Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto, Luca Mottola

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

Software Engineering II

March 2nd, 2017

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.

Question 1 Alloy (7 points)

You are to model in Alloy the relationships among faculty and students of a technical university. Students are characterized by an academic record comprising the courses they successfully passed. Students may be graduate or undergraduate. A course is taught by a faculty, has a set of pre-requisite courses, and includes both students taking the course as compulsory and students taking the course as optional. Graduate students may be teaching assistants in a course. Every course needs at least one teaching assistant.

You need to express the constraints below, plus any additional constraint deemed necessary to ensure the consistency of the model. In the answer, please separate the former from the latter. Also, please provide a **brief** comment in natural language for your specifications.

1) There exist no courses solely comprising students taking the course as optional;
2) Students cannot attend the same course they are a teaching assistant for;
3) Every faculty can teach at most two courses;
4) The students attending a course fulfill the corresponding pre-requisites.

Solution

```
abstract sig Person {}

sig Faculty extends Person {}

abstract sig Student extends Person {
  transcript: set Course
}

sig Graduate, Undergrad extends Student {}

sig Course {
  taughtby: one Faculty,
  assistants: some Graduate,
  compulsory: set Student,
  optional: set Student,
  prerequisites: set Course
}{compulsory & optional = none}

fact {

  -- Needed to ensure consistency: there are no cycles in the prerequisite
dependences
  all c: Course | !(c in c.^prerequisites)

  -- Needed to ensure consistency: no student is attending a course twice
  all c: Course, s: Student | s in (c.compulsory + c.optional)
       implies !(c in s.transcript)

  -- No course exists with only students taking that as optional
  all c: Course | some c.optional => some c.compulsory

  -- Teaching assitants are not enrolled in the same course
  all c: Course | c.assistants & (c.compulsory+c.optional) = none

  -- Every faculty teaches at most two courses
  all f: Faculty | #{c: Course | f in c.taughtby} <= 2
```

```
  -- All students attending a course fulfill the pre-requisites
  all c: Course, s: Student | s in (c.compulsory + c.optional)
       implies (c.prerequisites - s.transcript = none)
}

pred show[]{
  #Course.optional>0
  #Course.prerequisites>0
}

run show for 5
```

Questions 2 JEE (6 points)

The following Bean is used by a food delivery company to coordinate the deliveries through drivers biking around the city. The main functionality it should allow the company to perform are:

i) insert new drivers and bicycles in a data store;
ii) obtain the list of existing drivers and bicycles from the data store;
iii) assign one driver to a bicycle, and store this information onto the data store;
iv) release one bicycles from a driver, and reflect this information in the data store.

You are to check the functional correctness of the code against the functional requirements above and any other potential software defect that may prevent the system from correctly operating. For each issue you possibly identify, you need to **briefly** explain the nature of the issue, how it may prevent **which** requirement to be fulfilled, and provide a **short code snippet** to correct the issue. If you like, you can correct the code by directly modifying what is reported below. If you find no issue, please state that explicitly in your answer. Assume all involved entities are correct and provide standard implementations of getters and setters methods.

```
@Stateless
public class AdministrationBean {
   @PersistenceContext(unitName="restaurant-examplePU")
   private EntityManager em;
   public AdministrationBean(){}
   /*
   * This method queries the database to get the Bicycle entity associated with the input bicyclePlate.
   */
   public Bicycle getBicycleFromPlate(String bicyclePlate){
      try {
         return (Bicycle) this.em.createQuery("SELECT b FROM Bicycle b")
               .setParameter("plate", bicyclePlate).getSingleResult();
      } catch (NoResultException ex) {
         return null;
      }
   }

   /*
   * This method queries the database to get the Driver entity associated with the input driverName.
   */
   public Driver getDriverFromName(String driverName){
      try {
         return (Driver) this.em.createQuery("SELECT d FROM Bicycle d WHERE d.driverName=:driverName")
               .setParameter("driver", driverName).getSingleResult();
      } catch (NoResultException ex) {
         return null;
      }
   }

   /*
   * This method inserts a new Bicycle into the database.
   */
   public void insertBicycle(Bicycle newBicycle){
      em.persist(newBicycle);
   }

   /*
   * This method inserts a new Driver into the database.
   */
   public boolean insertDriver(Driver newDriver){
      if (getDriverFromName(newDriver.getDriverName()) == null){
         em.persist(newDriver);
         return true;
      }
      return false;
   }

   /*
   * This method returns the List of Bicycles stored in the database.
   */
   public List<Bicycle> getBicycles() {
      try {
         return (List<Bicycle>) this.em.createQuery("SELECT b FROM Bicycle b").getSingleResult();
```

```
        } catch (NoResultException ex) {
            return new ArrayList<Bicycle>();
        }
    }

    /*
    * This method returns the List of Driver stored in the database.
    */
    public List<Driver> getDrivers() {
        try {
            return (List<Driver>) this.em.createQuery("SELECT d FROM Driver d").getResultList();
        } catch (NoResultException ex) {
            return new ArrayList<Driver>();
        }
    }

    /*
    * This method releases the Bicycle with the given bicyclePlate by setting lastDriver to null.
    */
    public boolean releaseBicycle(String bicyclePlate) {
        Bicycle toRelease = getBicycleFromPlate(bicyclePlate);
        if (toRelease != null){
            toRelease.setLastDriver(null);
            em.merge(toRelease);
            return true;
        }
        return false;
    }

    /*
    * This method assigns a Driver identified by the input driverName to a Bicycle
    * identified by the input bicyclePlate.
    */
    public boolean assignDriver(String bicyclePlate, Driver driver) {
        Bicycle bicycle = getBicycleFromPlate(bicyclePlate);
        if (bicycle != null){
            bicycle.setLastDriver(driver);
            em.merge(bicycle);
            return true;
        }
        return false;
    }
}
```

Solution

There are six potential defects in the code. These are:

1) The method `getBicycleFromPlate` misses a WHERE clause in the SQL statement. The parameter `plate` does not appear in the statement. All methods relying on this are going to be affected. Therefore, requirement i), iii) and iv) will not be fulfilled.
2) The method `getDriverFromName` is pulling from the wrong DB table. Similar to the above, all methods relying on this are going to be affected. Therefore, requirements i) and iii) will not be fulfilled (the current code does not highlight the non-fulfillment of requirement iii), but the corrected code will do it).
3) In method `getDriverFromName`, the parameter binding is wrong, as `driver` doesn't appear in the SQL statement. As per point 2), requirement i) will not be fulfilled.
4) The method `insertBicycle` should check that the Bicycle is indeed a new one, but it doesn't. Requirement i) will not be fulfilled. Moreover, if the database will contain duplicates because of the erroneous implementation of `insertBicycle`, also the other requirements will not be fulfilled.
5) In method `getBicycles`, `getSingleResult` is used instead of `getResultList`. Therefore, requirement ii) will not be fulfilled.
6) The method `assignDriver` does not check that `driver` is not **`null`**. As a result, one may assign a **`null`** driver to a bicycle, which is semantically equivalent to releasing a bicycle because of the implementation of `releaseBicycle`. Requirement iii) will not be fulfilled.

Question 3 Function Points (7 points)

Consider the following case study and compute the function points for this software focusing exclusively on **Internal Logic Files** and **External Inputs**. Please provide an explanation for each point you consider and for the associated complexity. Refer to the following table to associate weights to the function types:

Function types	**Weights**		
	Simple	Medium	Complex
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiry	3	4	6
Internal Logic Files	7	10	15
External Interface Files	5	7	10

An e-commerce software offers to customers the possibility to:

1. search for products;
2. select products and store them in their cart;
3. delete products from the cart;
4. buy all products in the cart – this operation results in the creation of an order;
5. visualize the status of an order, the status can be placed, being delivered, delivered, returned;
6. return the products in an order, for simplicity, let's assume that the user can return all products or none of them.

Solution

ILFs:

- *Product*: we can imagine that products are of medium complexity because they will include various types of information including pictures, opinions of customers, availability in the warehouse etc.
- *Cart*: this includes the list of products. So, per se is simple.
- *Order*: this includes the identifier, quantity and price of each purchased product. Also, it includes the status and any information suitable for tracing the order while it being delivered. It can be seen as simple.
- *Customer*: for each customer the system will store information about personal data and payment information. It can be seen as simple.

Based on this analysis, we have 10+7*3=31function points corresponding to ILF.

External Inputs:

- *Select products and store them in the cart*: products are being visualized by the customer who can select them and add them into the cart. This is a simple operation.
- *Delete products from the cart*. This is a simple operation as well.
- *Buy all products in the cart*. This operation can be considered complex as it requires a transaction to be executed to ensure that the availability of the product in the warehouse is correctly updated. It also requires the interaction with an external service for managing payment (this will result in a corresponding External Interface File to be considered).
- *Return the products in an order*. This operation is quite complex as well as it requires the generation of a return document for the order and the interaction with the payment service for returning the money back to the customer.

Summing up, we have 3*2+6*2=18 function points corresponding to EIs.

The other functions offered by the system are inquiries and therefore are not considered.