Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto, Matteo Rossi

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

Software Engineering II

February 16th, 2018

Last Name
First Name
Id number (Matricola)

Note

1. The exam is not valid if you don't 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. **There are 2 variants of Exercise 3, one dealing with Project Management, and one dealing with Testing. You are required to answer only one of the two variants; if you answer both of them, one will be discarded.**

Scores of each exercise:

Exercise 1 (MAX 8) _____

Exercise 2 (MAX 8) _____

Exercise 3 (MAX 3) **PM** _____ **TEST** _____

Question 1 Alloy (8 points)
NB: Please read the whole exercise text (including the remarks) before starting to work at your solution.

Consider the following scenario:
The company *TravelSpaces* decides to help tourists visiting a city in finding places that can keep their luggage for some time. The company establishes agreements with small shops in various areas of the city and acts as a mediator between these shops and the tourists that need to leave their luggage in a safe place. To this end, the company wants to build a system, called *LuggageKeeper*, that offers tourists the possibility to: look for luggage keepers in a certain area; reserve a place for the luggage in the selected place; pay for the service when they are at the luggage keeper; and, optionally, rate the luggage keeper at the end of the service.

Given the scenario above, consider the following world phenomena:

- Every user owns various pieces of luggage.
- Every user can carry around various pieces of luggage.
- Each piece of luggage can be either safe, or unsafe.
- Small shops store the luggage in lockers, where each locker can store at most one piece of luggage.

Consider also the following shared phenomena:

- Each locker is opened with an electronic key that is associated with it (the electronic key is re-generated at each use of the locker; also, a locker that does not have an electronic key associated with it is free).
- Each user can hold various electronic keys.

Formalize through an Alloy model:

- The world and machine phenomena identified above.
- A predicate capturing the domain assumption `D1` that a piece of luggage is safe if, and only if, it is with its owner, or it is stored in a locker that has an associated key, and the owner of the piece of luggage holds the key of the locker.
- A predicate capturing the requirement `R1` that a key opens only one locker.
- A predicate capturing the goal `G1` that for each user all his/her luggage is safe.
- A predicate capturing the operation `GenKey` that, given a locker that is free, associates with it a new electronic key.

Remarks:

- You are not required to write all requirements and assumptions that make the goal hold, but only to formalize the listed ones.

Solution
In the following you find one of the possible solutions. It does not specify any integrity constraint on the model as this was not explicitly required by the text. Moreover, all predicates except `GenKey` are without parameters. Based on the text description, we could have defined `D1` with a piece of luggage as a parameter and `R1` with an electronic key as parameter. `G1` instead should not have parameters as it should be true for each user and his/her luggage, not for a specific user.

```
abstract sig Status{}
one sig Safe extends Status{}
one sig Unsafe extends Status{}
```

```
sig Luggage{
  luggageStatus : one Status
}

sig EKey{}

sig User{
  owns : set Luggage,
  carries : set Luggage,
  hasKeys : set EKey
}

sig Locker{
  hasKey : lone EKey,
  storesLuggage : lone Luggage
}

sig Shop{
  lockers : some Locker
}

// A predicate capturing the domain assumption D1 that a piece of baggage is safe
// if, and only if, it is with its owner, or it is stored in a locker, and the
// owner of the piece of baggage holds the key of the locker.
pred D1 {
  all lg : Luggage |
    lg.luggageStatus in Safe
    iff
    all u : User | lg in u.owns implies
                      ( lg in u.carries
                        or
                        some lk : Locker | lg in lk.storesLuggage and
                                           lk.hasKey != none and
                                           lk.hasKey in u.hasKeys )
}

// A predicate capturing the requirement R1 that a key opens only one locker.
pred R1 {
  all ek : EKey | no disj lk1, lk2: Locker | ek in lk1.hasKey and ek in lk2.hasKey
}

// A predicate capturing the goal G1 that for each user his/her luggage is safe.
pred G1 {
  all u : User | all lg : Luggage  | lg in u.owns implies lg.luggageStatus in Safe
}
```

```
// A predicate capturing the operation GenKey that, given a locker that is free,
// associates with it a new electronic key.
pred GenKey[lk, lk' : Locker] {
  //precondition
  lk.hasKey = none
  //postcondition
  lk'.storesLuggage = lk.storesLuggage
  one ek : EKey | lk'.hasKey = ek
}
```

Question 2 JEE (8 points)
A web site allows users to post articles and to comment on articles inserted by other users. In particular, a user should be able to:

1) Register to the system
2) Login into the system
3) Post a new article
4) Delete a previously inserted article
5) Browse the list of all articles
6) Browse the list of articles inserted by a given user
7) Read an article and its corresponding comments
8) Write a comment for an article
9) Delete a previously inserted comment

You are asked to design the software system using JEE. In particular, you should identify the required JEE entity/session beans that you wish to have in order to implement the system. For each bean, specify its type (e.g., stateful session bean), the requirements it is intended to fulfill (among those listed above), the methods it provides, what each method does and how (i.e., using which JEE feature). You are **not** required to implement the methods.
To ease our assessment of your solution, please use the following table to list the beans you need, together with their type and corresponding requirements.
Then, for each bean specify **at least 2 methods** that realize a significant functionality of the bean; for each specified method provide an explanation of what it does, and the JEE features it exploits.

Bean name	**Bean type**	**Requirements it contributes to fulfill**

Solution

Bean name	**Bean type**	**Requirements it contributes to fulfill**
`ArticleEntity`	JPA Entity bean	3), 4), 5), 6), 7), 8), 9)
`CommentEntity`	JPA Entity bean	7), 8), 9)
`UserEntity`	JPA Entity bean	1), 2), 6)
`ArticleBean`	Stateless session bean	3), 4), 5), 6), 7), 8), 9)
`UserBean`	Stateless session bean	1), 2)

Methods provided by `UserEntity`:

- `String getUsername()`/`setUsername(String username)` : 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()`/`setPassword(String password)` : getter and setter for the password of a user.
- `List<ArticleEntity> getArticles()`/`void setArticles(List<ArticleEntity> articles)` : getter and setter for the article of an article. The relation between an user and list of corresponding articles is specified using the `@OneToMany` annotation of the JPA.

Methods provided by `CommentEntity`:

- `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 comment.
- `String getTitle()` / `void setTitle(String title)` : getter and setter for the title of the comment.

- `String getText()` /`void setText(String text)` : getter and setter for the body of the comment.
- `UserEntity getAuthor()`/`void setAuthor(UserEntity author)` : getter and setter for the comment's author. The relation between a comment and its author is specified using the `@ManyToOne` annotation of the JPA.

Methods provided by `ArticleEntity`:

- `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 article.
- `String getTitle()`/`void setTitle(String title)` : getter and setter for the title of the article.
- `String getText()`/`void setText(String text)` : getter and setter for the body of the article.
- `List<CommentEntity> getComments()`/`void setComments(List<CommentEntity> comments)` : getter and setter for the comments of an article. The relation between an article and list of corresponding comments is specified using the `@OneToMany` annotation of the JPA.
- `UserEntity getAuthor()`/`void setAuthor(UserEntity author)` : getter and setter for the article's author. The relation between an article and its author is specified using the `@ManyToOne` annotation of the JPA.

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 `UserEntity`. 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(String username)`. 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 `ArticleBean`:

- `ArticleEntity insertNewArticle(String title, String body, UserEntity author)`. It creates a new `ArticleEntity` and uses an `EntityManager` to persist the new article into the database. Finally, it returns the persisted `ArticleEntity`.
- `void deleteArticle(ArticleEntity article)`. It uses an `EntityManager` to remove an article from the database.
- `ArticleEntity getArticleById(Integer articleId)`. It queries the database (using an `EntityManager`) looking for an article with the specified id, and it returns the corresponding `ArticleEntity` if this exists, `null` otherwise.
- `List<ArticleEntity> getAllArticles()`. It queries the database (using an `EntityManager`) for the list of all the persisted articles.
- `List<ArticleEntity> getArticlesByAuthor(UserEntity author)`. It queries the database (using an `EntityManager`) for the list of all the articles posted by a specific user.

- `CommentEntity commentArticle(ArticleEntity article, String title, String body, UserEntity user)`. It creates a new `CommentEntity` and updates the persisted article adding the new comment. It returns the created `CommentEntity`.
- `void removeComment(Integer commentId, ArticleEntity article)`. It updates the persisted article by removing the associated comment identified by the received comment id. It throws an exception if the article is not found.

Question 3, Alternative 1: Project management (3 points)
A simple project has been set up to build an application for the reservation of conference rooms.
The project is divided into four stages: initiation, planning, execution and closing.
The work breakdown structure has identified four software components for the application: request management, reservation activation, reservation deletion and communication management.

The baseline plan is:

- Initiation phase: 2 working days and cost 200€.
- Planning phase: 8 working days and cost 800€.
- Execution phase: planned 5 working days and 1.500€ for each software component.
- The budget for the project is 8000€.

The project status after 20 working days is the following:

- initiation phase completed;
- only the first software component completed;
- actual cost 3150€.

Calculate earned value (EV), planned value (PV), cost performance index (CPI), estimated budget at completion (EAC) in the hypothesis that the cost performance index remains constant for the project lifecycle.

Solution:

BAC = 8000
AC = 3150

EV = 1000€ (initiation+planning) + 1500€ (first component) = 2500€
Note that the text was ambiguous on this point as planning was not mentioned explicitly among the completed phases. So, we accept also solutions where:
EV = 200€ (initiation) + 1500€ (first component) = 1700€

PV = 1000€ (initiation+planning) + 3000€ (two components) = 4000€
CPI = EV / AC = 2500 / 3150 = 0,80
(clearly, you get a different value if you use EV = 1700€)
EAC = BAC / CPI = 8000 / 0,80 = 10000€

Question 3, Alternative 2: Testing (3 points)

Consider the following C instructions:

```
1 int a, b;
2 scanf("%d, %d", &a, &b);
```

Write a program fragment that uses the two variables `a` and `b` in which there is an unfeasible path. Show by symbolic execution that the path is indeed unfeasible, and thus no test case can be found to cover it.

Solution

A simple example of a program with an unfeasible path is the following:

```
1 int a, b;
2 scanf( “%d, %d” , &a, &b);
3 if (a > 0)
4    if (a < -1)
5        b = a;
6    else b = c;
```

If we symbolically execute it trying to follow the path 1, 2, 3, 4, 5, we should have that:
Given `a = A`, `b = B`, the path condition `A > 0` and `A < -1` must be true for the chosen path to be executed.
This is impossible, thus the path is unfeasible.
Hence, the program will always follow the path 1, 2, 3, 4, 6.