, c : Coin ] {
  leqCoin[s.minCoin, c] and
  s'.money = computeRest[s.money, c] and
  s'.minCoin = c
}

pred getMoneyBack[ s, s': VendingMachineState, res : AmountMoney ] {
  s'.money = Tot0 and
  s'.minCoin = Quarter and
  res = s.money
}
```

Question 2 Design and JEE (7 points)

Consider a system for sharing car rides. In the system there are two kinds of car users: people who offer rides (i.e., *drivers*) and people who use shared rides (i.e., *passengers*). Each person is identified by his/her name, birth date, and address. Drivers, in addition, have a driving license (identified by its number) and a reputation (a value between 0 and 4, where 4 indicates that the driver offers an excellent service). Drivers offer shared rides, while passengers take them. A shared ride corresponds to a path between two locations, an origin and a destination, which is driven in a specific date, starting from a specific time of the day. Multiple passengers can take the same shared ride. Each passenger can rate the driver at the end of the ride. Each individual rating impacts on the reputation of the driver, which is computed as the average of the ratings given by all passengers to whom the driver has offered a ride in the last year.

Drivers and passengers exploit a mobile-friendly web-based application to interface with the system. The application allows drivers to insert in the system the information of a shared ride (origin, destination, date, time). Dually, a passenger can use the application to retrieve the list of available rides from a certain origin to a certain destination in a certain date and time, together with the reputation of the corresponding driver, and to select one of these rides. When a passenger requests to share a ride, the request is put on hold and the system notifies the driver, who can then accept or reject the request. Finally, the system sends a confirmation to the passenger. In the timeframe between the time the request has been issued and the time it has been accepted/rejected by the driver, the passenger cannot request another shared ride.

Assume to use JEE and, in particular, Enterprise Java Beans for the implementation. Answer to the following questions:

A. List the **entities** that are needed to implement the system, define their attributes, and highlight the relationships between them, indicating the corresponding annotations in JEE (e.g., @OneToMany).
B. Assume that your boss has decided to develop three session beans, one focusing on login/logout operations (`LoginManager`), another supporting the creation and management of rides by drivers (`OfferedRideManager`), and a third one allowing passengers to visualize all relevant rides, select one, receive a confirmation, and rate the driver at the end of the drive (`PassengerAssistantManager`).
   Define the signatures of the methods of bean `OfferedRideManager`, and describe their behavior.
C. Shortly describe a possible solution concerning the interaction between the system and the driver when a passenger selects a ride and the system must receive the confirmation or rejection by the driver.

Solution
The one below is a possible, not the unique solution. We tried to make it as complete as possible. Some of the methods in `OfferedRideManager` are not strictly required by the problem description.
Point A
The entities to be defined for this system are `Person`, `Driver` and `SharedRide`.

Person has attributes userId (Integer, @GeneratedValue), name (String), birthdate (Date), address (String), onHoldRide (@ManyToOne relationship with the selected and not confirmed yet SharedRide), and bookedRides (@ManyToMany relationship with SharedRide).
Driver inherits from Person. It has the following additional attributes: licenseNumber (String), activeRides (@OneToMany relationship with SharedRide. This is to refer to all rides that are still offered to passengers) and pastRides (@OneToMany relationship with SharedRide to refer to all rides that have already happened in the past).
SharedRide has the following attributes: rideId (Integer, @GeneratedValue), origin (Coordinates), destination (Coordinates), departure date and time (Date and DayTime), acceptedPassengers (this is a @ManyToMany relationship with Person), driver (this is a @ManyToOne relationship with Driver).

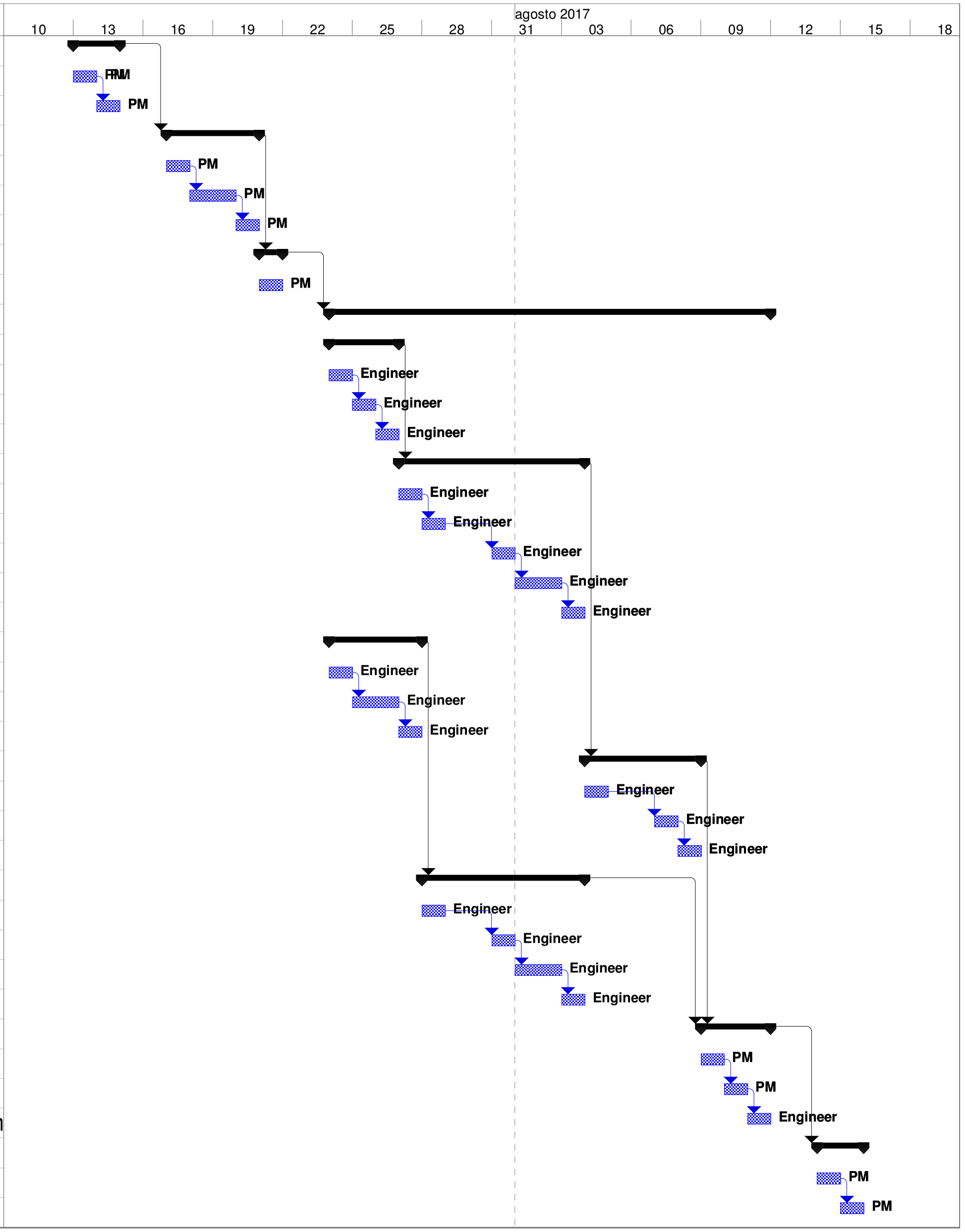
Point B
OfferedRideManager could offer the following methods:
Integer createRide (Coordinates origin, Coordinates destination, Date date, Integer driverId): this method allows drivers to create a ride. It assumes that Coordinates, Date and DayTime are serializable classes. It returns a rideId.
boolean confirmRide(Integer rideId, Integer passengerId, Integer driverId): this method is used when the driver confirms that he/she will give a ride to the specified passenger
boolean rejectRide(Integer rideId, Integer passengerId, Integer driverId): this method is used when the driver rejects a passenger for a ride.
List <SharedRide> getMyRidesHistory(Integer driverId): this method is used to retrieve the information about all rides by a certain driver.
boolean CloseARide(Integer rideId, Integer driverId): this method is called when the driver decides that a ride is full and no other passengers can join.
boolean DeleteARide(Integer rideId, Integer driverId): this method is called when the driver decides to eliminate a ride.

Point C
We can address the problem by having the system sending an email or text message to the driver with a link that brings him/her to the acceptance/rejection page. The web layer in charge of acquiring the user input will then create a connection with the OfferedRideManager session bean and call either the confirmRide or the rejectRide.

Question 3 Project management (3 points)
While replacing a local banking system with a multi-country one, it is required to implement the application that covers the Italian regulatory obligations.
The Gantt diagram in the next page is the project schedule defined for the implementation of this application. The diagram represents the tasks with their mutual relationships, the summary tasks are those in black covering the corresponding sub-tasks; the arrows between the summary tasks represent the predecessor-successor relationships (for example task 10 cannot start before task 8 is completed).
Tasks can be identified by their task number (#) which is the task line number. For simplicity, all tasks have duration of 1 day, except for some, which have a length of 2 days. The tasks lasting 2 days are task # 6, task # 19, task # 23 and task # 32; they can be crashed by one day. Crashing a 2-day task by one day means doubling the resources working on that task.
There are two type of resources: Project Manager (PM) and Software Engineer (Engineer); the cost of PMs is 450 euros/day, whereas the cost of Engineers is 350 euros/day. Each task is associated with a type of resource as shown in the Gantt diagram.

Please answer the following questions (motivate your answers):
A: Define the critical path indicating all its component task numbers (#).
B: Reduce the duration of the project by one day using the crashing technique for one task; please explain:
1) which are the tasks that could be crashed to reduce the project length and why;
2) which is the task to be actually crashed if one wanted to have the least impact on the budget.

ID	Task Name
1	**Project setup**
2	definition of the project organization
3	overall business case
4	**Poject initiation**
5	project documentation
6	detailed business case
7	Systems gap analysis
8	**Planning phase**
9	Project plans
10	**Project execution**
11	**Enrichment data base**
12	design
13	implementation
14	technical test
15	**Enrichment online, automatic and batch programs**
16	specifications
17	development
18	technical test
19	Historical data base population
20	Enrichment system functional testing
21	**COTS reporting system installation**
22	technical specification received from the provider
23	Iseries test environment setup
24	functional testing
25	**Develop extraction application**
26	specifications
27	development
28	functional testing
29	**Develop network interface**
30	test environment setup
31	functional specifications
32	development
33	functional testing
34	**User Acceptance Testing**
35	Business using old and new system
36	Compare reports produced with old and new system
37	Implementation syncronized with CORE system activation
38	**Closing Project**
39	user, techncal and support reference documentation
40	release resources



Solution

A Critical path: tasks # 2 – 3 – 5 – 6 – 7 – 9 – 12 – 13 – 14 – 16 – 17 – 18 – 19 – 20 – 26 – 27 – 28 – 35 – 36 – 37 – 39 – 40. Tasks 21 and 29 (together with their sub-tasks) are not on the critical path as they can be started independently from the others, provided that the company has enough resources.

B: tasks that could be crashed for reducing project length are those on the critical path, so, tasks # 6 and 19.

Crashing a task means allocating a new resource working in parallel with the others already allocated to the project. This also implies that we are going to have the resources free at the end of the crashed task, for the amount of time we have saved with crashing. If such free resources cannot be reallocated to a different project, crashing results in an extra cost due to the intervention of the new resource working in parallel with the others, otherwise, the cost of the project remains the same. Assuming to be in the first case, since task #19 is less expensive than task #6, as it is performed by an Engineer, we should consider crashing that task.