Modelling & System Development (INFOMSO)
February 2, 2010

Part I *(50 points)*

In the multiple choice questions below, there is only one good answer. You get +2.5 pt for each good answer, 0 pt if you don't answer, and −0.83 pt for a wrong answer. A negative total is rounded up to 0.

Question 1

A development team is considering whether to use a relational database system as the persistence layer, or to build a custom persistence layer directly on top of the file system instead. Which of the following is *not* a relevant factor when considering the choice above?

a) Whether or not we want to have well encapsulated classes.

b) Whether or not we will need to roll back transactions.

c) Whether or not we will have multiple users that concurrently use the system.

d) Whether or not we need to do complex queries on the saved data.

Question 2

When deciding to persist/save objects in a relational database we need to keep in mind that we will have to deal with so-called impedance mismatch. What do people mean with it?

a) Objects do not map straightforwardly to tables.

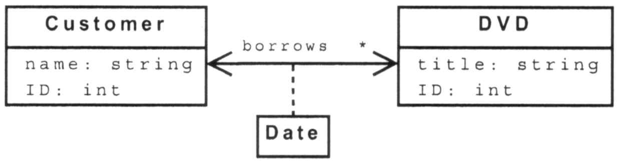
b) Mismatch in processing speed, between objects and tables.

c) Some information is lost, because we save in tables.

d) Objects and the database reside in different machines, which may lead to hardware mismatch.

Question 3

Consider these classes:



We decide to save customers, dvds, and the 'borrow' association all in a single table like the one below in a relational database:

CustomerID	CustomerName	DVDID	DVDTitle	BorrowDate
1	Patrick	1	Sponge Bob I	12/12/09
1	Patrick	1	Sponge Bob I	01/01/10
1	Patrick	2	Kungfu Panda	01/01/10
2	Bob	2	Kungfu Panda	01/01/10

Which of the following statements about the table above is correct?

a) It is not even in 1NF.

b) It is in 1NF, but not in 2NF.

c) It is in 2NF, but not in 3NF.

d) It is in 3NF.

Question 4

Dennis et al name a number of aspects they consider important towards designing a good user interface, e.g. layout, consistency, and user experience. Name another one: (to be clear: just one of these alternatives is correct)

a) Inheritance.

b) Class cohesion.

c) Minirnized user effort.

d) Good documentation.

Question 5

WND is a useful instrument in the design of user interface. What do you use it for?

a) To design the layout.

b) To design the structure of the user interface.

c) To develop the user interface's prototype.

d) To evaluate the user interface's prototype.

Question 6

Which of these statements is correct?

a) A cohesive and well encapsulated class is easier to understand.

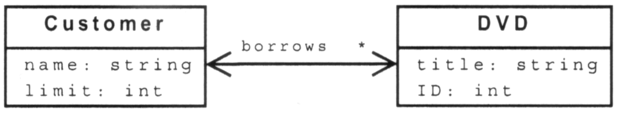
b) A cohesive class is bad, but it can be improved by decreasing encapsulation.

c) We can improve the encapsulation of our class by making more attributes or operations public.

d) We can make a class more cohesive by eliminating inheritance.

Question 7

Consider these two classes:



We want to express the following business constraints:

A. No DVD can be borrowed by two different customers.

B. The number of DVDs a customer borrows never exceeds his borrowing limit.

Which of the following statements is correct regarding how to express those constraints?

a) They can be expressed as multiplicity constraints in the class diagram.

b) They really require OCL to express.

c) A can be expressed as multiplicity constraints in the class diagram; B has to be expressed with CCL.

d) A must be expressed with OCL; whereas B can be expressed as multiplicity constraints in the class diagram.

Question 8

Consider the class below; don't worry about details like numerical rounding, since the question will be about coupling.

```
public class Product {

int price ; // the product's price
    float discount ; // possible discount

    // Base tax, implemented as a static (!) attribute:
    private static float basetax = 5.0 ; // by default set to 5%

    // A constructor:
    public Product(int p) { price = p ; discount = 0.0 ; } .

    // A helper method for calculating percentage
    private float percentage(f1oat percent, int base) {
        return percent * (float) base / 100.0 ;
    }

    // To calculate the price after base tax, other tax, and discount:
    public int getPrice(int tax) {
        // deducting discount first:
        float p = percentage(100.0 | discount,price) ; ˜
        // price after tax: "
        return percentage(100.0 + tax + basetax,p) ;
    }

    // for changing the base-tax:
    public int setBaseTax(float tax) { basetax = tax ; }
}
```

Because in an OO system we will be passing objects around (to methods) almost all the time, we will *not* distinguish between data and stamp coupling.
Now, what kind of coupling we have (A) between the methods `getPrice` and `percentage`, and (B) between `getPrice` and `setBaseTax`?

a) A is *data coupling*, and B is *global coupling.*

b) A is *data coupling*; but there is actually *no* coupling in the case B.

c) We have *global coupling* in both cases.

d) We have *data coupling* in both cases.

Question 9

Which of the following would best characterize what a *design pattern* is?

a) It is a code transformation, to improve its cohesion.

b) It is a pattern for designing UML class diagrams, towards improving their cohesion.

c) It is a generic solution, that is well tried in practice.

d) It is a generic code transformation, that is well tried in practice.

Question 10

We need to extend a class *Film* so that we can also have e.g. ‘new’ film and ‘premium’ film, each with its own additional functionalities. We should be able to turn a ‘new’ film to an ordinary or ‘premium’ film, or the other way around, at the runtime. So, the binding of a film to its ’new’ or ’premium’ features should not be fixed statically. Which design pattern would be a best fit for this situation?

a) Decorator

b) Composite

c) Bridge

d) We make ‘new’ and ‘premium’ subclasses of `Film`.

Question 11

Which problem is solved by the MVC pattern?

a) We have a module that needs to access a third party library. We need a stable interface towards this library, so that the connection with our module does not break when the library is changed.

b) We have a set of data that may be updated at the runtime; we need to maintain various views/presentations of the data, which should be updated automatically when the data changes.

c) We have objects representing various products, e.g. coffee, tea, etc. For each product we have a European and US variants of it. We want to have a convenient interface to create products from a given variant (e.g. to create US-coffee and US-tea).

d) A class has several methods with too many parameters. We need a generic way to transform the code of these methods so that we can either get rid or group some of the parameters.

Question 12

Your test suite delivers 100% decision coverage. What is the most reasonable conclusion you draw from this?

a) We have found and fix all the bugs; so we are done.

b) Decision coverage is already stronger than line and path coverage; so we should definitely stop now.

c) Decision coverage is actually weaker than path coverage; we should test more until we have 100% path coverage.

d) This gives confidence. Under constrained time/budget we have at least a ground to stop testing now; but keeping in mind that we may not have seen all bugs.

Question 13

After a lot of unit, integration, and system testing, what is the most important reason for doing another round of testing, namely *acceptance testing*?

a) The system owner, and his users, need to convince themselves that that the developer delivers them a system that indeed meets their business requirement.

b) We need to test the system as a black box as well, which we do in the acceptance testing.

c) To make sure that no class invariants are violated, in particular those class invariants expressing the system's business rules.

d) To make sure that the system can perform adequately, at least under its normal oper- ation load.

Question 14

Which of the following would be a good reason to do black box testing?

a) At the integration or system-level, the test object (SUT) would be quite complex; considering its entire source code will be too overwhelming.

b) It is an effective way to reduce testing cost.

c) It can help us improving test coverage.

d) Users see the target system as a black box; so black box testing gives us a way to test the system from the users' perspective.

Question 15

Which of the following would best characterize what *refactoring* is?

a) It is a transformation on the source code towards making it more maintainable.

b) It is a transformation on the source code to improve performance.

c) It is a proven approach to eliminate unnecessary parts of software.

d) It is a generic solution which has been well tried in practice.

Question 16

Which one of the following is ***not*** *bad smells* (indicators that refactoring may be needed)?

a) Some methods are too long.

b) A class has many bugs.

c) What should be a single fix leads to a shotgun fix.

d) A class with feature envy.

Question 17

Which of the following is *not* a refactoring step?

a) Move Method.

b) Replace Temp with Query.

c) Hide Delegate.

d) Domain Partition.

Question 18

When considering a method M in a class C, we decide to do *Extract Method.* Give a good reason why we want to do that:

a) M is doing too much work that should be done by another class.

b) The original block of statements that is extracted (from M) is complex and textually it (the block) does not help much in understanding the global algorithm of M.

c) M contains a switch.

d) Only some instances of C use M; so we should create a subclass to represent these instances, and move M to this subclass.

Question 19

Which of one of these activities do *not* belong to the change management, in the context of system migration in a company?

a) Revising management policy to suit the new system.

b) Anticipating and understanding resistance (by employees/users) towards the new sys- tem.

c) Training employees to use the new system.

d) Keeping track of various changes to the source code.

Question 20

What do Dennis et al mean by a *parallel* conversion strategy/style?

a) All subsystems are simultaneously converted from the old to the new ones.

b) The old system is not immediately disengaged, but is run in parallel with the new system for some period of time to detect possible inconsistencies.

c) If the company owning the system has multiple locations, then all locations are simultaneously converted from the old to the system.

d) If the company adopts an agile development approach, the conversion is carried out in parallel with development.

Part II *(50 points)*

Question 1: Persistence *(20 points)*

a) [10pt] Consider this table:

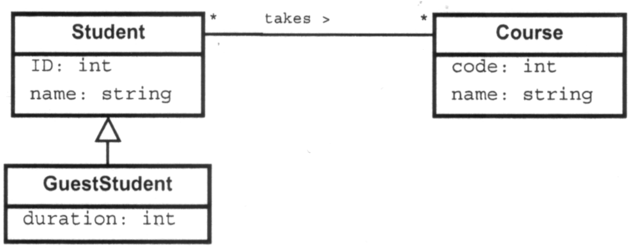
StdID	StdName	CourseID	CourseName
0	Patrick	0	Introduction UML
0	Patrick	1	Introduction UML
1	Bob	1	Introduction UML
7	Patrick	1	Introduction UML
8	Octo	9	Project Planning

Give all partial attribute-dependencies of this table; then turn the table to 2NF.

b) [3pt] What is a *transitive dependency*?

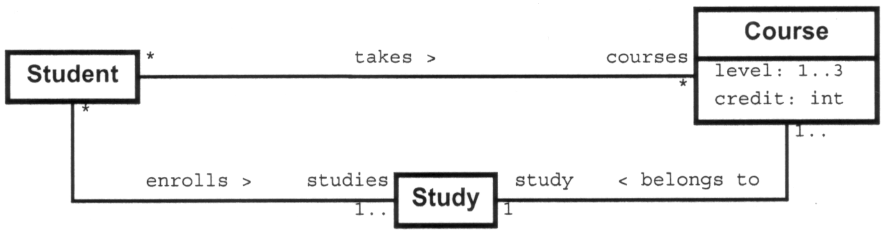
c) [3pt] Can you further normalize your 2NF table to 3NF? If so, give the 3NF; else explain why not.

d) [4pt] Consider this model:



Propose how to save objects from the above model in a relational database; you can do this by giving samples of the needed tables to demonstrate your proposal. Your tables should contain sufficient information to reconstruct the original objects when we need to pull them out form the database.

Question 2: Specification *(10 points)*



Consider the model above. So, a 'student' can enroll to one or more studies. A 'study' is made of a set of 'courses'; a course belongs to exactly one study. Express the following constraints as class invariants of, respectively, Course and Student. Use OCL.

a) *A level 3 course should have a credit of at least 10.* You may use boolean operators with the usual meaning.

b) *A student can only take courses that belong to the studies he/she enrolls to.*

Question 3: Design Pattern *(10 points)*

Consider the following simplified concept of electronic archive. An archive is either a single *text file*, or it is a collection of other archives – the nesting can be arbitrarily deep. An archive is never part of itself, and an archive cannot be a part of two different archives. There is a client class that needs to use archives, and it requires the functionality `size` to calculate the total size of all text files contained (directly or indirectly) in an archive.

a) Give an UML class diagram showing how you model the recursive structure of archive. Which design pattern is useful here?

b) Show how you implement `size`, according to your design pattern. You can assume that a *text file* knows its own size.

Question 4: Testing *(10 points)*

Consider these classes Person and Insurance:

```
public class Person {
string name ;
int    age ;
int    children ; // number of children
...    // constructors and other fimctionalities
}
public class Insurance {
    int basePremium ;

public int premium(Person p){
        int p = basePremium ;
        if (p.age >= 55) p += 0.1 * (float) p ; _
        else
          if (p.children>0) P = 0.9 * (float) p ;
        return p ;
    }
}
```

The method premium calculates the premium that a given person must pay for the insurance. Its specification is:

> *If the person `p` is 55 years or older, he pays 10% more over the insurance's base premium. If he/she has children, she gets 10% discount over the base premium, but only if he is not older than 55 years.*

Give a test suite (set of test cases) to test the method premium according to the specification above, and furthermore would give you a full decision coverage over the method *premium.*
You may assume you have sufficient functionalities in both classes. You don't have to give Java code; pseudo code will do.