Victoria University of Wellington
School of Mathematics, Statistics and Computer Science

COMP205: Software Design and Engineering

Notes on 2007 Terms Test

1 Question 1

Generally speaking this question was done reasonably well. The main area people stumbled upon was in identifying the candidate use cases and providing an example Essential Use Case card.

a) The basic nouns are: auctioning system, goods, customers, auction, seller, buyer, items, furniture, CDs, books, description, photograph, reserve price, end-date, bid, postal address and email address.

b) The candidate objects are: items (same as goods), customers, buyers, sellers, auction and bid.

   I did not consider customers to represent the same thing as "buyers" and "sellers". This is because, although a buyer is a customer, *a customer is not always a buyer*. Likewise, it doesn't make sense to say that "items" represent the same thing as "furniture".
   Many people did not include "bid" as a candidate object. This word is somewhat confusing (since it can be a verb as well), but you can say "the bid", "the highest bid" etc, so it's clearly a noun (and an important one).

c) The candidate use cases are: bidding on auction, auctioning item, closing auction, verifying address, initiating contact.

   Many use cases given weren't strictly from the description. For example, "creating customer". This is perhaps implied, but the procedure outlined in lectures is to identify verbs from the description and then shortlist them.
   Many people also included duplicate use cases. For example, "create auction" and "set reserve price". It seems fairly reasonable to assume that when you create an auction, you set the reserve price then. Therefore, these should be treated as one use case and merged.

d) In general, most people seemed a little rusty on their essential use case cards. An example would be:

Create Auction	
Identify User	Verify User
Set item details	Record Item details
Set reserve price	check reserve price > 0
Set end-date	check end-date in future

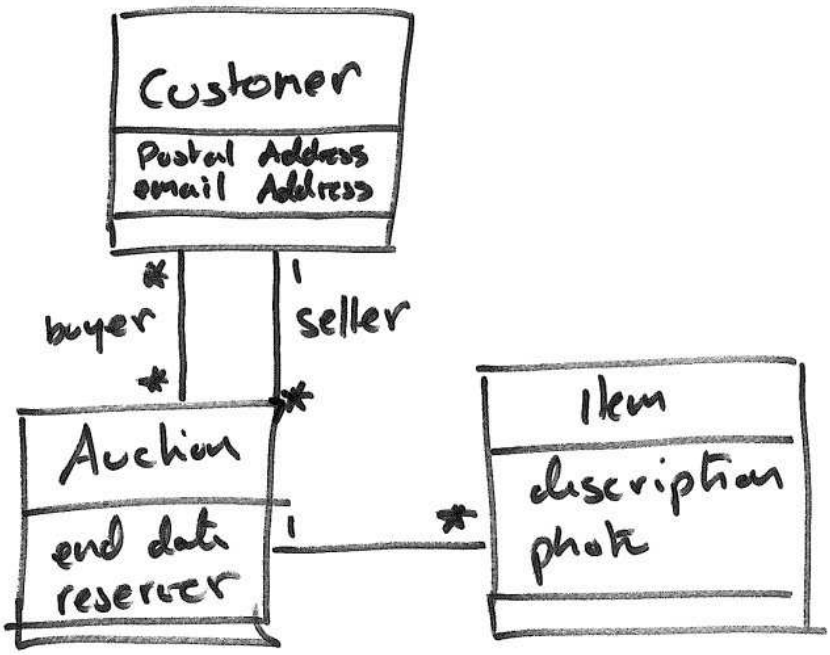
The user must always identify him/herself, whilst the system must always verify this. In some cases, the same applies to the auction (or perhaps item) involved. Also, it is part of the systems responsibility is to check things being entered are valid. It helps to consider that use cases are really the boundary between user and the machine; so, they're the first line of defence against "idiot" users!

e) Some example issues are:

- What happens if auction closes and no bid over reserve price?
- Who sets the reserve price and end-date — system or user?
- How is postal address verified?
- Can sellers be buyers on their own auction?

This question was quite open ended, and lots of people came up with some interesting answers! I was a little strict on some, however, especially when there was no justification from the description. For example, "Are there different types of item?" — in this case, the description says quite clearly that everything has a description and photograph; there is no reason to think that different types are required. Another example: "Can any customer sell items, or only special customers?". In this case, there's no mention of "special customers" in the description and this seems too far removed from the description to be really worth considering.

d) A good UML diagram is something like:



Some people included "Buyer" and "Seller" as subclasses of "Customer", with their links to the auction going from these. One problem with this is that it suggests buyers and sellers are mutually exclusive (i.e. a buyer cannot be a seller), which is unlikely to be the case in practice. Including multiplicity in this UML diagram is important, especially to make clear that "an item can only be sold in one auction at a time".

2 Question 2

a) Example differences include:

- ShelfItem in UML is named `ShelfObject` in class.
- There is no `Author` class in code.
- There is no `Magazine` class in UML.
- There is no field `Number` in code for class `Journal`.
- Class `Journal` extends `Book` in code, but not in UML.
- Association between `ShelfItem` and `Shelf` is 1:1 in UML, but 1:* in code.
- `Shelf` extends `ShelfItem` in code, but not in UML.

Generally, this part was done very well. The only real pitfalls arose from not answering the question properly. For example, something like: "The program will not compile because the `_author` field is assigned but doesn't exist" is answering a different question. That is, although it is correct, it is identifying a problem in the code, not a *difference* between the UML and the code.

b) A good answer to this question was something like:

> "If one author can write many books, or there was more information about an author (e.g. address) it would be sensible to have a separate class. This is because, should the author's details change, you don't have to go through every book object and change it — you can just change the single author object. However, as it stands, the UML implies each author can write only one book. In this case, it makes more sense to have author as a field of book."

Generally, I wanted to see some discussion of what the advantages of using a separate class were; namely, that we don't suffer the copy update problem where changing the authors details requires changing many books written by the same author. Other useful comments included the fact that a book could be written by several authors and that an author might change his name (albeit unlikely).

3 Question 3

Good answers to this question include:

- Interface `MyCollection<T>` includes code for `toString()` method. But, interfaces can only include method stubs!
- Class `MyVector<T>` does not implement the `add(T)` method, as required by the interface. This is because the method `add(T,int)` does not override `add(T)`, rather it overloads it.
- In method `get(Integer)`, an object is being returned where a type T is required. This is a problem since `Object` is not a subtype of `T`.
- Class `MyVector<T>` does not implement the `get(int)` method, as required by the interface. This is because the method `get(Integer)` does not override `get(int)`, rather it overloads it.
- In method `length()`, the statement `new int(last)` does not make sense, since `int` is a primitive type, not an object type.

- In `Test.main`, the line `print(myVec)` will not compile since `Vector<String>` is not a subtype of `Vector<Object>`.

On the whole, this question was done badly. There were two common problems:

1. Not answering the question precisely. For example, something like: "The `add(T, int)` method has two parameters, but the interface specifies just one". The thing is, this answer doesn't identify the real cause of the problem. There is nothing really wrong with `add(T, int)`, since it could exist in a valid implementation of `MyVector<T>`. The problem is that `add(T, int)` does not override `add(T)` and, hence, `MyVector<T>` does not implement a method required by the interface.

2. Not answering the right question. For example, something like: "The `add(T, int)` method goes into an infinite loop because `last` is not decremented". This is quite right, but it is a *runtime problem*, not a *compilation problem* (which is what the question is after).

Finally, several people commented that `++i` was a syntax error; however, this is not the case in Java! In fact, it makes no difference in a `for` loop whether you say `++i` or `i++` — the variable `i` is incremented at the same time in both cases. Generally, I suppose that it's considered bad style to write `++i`. However, I'm an old C/C++ programmer and the habit is hard to beat, but there are lots like me out there!!