ENGINEERING TRIPOS PART IIA

Friday 25 April 2008 9.00 to 10.30

Module 3F6

SOFTWARE ENGINEERING AND DESIGN

*Answer not more than **three** questions.*

All questions carry the same number of marks.

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

There are no attachments.

STATIONERY REQUIREMENTS
Single-sided script paper

SPECIAL REQUIREMENTS
Engineering Data Book
CUED approved calculator allowed

You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

1 The warehouse of a large chemical company receives daily orders for chemicals as a sequence of individual requests. All chemicals are shipped in 1 litre bottles and each incoming order consists of a quantity in litres and a product code. At the end of each day, the warehouse processes each order list. Every bottle ordered is given a unique id and groups of 4 bottles with the same code are shrink-wrapped to form a "4-pack". All 4-packs and all individual bottles are then packed in boxes capable of holding three 4-packs, 12 bottles or a mixture of both. Fig. 1 shows an example order and an illustrative corresponding packed order list ready for dispatch. Fig. 2 shows a UML class diagram for the core of the software system which inputs individual orders and outputs packed order lists. The initial orders are input via `Order::AddBottles()`, given a unique id and stored in `Order::items` as a linear list of individual `Bottle` objects. When the `Pack` method is called, the linear list is transformed into a hierarchical structure matching the packed order.

(a) Explain briefly the difference between a UML class diagram and a UML object diagram. [15%]

(b) Draw a UML object diagram showing the objects instantiated after the order shown in Fig. 1 has been packed. Your diagram should show the full structure but you can use ellipsis (i.e. ...) to indicate repeated objects of type `Bottle`. [25%]

(c) Draw a UML sequence diagram to show the call sequence following a call to `Order::print` to print the packed order list. Use loop sequence fragments where appropriate. [30%]

(d) Outline a design for the `Pack()` methods. Identify in your design any additional methods required for the classes shown in Fig. 2. [30%]

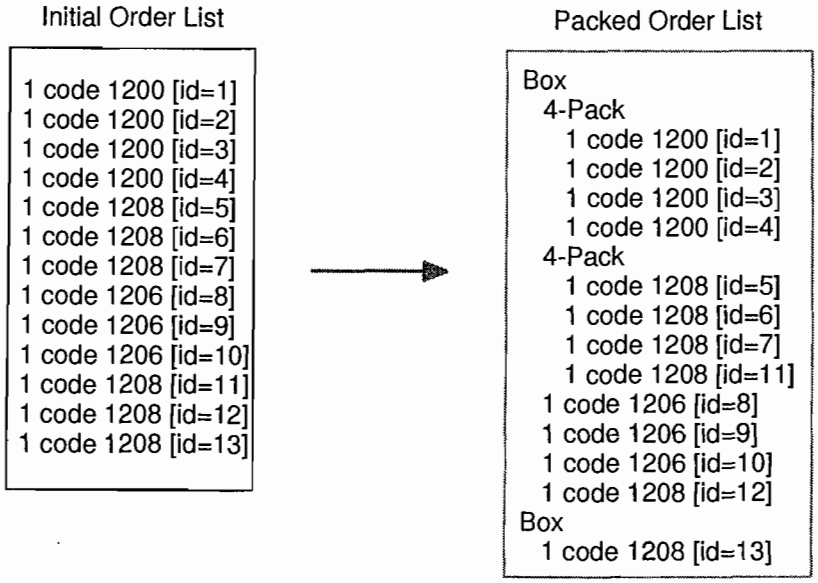

Fig. 1

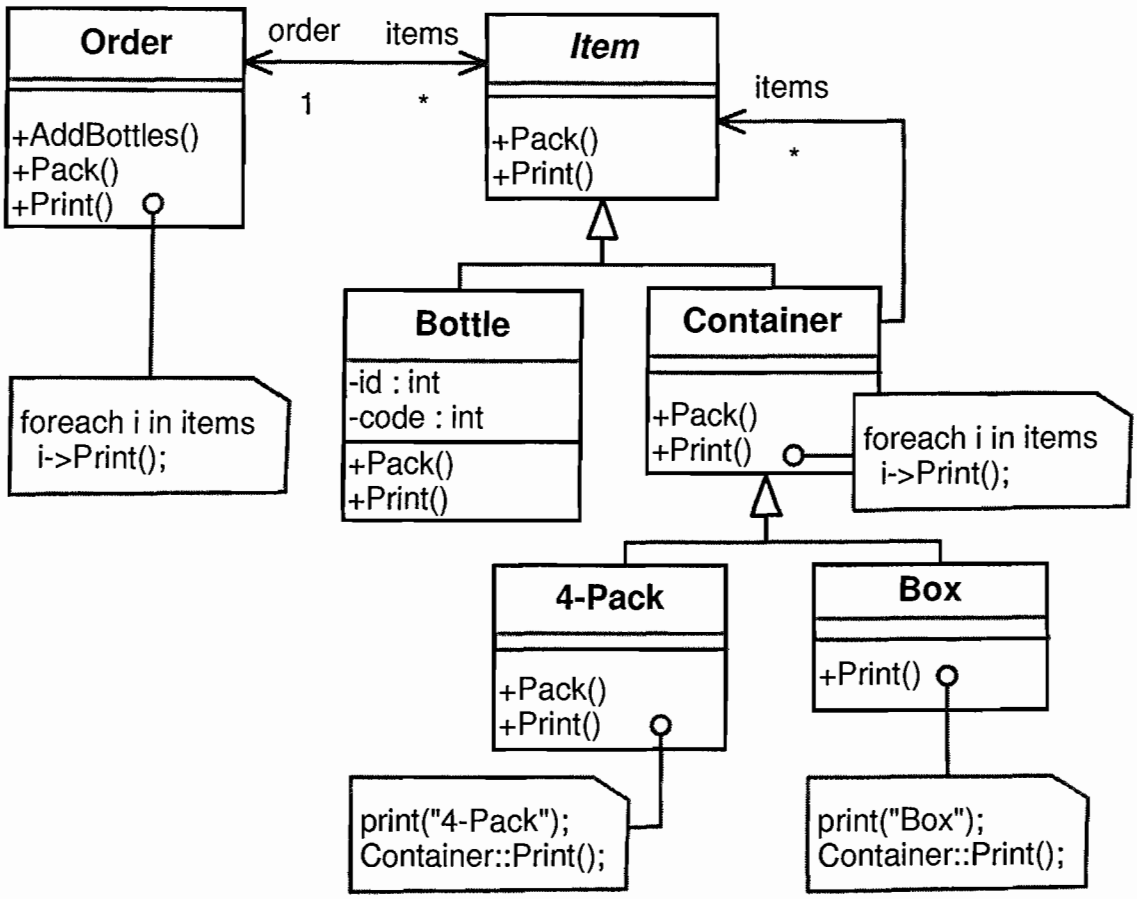

Fig. 2

(TURN OVER

2 (a) Describe the roles of semaphores and signals in concurrent programming and briefly explain how they are implemented. [20%]

(b) Figure 3 shows a UML class diagram of the implementation of a protected buffer for use in a concurrent program. The `ProtectedBuffer` class uses a semaphore and a signal to ensure safe access to a queue of elements of type `T`. Write suitable implementations for the `bput` and `bget` access functions in C++. You may assume that there is sufficient memory to ensure that the queue never becomes full. [30%]

(c) The `ProtectedBuffer` class is now used to implement the packet concentrator shown in Fig. 4. The concentrator collects packets input via its N input channels `in[0]..in[N-1]` and outputs them as a single stream via its output buffer `out`. Explain why the following code for the packet concentrator does not provide an acceptable solution either with or without the `if` statement on line 6: [20%]

```
1   ProtectedBuffer<Packet> in[N];
2   ProtectedBuffer<Packet> out;
3   Packet p;
4   do {
5      for (i=0; i<N; i++)
6         if (in[i].size()>0) {
7            p = in[i].bget();
8            out.bput(p);
9         }
10  } until forever;
```

(d) Outline acceptable solutions to the concentrator problem using:

(i) multiple threads;

(ii) an additional buffer.

[30%]

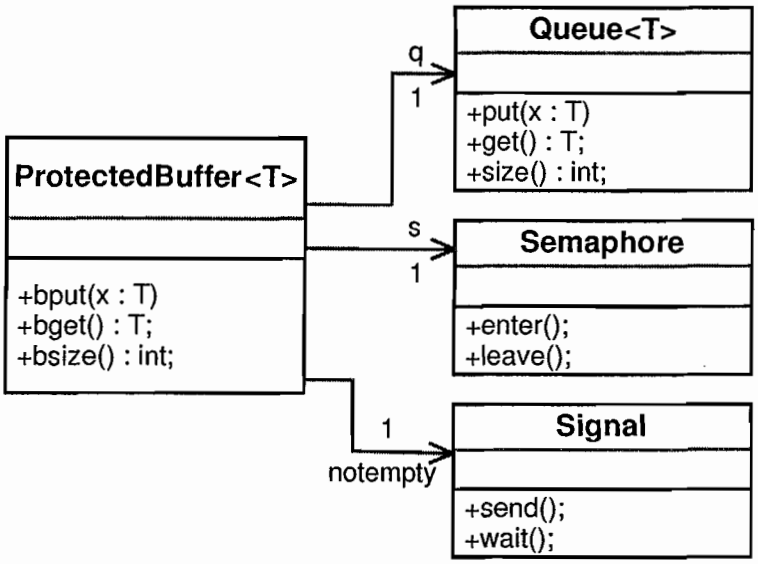
Queue<T>
+put(x : T)
+get() : T;
+size() : int;
q
1
ProtectedBuffer<T>
+bput(x : T)
+bget() : T;
+bsize() : int;
s
1
Semaphore
+enter();
+leave();
1
notempty
Signal
+send();
+wait();

Fig. 3

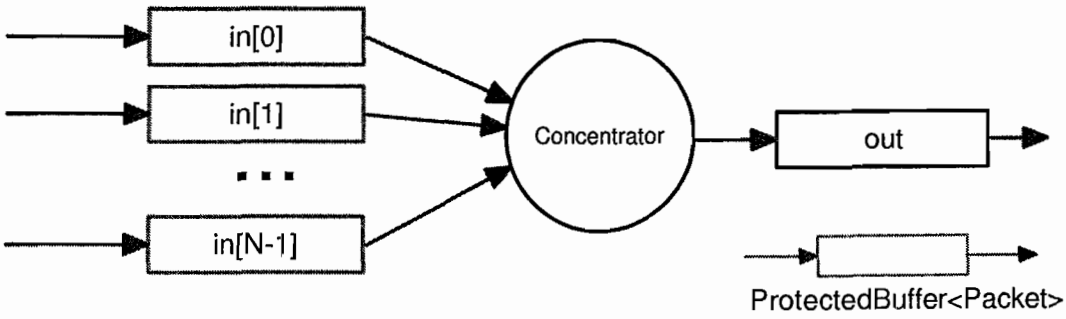
in[0]
in[1]
in[N-1]
Concentrator
out
ProtectedBuffer<Packet>

Fig. 4

3 (a) Describe the role of an interface definition language (IDL) in distributed systems. Explain why the direct use of C++ class interfaces would not be appropriate for this role. [20%]

(b) Figure 5 shows a CORBA interface specification for a simple remote patient record system called "Med-Online" which allows patient records to be downloaded from a central database and, if required, modified and uploaded.

(i) Why do the function arguments require the direction modes `in` or `out` to be specified?

(ii) Why are two remote interfaces provided? What is the role of each?

[20%]

(c) A software system written in C++ provides the class `PatientFactory` shown in Fig. 6 to encapsulate the remote Med-Online interface. Outline the C++ code required to implement the `PatientFactory` constructor and the `getPatient` function. [30%]

(d) In operation, the remote access to patient records is found to be very slow. It is noticed that the same core set of patient records are frequently downloaded from the remote server but only rarely updated. Describe with the aid of a UML class diagram an extension to the software which would reduce access times to frequently used patient records. [30%]

(cont.

```
module MedOnline {
   typedef char PatientId[10];
   typedef char AccessCode[8];
   typedef .... PatientRecord;

   interface Patient {
      boolean getPatientRecord(out PatientRecord pr);
      boolean putPatientRecord(in  PatientRecord pr);
      //each interface function returns false on error
   };
   interface PatientFactory {
      Patient getPatient(in PatientId pid, in AccessCode ac);
   };
}
```

Fig. 5

```
class Patient {
public:
      bool getPatientRecord(PatientRecord& pr);
      bool putPatientRecord(PatientRecord pr);
};
class PatientFactory {
public:
   PatientFactory(CORBA::ORB_var orb, string ior);
   // create a Patient Factory given an Object server (orb)
   // and an interoperable object reference for MedOnline (ior)
   Patient * getPatient(PatientId pid, AccessCode ac);
private:
   PatientFactory_var pfvar;
};
```

Fig. 6

(TURN OVER

4 (a) Explain what is meant by a *virtual function* and explain how virtual functions are implemented in languages such as C++. [20%]

(b) A new on-line store is commissioning a software system to perform stock control. Part of the specification for the software reads as follows:

"Products for sale will be divided into a number of categories (e.g. books, DVDs, etc.) and each category contains a range of products (e.g. specific books). Each product has a name, a price and a stock position. The stock position records the actual stock level and a desired minimum stock level. When a product is purchased, the stock position is updated, and when the stock level falls below the minimum level the stock status changes from normal to low. A re-stock order is then generated and an estimated delivery time is recorded for display to potential customers. All inventory is stored in a warehouse consisting of a number of bins. Every bin holds a number of products and every product records which bin it is stored in. When a product is re-stocked, the corresponding stock level is updated."

Using good design principles, draw a UML class diagram which shows the main classes that will be needed for this software, the relationships between these classes and the main attributes and operations that they should support. You do not need to give any pseudo-code annotations in your answer. [35%]

(c) Describe the principle of the decorator design pattern and explain the role that virtual functions play in its implementation. A class diagram for the decorator pattern is not required. [15%]

(d) Some time after installation of the on-line store, an extension is required to the software to support special offers on various products. Two kinds of special offer are to be implemented which can be used individually or in combination. The first type of offer is a simple discount on the price of the product; and the second type of offer is to give additional loyalty points to the customer. Furthermore, the upgraded software must be sufficiently flexible to allow further special offer schemes to be introduced without major changes to the installed code. Explain how the decorator design pattern can provide a solution to this upgrade problem and illustrate your answer by showing the changes that would be needed to your UML class diagram. [30%]

END OF PAPER