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

September 13th, 2018

Last Name
First Name
Id number (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 8) _____

Question 2 (MAX 8) _____

Question 3 (MAX 3) _____

Question 1 Alloy (8 points)
Consider an application that recommends to users what movies to rent or buy; the application, given the history of rentals and buys of the user, recommends what other movies could be of interest for her/him.
Given the application outlined above, consider the following world phenomena:

- Every user has seen some movies.
- Every user likes some movies (it is not necessary that a user has seen a movie that she/he likes; also, a user might have seen movies that she/he does not like).

Consider also the following shared phenomena:

- Each user has bought some movies.
- Each user has rented some movies.
- Each user rates as excellent, good, or bad every movie she/he has bought or rented.

Finally, consider the following machine phenomena:

- Each movie has a set of movies that are similar in nature; the notion of "similarity" is symmetric (if movie *A* is similar to movie *B, B* is also similar to *A*), and it is used to recommend movies to people, but it is not specified any further.

Formalize through an Alloy model:

- The world, shared and machine phenomena identified above.
- A fact capturing the domain assumption D1 that a user rates as excellent or good exactly the movies that she/he likes.
- A fact capturing the domain assumption D2 that a user has seen exactly all movies that she/he has bought or rented.
- A predicate canBeRecommended that is true when a set of movies can be recommended to a certain user. A set can be recommended to a user if it contains at least 1 and at most 20 movies that the user has not seen, and that are similar to movies that she/he has positively rated (excellent or good).

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

```
sig Movie {
  similar : set Movie
}

fact symmetrysimilar { similar = ~similar }

abstract sig Rating {}
one sig Bad extends Rating{}
one sig Good extends Rating{}
one sig Excellent extends Rating{}

sig User {
 hasSeen: some Movie,
 likes: some Movie,
 bought: some Movie,
```

```
  rented: some Movie,
  ratings: Movie -> lone Rating
}{ ratings.Rating = (rented + bought) }


fact D1 {
  all u : User | u.ratings.(Good + Excellent) = u.likes & (u.bought + u.rented)
}

fact D2 {
  all u : User | u.bought + u.rented = u.hasSeen
}

pred Recommend[ u : User, res : set Movie ]{
  #res > 0 and #res =< 20 and
  res in (Movie - (u.bought + u.rented)) & u.ratings.(Good + Excellent).similar
}
```

Question 2 JEE (8 points)

A food delivery website allows users to look at the menus of various restaurants and to place delivery orders. In particular a user should be able to:

1) Register to the website
2) Login into the system
3) Look at the list of available restaurants
4) Look at the menu of a specific restaurant
5) Select the food items which compose her/his order at a specific restaurant.
   NOTE: within a user session, if the user starts filling up an order, but then leaves the page for any reason and comes back later for finishing the order, she/he should be able to continue from where she/he left.
6) Forward an order to a restaurant through the delivery website
7) Leave a review for a restaurant
8) Read reviews left by other users for a restaurant

You are asked to design the software system using JEE. In particular:

A. You should identify the required JEE entity/session beans that you wish to have in order to implement the system. For each bean, you should specify the type (e.g., stateful session bean) and the requirements it is intended to fulfill (among those listed above). To ease our assessment of your solution, please use the table below.
B. Specify **at least 2 methods** for each bean choosing some significant functionality of the bean. In particular, provide the signature of the method, a description of what it does, and list the JEE features it exploits.

Solution

Bean name	Bean type	Requirements it contributes to fulfill
RestaurantEntity	JPA Entity bean	3), 4), 5), 6), 7), 8)
DishEntity	JPA Entity bean	4), 5)
OrderEntity	JPA Entity bean	5), 6)
ReviewEntity	JPA Entity bean	7), 8)
UserEntity	JPA Entity bean	1), 2)
OrderBean	Stateful session bean	5), 6)
UserBeans	Stateless session bean	1), 2)
RestaurantBean	Stateless session bean	3), 4), 7), 8)

Methods provided by UserEntity:

- String getUsername()/void setUsername(String name) : getter and setter for the user name. The user name is also the primary key for a user and thus the corresponding attribute is annotated with the @Id JPA annotation.
- String getPassword()/void setPassword(String password) : getter and setter for the password of a user.
- other getters and setters for other personal information

Methods provided by ReviewEntity:

- Integer getId()/void setId(Integer id) : getter and setter for the id (which we assume to be automatically generated by the JPA using the @GeneratedValue annotation) of the review
- String getText() /void setText(String text) : getter and setter for the body of the review
- UserEntity getAuthor()/void setAuthor(UserEntity author) : getter and setter for the review's author. The relation between a review and its author is specified using the @ManyToOne annotation of the JPA.

Methods provided by RestaurantEntity:

- Integer getId()/void setId(Integer id) : getter and setter for the id (which we assume to be automatically generated by the JPA using the @GeneratedValue annotation) of the restaurant
- String getName()/void setName(String name) : getter and setter for the name of the restaurant
- String getMail()/void setMail(String mail) : getter and setter for the mail of the restaurant (under the assumption that orders are forwarded to restaurants via email)

- List<ReviewEntity> getReviews()/void setReviews(List<ReviewEntity> reviews) : getter and setter for the reviews of a restaurant. The relation between a restaurant and the list of corresponding reviews is the @ElementCollection annotation of the JPA.

Methods provided by OrderEntity:

- Integer getId()/void setId(Integer id) : getter and setter for the id (which we assume to be automatically generated by the JPA using the @GeneratedValue annotation) of the order
- UserEntity getUser()/void setUser(UserEntity user) : getter and setter for the user who specified the order. The relation between an order and its user is specified using the @ManyToOne annotation of the JPA.
- RestaurantEntity getRestaurant()/void setRestaurant(RestaurantEntity restaurant) : getter and setter for the restaurant with which the order is associated. The relation between an order and its restaurant is specified using the @ManyToOne annotation of the JPA.
- RestaurantEntity getRestaurant()/void setRestaurant(RestaurantEntity restaurant) : getter and setter for the restaurant with which the order is associated. The relation between an order and its restaurant is specified using the @ManyToOne annotation of the JPA.
- Set<DishEntity> getDishEntity()/void setDishEntity(Set<DishEntity> dishEntity) : getter and setter for the dishes with which the order is associated. The relation between orders and set of dishes is specified using the @ManyToMany annotation of the JPA.

Methods provided by DishEntity:

- String getDishName()/void setDishName(String name) getter and setter for the name of the dish
- we may have other getters and setters for other features of a dish

Methods provided by UserBean:

- UserEntity registerNewUser(String userName, String password). It creates a new UserEntity and uses an EntityManager to persist the new user into the database. It returns the persisted user. If a user with the same username does already exist it throws an exception.
- Boolean loginUser(String userName, String password). It queries the database (using an EntityManager) looking for a user with the specified username. If this exists, it compares the corresponding password with the one received as argument. If they match, the method returns true. In any other case it returns false.
- Boolean existsUser(String userName). It queries the database (using an EntityManager) looking for a user with the specified username and it returns true if this exists, false otherwise.
- UserEntity getUserById(Integer userId). It queries the database (using an EntityManager) looking for a user with the specified id and it returns the corresponding UserEntity if this exists, null otherwise.

Methods provided by RestaurantBean:

- RestaurantEntity getRestaurantByName(String name). It queries the database (using an EntityManager) looking for a restaurant with the specified name and it returns the corresponding RestaurantEntity if this exists, null otherwise.
- RestaurantEntity getRestaurantById(Integer id). It queries the database (using an EntityManager) looking for a restaurant with the specified id and it returns the corresponding RestaurantEntity if this exists, null otherwise.
- List<RestaurantEntity> getAllRestaurants(). It queries the database (using an EntityManager) for the list of all the persisted restaurants.
- List<OrderEntity> getOrders(RestaurantEntity restaurant). It queries the database (using an EntityManager) for the list of all the orders sent to the specified restaurant.
- ReviewEntity reviewRestaurant(Integer restaurantId, String content, UserEntity user). It creates a new ReviewEntity for the given user and with the given content and updates in the persistent storage the restaurant having the given id adding the newly created ReviewEntity.

Methods provided by OrderBean:
(We assume for a given session there could be no more than one order per restaurant, so for instance we do not need to specify an order when calling addDishToOrder, but just the restaurant, and the OrderBean will check for the single order currently under definition for the given restaurant. In other words, the OrderBean is not expected to manage multiple orders within a given session and for the same restaurant. An order for a restaurant gets created, filled and sent or removed. Only after doing this it is possible to place another order for the same restaurant. Different implementations could still be acceptable):

- OrderEntity createNewOrder(RestaurantEntity restaurant). It creates a new OrderEntity and associates it with the given restaurant. The order is kept in memory as the conversational state until it is either removed or forwarded.
- OrderEntity addDishToOrder(RestaurantEntity restaurant, String dishName). It queries the database looking for the DishEntity associated with the given restaurant and having the given name. If this exists, it adds the dish to the order kept in memory for the given restaurant and returns the updated OrderEntity. Otherwise, it throws an exception to be handled by the caller.
- OrderEntity removeDishFromOrder(RestaurantEntity restaurant, DishEntity dish). It updates the order kept in memory for the given restaurant by removing the specified dish and returns the updated OrderEntity. If there is no pending order for the given restaurant or if the given dish is not contained in the pending order it throws an exception to be handled by the caller.
- Boolean forwardOrder(OrderEntity order). It performs the business logic necessary for forwarding an order to the corresponding restaurant and returns a Boolean signaling if the forward succeeded or not. In both cases it persists (using an EntityManager) the input order in the database.
- OrderEntity getCurrentOrder(RestaurantEntity restaurant). If there is a pending order for the given restaurant that the OrderBean is keeping in the conversational state, it returns such order (otherwise it throws an exception, which is handled by the caller).

- Boolean existsPendingOrder(RestaurantEntity restaurant). Returns true if there is a pending order for the given restaurant that the OrderBean is keeping in the conversational state, false otherwise.

Question 3: Testing (3 points)
Consider the following fragment of C code, which contains an issue that C compilers typically do not recognize as an error:

```
int a, b;
for(a = 0; a<b; a++)
  printf("Executing the loop...¥n");
```

A. Describe the problem in the code.
B. Consider the various analysis approaches we have seen in the course. For each of them, explain if and how they could be able to discover the problem. Provide a justification also for negative answers.

Solution
Point A: The problem is clearly the comparison a<b where *b* has not been initialized.

Point B:
Manual inspection/code reviews: A typical check list would include a question such as: are all variables properly initialized and therefore would allow the inspectors/reviewers to discover the problem.
Def-use pairs analysis: This technique allows us to spot that there is no definition associated with the use of *b*.
Symbolic execution: with this technique we associate a symbolic value with each variable as soon as it is initialized. In this case, *b* will not acquire a symbolic value and therefore, the symbolic executor will not be able to handle the condition a<b.