Faculty of Arts and Science
University of Toronto

Midterm Test

Department: Computer Science
Instructor: Steve Easterbrook
Date and Time: 1:10pm, Monday 27th Oct, 2008

Conditions: Closed Book
Duration: 50 minutes

This test counts for 20% of your final grade

Name: _____
(Please underline last name)

Student Number: _____

Question Marks

1 _____/20

2 _____/20

3 _____/20

Total _____/60 = _____%

1. [Short Questions; 20 marks total]

(a) [Modeling – 5 marks] What are the advantages and disadvantages of using a modeling language, such as UML, during software development?

[Many possible answers; give credit for each reasonable advantage and disadvantage given; must have both advantages and disadvantages for full marks]

E.g: Advantages:

- Gives you a map of the program code, allowing you to get an overview and find your way around, especially if you are less familiar with the code.
- Allows you to build abstractions, ignoring detail, to concentrate on particular design decisions
- Allows you to document and/or communicate interesting aspects of the design

Disadvantages:

- Keeping the models consistent with the code can take a great deal of effort;
- Might end up having to do everything twice – in the model and then in the code;
- Modeling languages tend to have ambiguous semantics – might not be understood in the same way by everyone.

(b) [Risk Management – 5 marks] You are developing flight control software for a spacecraft. Your project team has come up with a long list of things that could go wrong. How would you decide which of these risks are important?

To decide importance, need to have a way of measuring risk. There are two common ways of doing this:

One is to quantify the probability (p) of occurrence and the cost (c) of the loss for each risk, and calculate the risk exposure as RE = p x c. Can then compute this for each risk, and use this to put them in order.

The other approach is to use a qualitative matrix, where probability and loss are evaluated on a small number of levels (e.g 3-5 levels for each). Cells in the matrix that have high probability and/or high loss are treated as critical risks to be managed:

		Likelihood of Occurrence		
		Very likely	Possible	Unlikely
Undesirable outcome	(5) Loss of Life	Catastrophic	Catastrophic	Severe
	(4) Loss of Spacecraft	Catastrophic	Severe	Severe
	(3) Loss of Mission	Severe	Severe	High
	(2) Degraded Mission	High	Moderate	Low
	(1) Inconvenience	Moderate	Low	Low

(c) [Agile Planning – 5 marks] Does agile planning make it more likely or less likely to deliver software on schedule? Explain the reasoning behind your answer.

[Note: could argue it either way! Award points for good argument, and inclusion of the main features of agile planning]

(a) agile planning makes it more likely to deliver on schedule because it concentrates on small increments and dynamic adjustment of the plan, hence more able to adjust the plan to meet customer's needs and expectations. Estimates for small coding tasks are much more likely to be accurate, and much easier to keep track of. Hence managers have more insight into what is actually happening in the project, and are better able to adjust the plan to fit reality.

(b) agile planning makes it less likely to deliver on schedule because it doesn't look at the big picture, but just focuses on immediate tasks for the upcoming release. This allows the project to ensure ***something*** is ready by a fixed release date, but it probably won't be what the users/customers were expecting. Hence customers/users can never be sure when they'll get all the features they are requesting.

(d) [Software Architecture – 5 marks] Layered architectures are designed to reduce coupling between components of a software system. Why is this reduced coupling useful? Describe a typical layered architecture, and explain the role of each of the layers.

Reduced coupling is good because it separates the functions that might need to be changed at different times. This is good for:

- Modifiability – changes can be made at one layer without affecting others
- Reusability – layers can be reused in similar systems
- Understandability – easier to understand how the software works

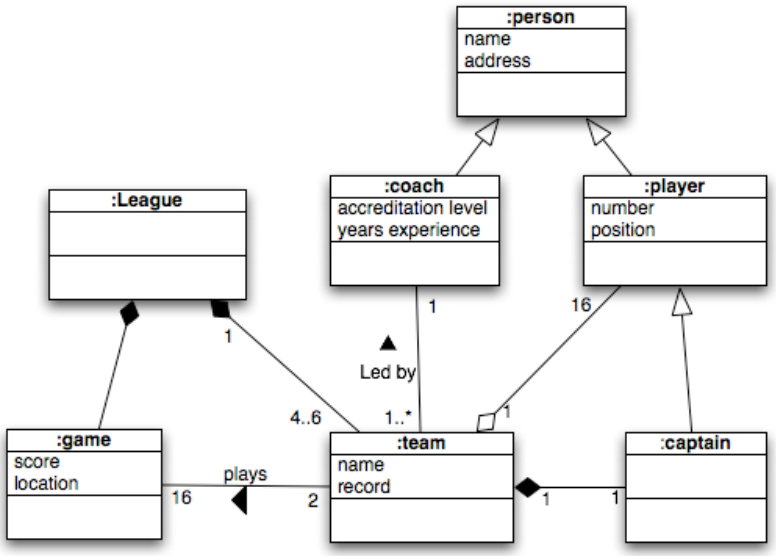
[Must have at least two advantages for full marks]

A typical 3-layered architecture has:

- Presentation layer, responsible for the user interface – e.g. to accept input from the user, display results, and manage the appearance of the interface;
- Business Logic layer for the basic functions provided by the system. Includes all the entity classes, and control classes needed to implement the use cases;
- Data storage layer, responsible for persistent storage of the information users in the system. Usually includes a database component.

[Notes: other possible answers: 2-layers (essentially client-server); 4-layers model splits business logic layer into application layer (responsible for controlling the use cases) and domain entity layer (for basic functions shared by different applications).]

2. **[Domain Models – 20 marks]** The following domain model captures some basic information about a kids' hockey league. In answering the following questions, state any assumptions that you make.



a) How many games will a captain play in? [2 marks]

According to the model, a team plays exactly 16 games. Each captain can only play in one team, and each team has exactly one captain, so a captain must play 16 games

b) What is the maximum number of games that any given coach can be involved in? [3 marks]

In principle, one coach could lead up to 6 teams (the max number of the teams in the league), so could be involved in all (16 x 6)/2 = 48 games.

[Note: nothing in the model stops a coach working in more than one league, so an equally valid answer is that the number is unlimited]

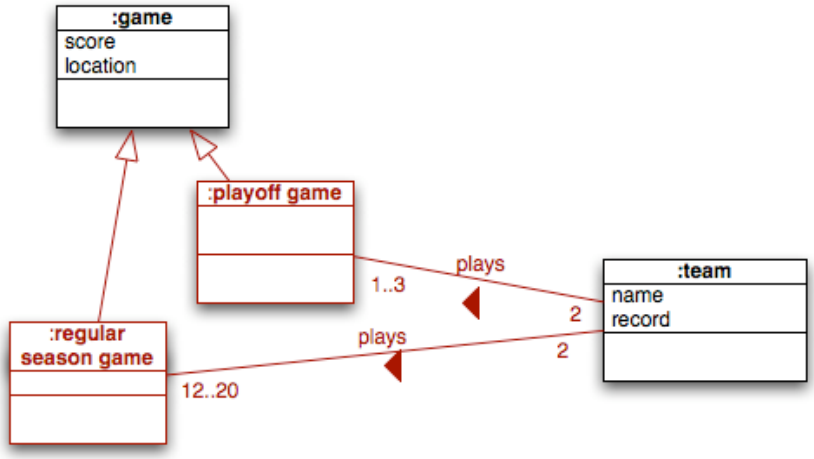
c) In the model, the relationship between player and team is shown as an aggregation, while that between captain and team is shown as a composition. Is this a good choice? Why? [5 marks]

A composition is a stronger form of aggregation, implying that the contained class has no independent existence from the container class. In this case, it makes sense for the captain relation, as a player is only a captain while the team exists. If a team is disbanded, the captain is no longer a captain. If a new team is created, a captain needs to be appointed. In an implementation, it would be reasonable to include code in the constructor and destructor methods for the "team" class to assign/de-assign a captain. For player, the relationship is weaker – players are not created/destroyed when a team is created/disbanded.

[On the other hand, you could argue that the aggregation relation between player and team is meaningless, and a regular association would have done just as well]

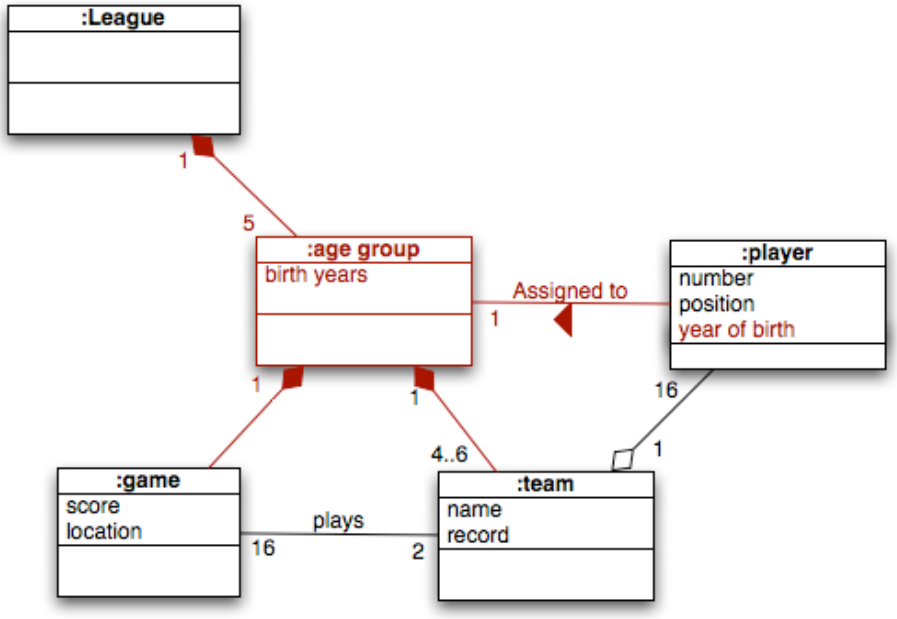
d) The league actually has two types of game: regular season games and playoffs. Each team plays 4 regular season games against each of the other teams. Every team also gets to play in the playoffs, which are conducted as a knockout – a team is out of the playoffs when it loses a playoff game. How would you modify the model to capture this additional information? [5 marks]

The simplest way to do this is to subclass the game class, and move the association from team to each of the two game subclasses:



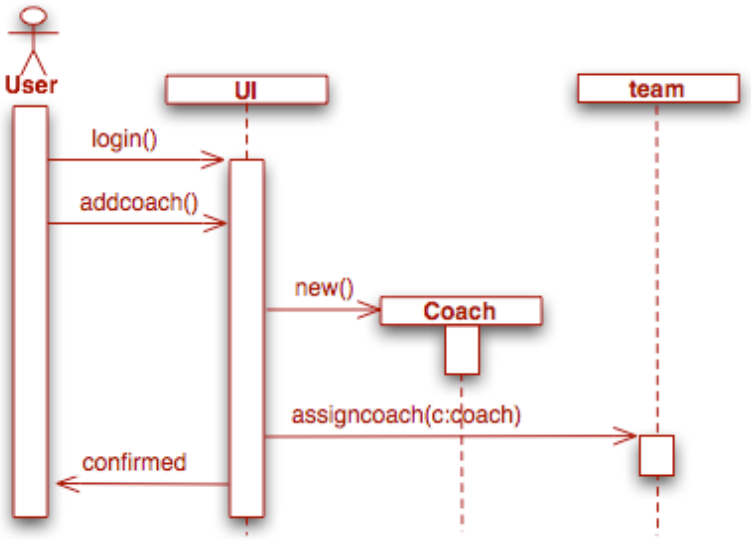
Note: multiplicities for the new associations depend on these assumptions: if there are 4..6 teams in the league, and each team plays 4 regular season games against each of 3..5 other teams, that yields 12..20 regular season games. A knockout competition for the playoff games can go to a max of three rounds with if there are at most 6 teams

e) The league is being expanded to include five different age groups. Players are placed into teams based on their year of birth, and teams only play other teams from the same age group. How would you modify the model to capture this additional information? [5 marks]



3. **[Sequence Diagrams – 20 marks]**. The hockey league described in the previous question is planning to write a program to keep track of information during the hockey season, with the domain model (from the previous question) as an initial design for a set of Java classes. So far, two use cases have been identified: (a) registering a new coach who wants to coach a particular team, and (b) recording the result of a game (which includes updating the teams' records). Use sequence diagrams to sketch an initial design for each of these two use cases, showing which class will be responsible for the various functions, and how these classes will communicate via method calls. State any assumptions you make, including any changes you think are necessary to the class model.

(a) assume there is a UI class that handles requests from the user. Assume that the user has to authenticate at the beginning of the use case. Note: the parameter used in the assigncoach method call is the coach object just created.



(b) assume the game object has already been created. Assume the game object is responsible for updating the two team's records when its score is updated. Assume a score (presumably two integers) is handled as a single object that somehow indicates which team is which.