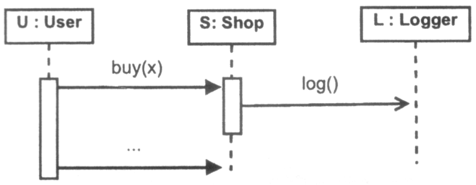
Modelling & System Development (INFOMSO)
01-02-2011

PART I

For eacht multiple choice question below, there is only one good answer. You get +3pt for each good answer, 0 pt if you don't answer, and -0.8pt for a wrong answer. A negative total is rounded up to 0.

Question 1.

In the sequence diagram on the right, *U* sends a `buy(x)` message to *S*, which subsequently sends a `log()` message to *L*.
Sending a message like above is actually a request to the receiver to execute the corresponding operation.
What else does the diagram tell us?



a) *U* will wait for *S* to finish executing `buy(x)`; however *S* will *not* wait for *L* to finish executing `log()`.

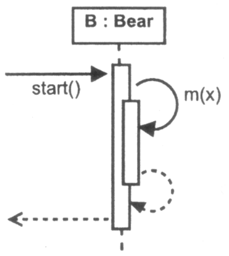
b) *U* will *not* wait for *S* to finish executing `buy(x)`; however *S* will wait for *L* to finish executing `log()`.

c) *U* will implicitly wait for *L* to finish executing `log()`.

d) Both *U* and *S* wil *not* wait to finish executing `log()`.

Question 2.

Consider the sequence diagram on the right.
On receiving the message `start()` there is that arrow labelled with `m(x)` on *B*'s lifeline.
What does this arrow mean?



a) *B* waits for the condition `m(x)` to be satisfied.

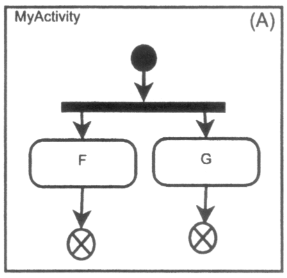
b) *B* *repeatedly* calls its own operation `m(x)`.

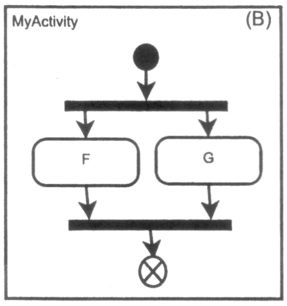
c) Simply that *B* will call its own operation `m(x)`.

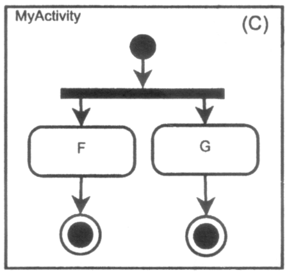
d) None of the above. It is not a valid UML notation.

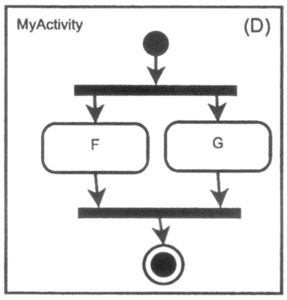
Question 3.

We want to model an activity (call it 'MyActivity') that does F and G subactivities. The *order does not matter*, but *both* must be completed. Which of the following does model NOT correctly captures this?









Question 4.

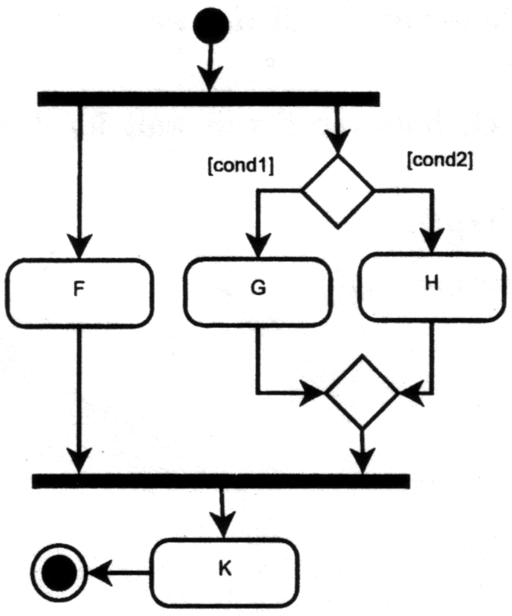
Consider the diagram on the right.
Which activities have to be completed before we can start with the activity K?

a) all of F,G,H.

b) either (F and G) or (F and H).

c) just one of F,G,H.

d) either just J, or both (G and H).



Question 5.

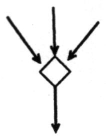
Which of the following statements about *activity diagram* in UML is NOT correct?

a) You can use partitions/swimlanes to group activities in an activity diagram according to the parties/classes that carry them out.

b) You can use an object flow to model objects that you want to pass from one activity to another in an activity diagram.

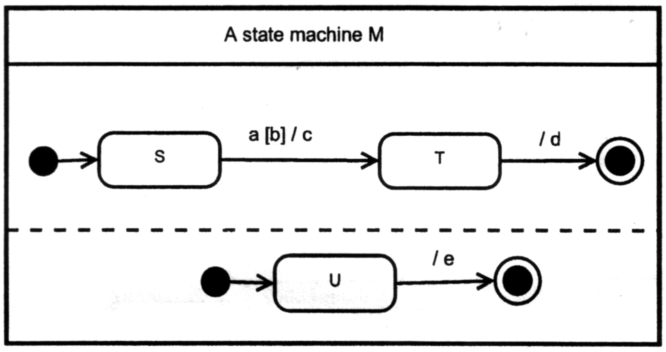
c) You can use partitions/swimlanes to synchronize multiple activities (so that activities in the same swimlane cannot overtake each other)

d) The notation below is called a *merge node*. The output is activated as soon as there is one token in one of its input.



Question 6.

Consider the following (behavioral) state machine M: Which of the following is NOT correct? (please



notice the different symbols in the model)

a) Event a will move the machine from state S to T, provided the condition b is satisfied. The transition will also execute the action c.

b) It is possible that the machine can be in both states T and U at the same time.

c) When the machine arrives in state T it will make an automatic transition to the terminal state of the upper submachine.

d) When the machine arrives in the state U, only on event e it will proceed to the terminal state of the lower submachine.

Question 7.

Consider a class `Magic` which is a part of some software system. In which situation you want to use the Singleton Pattern on the class `Magic`?

a) You want to make sure that there is at most one client program that can access any object of the class `Magic`.

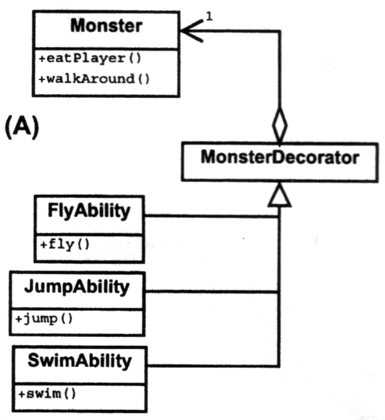
b) You want to hide the constructors of `Magic`.

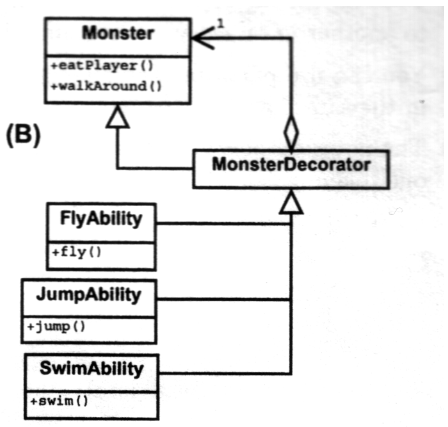
c) You want to enforce that any object of the class `Magic` can only be accessed/used once.

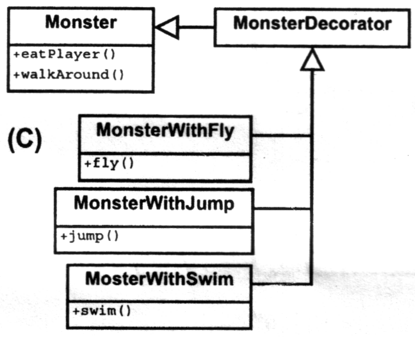
d) You want to make sure that in the whole system there is only one object/instance of the class `Magic`.

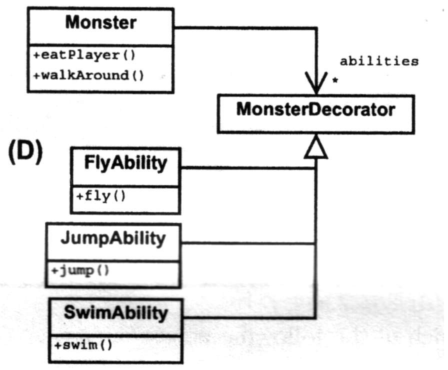
Question 8.

Consider the class `Monster` representing monsters in a computer game. In the game a monster can dynamically gain (or lose) abilities, such as abilities to ljump, to swim, to fly, etc. We want to build these abilities as decorators yusing the Decorator Pattern. How does the model of this look like?









Question 9.

Suppose you are developing a computer game with monsters. The game runs on a server and has multiple users/games connecting to it. Consider these situations:

A. To calculate the monsters' actions, we want to reuse a game logic provided by a special package delivered by another company. It has a class `ThePlan` to represent a plan of actions. However, you already have your own class `Plan` - you do not want to change it because there are other parts of your system that already depend on it.

B. Some monsters actually run on players' machine. The game logic at the server needs to control these monsters.

Which design patterns are applicable to help us solve the above situations?

a) Adapter Pattern for A, and Proxy Pattern for B.

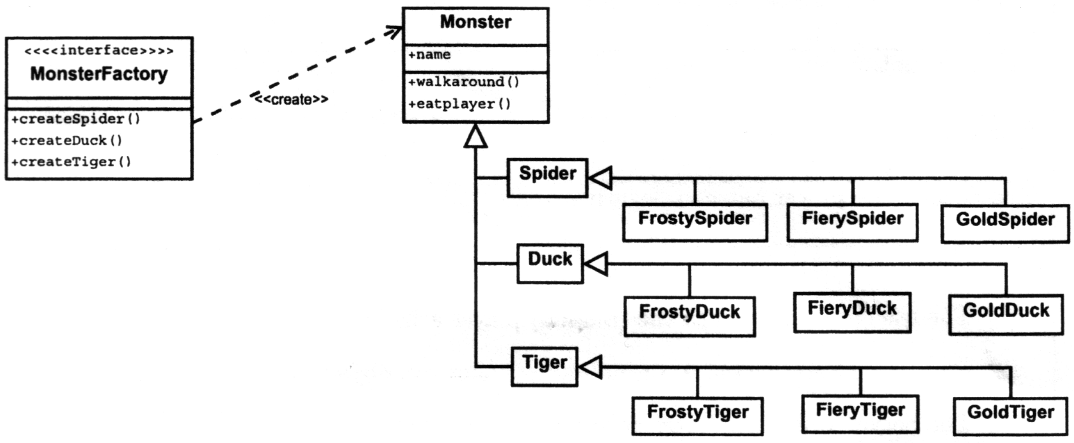
b) Facade Pattern for both A and B.

c) Decorator Pattern for A, and Proxy Pattern for B.

d) Proxy Pattern for A, and Adapter Pattern for B.

Question 10.

Consider again the computer game we talked about above. There are many subclasses of `Monster`, e.g. Spider, Duck, Tiger, etc. All these monsters come in a number of 'lines'. E.g. we have forsty-line, fiery-line, gold-line, etc. So, we will e.g. FrostySpider, FrostyDuck, FierySpider, FieryDuck, etc. The game generates onlu fiery monsters during the summer, and only frosty monsters during the winter. Gold monsters are generated during special events.
We will use the Abstract Factory Pattern to facilitate the creation of monsters of a particular line. Here is a partial model showing how we apply the pattern: The model is not complete yet. What do



we need to add, according to the Pattern, to allow us to create *frosty* monsters?

a) We need to add a `FrostyMonsterFactory` class, and make it a subclass of `FrostySpider`, `FrostyDuck`, and `FrostyTiger`.

b) We need to add associations connecting `MonsterFactory` to `FrostySpider`, `FrostyDuck`, and `FrostyTiger`.

c) We need to add operations `createFrostySpider()`, `createFrostyDuck()`, and `createFrostyTiger()` to the class `MonsterFactory`.

d) We need to add a subclass `FrostyMonsterFactory` that extends `MonsterFactory`, where we override all those `create`-operations in `MonsterFactory`.

PART II (70pt)

Question 1. Sequence Diagram (15pt).

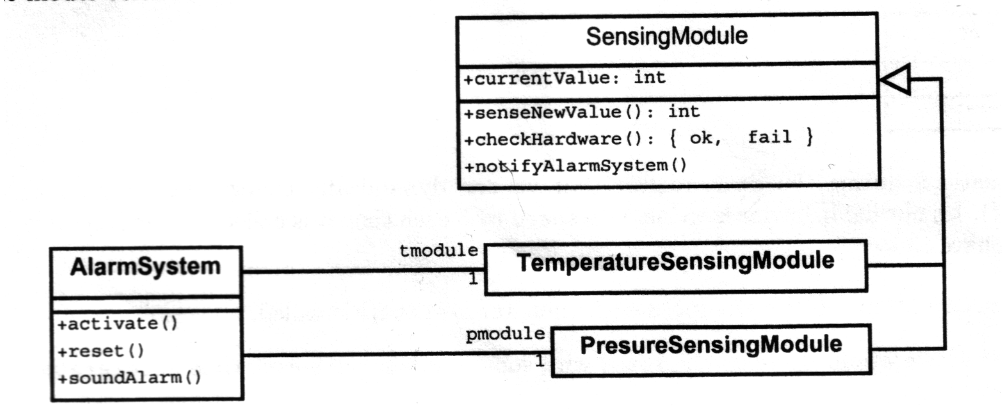
Consider the use-case below. The system is a Game Server.

Use case: play game
Brief desc.: this is used to start and play a game.
Primary actor: (human) player
Secondary actor: none
Pre-cond.: the player has non-negative credit.
Main flow: 1. The use case starts when the (human) player asks for it. 2. The Game Server creates a random opponent (an object of class Opponent). 3. The Server will concurrently do these (3.a and 3.b): (3.a) (3.a.1) It asks the opponent to calculate and send a move. (3.a.2) It (Server) then executes this move. (3.b) It asks the player to send a move. (3.b.2) The Server then executes this move. 4. We repeat (3) until either the opponent's or the player's hit point gets to 0. The game is then over. 5. The server sends the game result back to the player.

Work out the above use-case to a sequence diagram.

Question 2. State machine (15pt).

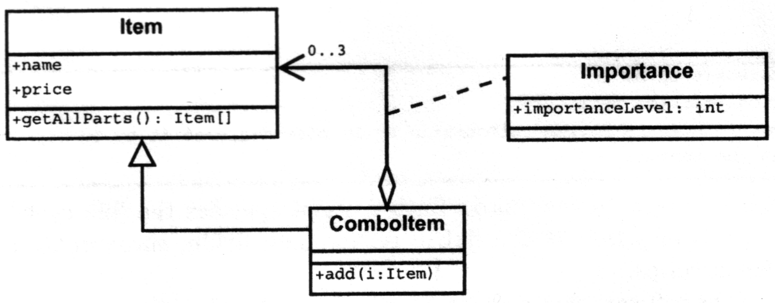
An alarm system consists of a temperature sensing module and a pressure sensing module. See the model below: When activated, the system first asks the two modules to start up themselves. They



will do so *concurrently.* To start up, each will run a check on tis own hardware. This may take a while. If successfull then it goes over the 'ready' state. Only when both modules are in the 'ready' state, then the whole system is in de 'ready' state.
If one of the modules fails its own hardware check, the system should go to a separate 'failure' state, so that the user knows that it fails to start up properly.
When in the 'ready' state, each module will sense temperature respectively pressure at every t seconds. When the measurement increases with more than 5 units twice in a row, it will notify the system, which will subsequently raise an alarm.
There is a reset button that will turn off the alarm, and reset the system back to the ready state.
Model the above behaviour with a state machine.

Question 3. Implementing models (15pt).

Consider this model:



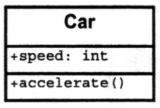
Your tasks:

1. Provide a Java skeleton that implements the model above. You can leave the operations unimplemented, but all other aspects of the models should be implemented.
(Notice the use of association class in the model.)

2. The operation `add(i)` should add an item i as an element of a ComboItem (in the aggregation that you see in the model). Provide an implementation of this operation. You can use a pseudo code, but it should clearly show what you are doing.
You can assume that the operation `getAllParts` is already implemented (so you don't have to do it yourself). When you call it e.g. `i.getAllParts()`, it will return an array of all items which are direct or indirect elements of `i`.

Question 4. Strategy Pattern (10pt).

On the right is a model of cars in a racing game. Some situations during the game require that we can dynamically change the behaviour of `accelerate()`. Its normal behaviour is to increase `speeed` by 5 each time it is called, but additionally we may also have these:



1. Turbo acceleration: double the speed each time `accelerate()` is called.

2. Inverted acceleration: decrease the speed with some random value each time `accelerate()` is called.

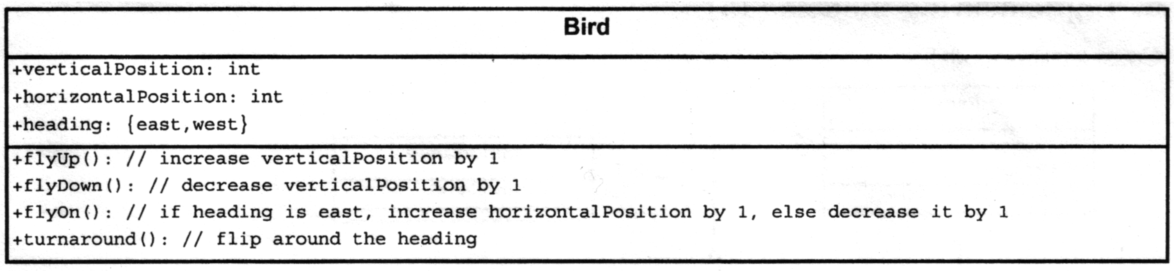
We want to provide a solution for this, and moreover we want the solution te be extensible (in case we want to add more kinds of acceleration in the future). Provide such a solution by using the *Strategy Pattern*:

1. Give a model showing the classes and realtions you need for the pattern.

2. Show how, in your solution above, to implement the above three kinds of acceleration behaviour. Pseudo code is sufficient; but it should be clear what you are doing there.

Question 5. Command Pattern (15pt).

Here is a model of birds in a bird simulation software; what each operation does is explained in the comment:



```
Bird
+verticalPosition: int
+horizontalPosition: int
+heading: {east,west}
+flyUp(): // increase verticalPosition by 1
+flyDown(): // decrease verticalPosition by 1
+flyOn(): // if heading is east, increase horizontalPosition by 1, else decrease it by 1
+turnaround(): // flip around the heading
```

We want to be attach a 'script' to each bird. Such a script specifies the 'life cycle' of this bird in the simulation, e.g. it may tell: flyUp; flyUp; turnaround; flyOn; turnaround, etc. Different birds may have different scripts.
There is a Generator part of the software that generates birds along with their life cycle scripts. There is the Simulator itself that will execute the script of each bird.
To facilitate such scripting we will apply the Command Pattern, where you will need to turn each operation of Bird above to a Command. Provide a model showing how this pattern is applied here, and show how the `execute()` and `undo()` of each Command is defined.