I have neither given nor received unauthorized aid on this examination, nor do I have reason to believe that anyone else has.

_____
Name (printed)

_____
Signature

C S 330
Object-Oriented Programming and Design
Final Exam
Fall 2006

Instructions: This is a closed-book, closed-notes exam. All work is to be done on these pages. There are a total of 180 points worth of questions on this test. The point values assigned to each question represent a rough indicator of the difficulty and/or level of detail expected. Use your time accordingly.

Your answers, especially where judgments or explanations are requested, should be precise and to the point. Completeness and precision are important, but extraneous material will not be rewarded and may actually cost you points if it gives me the impression that you do not know what part of your own answer is actually important. Remember, it is up to you to demonstrate that you know the material – it is not up to me to try to ferret it out.

Questions that are marked with an asterisk (*) are to be answered on the **backs** of the test pages. Be sure to label your answers clearly. For other questions, if you need more room for a particular question than is provided, you may also use the backs of these pages but be sure to label your answers clearly.

The final page of this exam contains some Java and C++ reference material that may be helpful to you.

There are 12 pages on this test, including this cover page and the final reference page.

1. (8 pts) What are the four activities ("work-flows") that take place during each phase of the unified model of software construction?

2. (8 pts) What is "downcasting"? Why is it that C++ programmers avoid downloading but Java programmers use it frequently?

3. (8 pts) Programming instructors have always tried to convince students to use meaningful names for their variables, functions, etc. Explain why this is even more important in OO software development than it had been in older styles.

4. (8 pts) A system has been designed according to the MVC pattern. Suppose that the documentation for this system includes a sequence diagram that shows three classes, one each of type "M", "V", and "C". In this diagram, what possible (call) arrows will definitely not appear?

5. (8 pts) In an early paper discussing multiple inheritance, Wiener and Pinsion defined a class `IntegerArray` (an Array of integers) by multiple inheritance from two classes: `Array` (a general-purpose array-of-objects container) and `Integer`.
   Is this a good idea? Explain.

6. * (21 pts) Give UML diagrams illustrating the following class and object relationships. In all cases, be as precise as possible.

   (a) A project has one or more people assigned to it as staff, and a person may be assigned to more than one project.

   (b) The company is divided into divisions, and each employee is hired by a single division.

   (c) A company's personnel include both salaried and hourly workers.

7. (18 pts) The code below is taken from the textbook formerly in use for CS 451, "Survey of Software Engineering". It illustrates the author's design for an automatic teller machine, a design arrived at via a non-OO design technique.

```
INPUT
loop
   repeat
      Print_input_message ("Welcome - please enter your card");
   until Card_input;

   Account_number := Read_card;
   Get_account_details (PIN, Account_balance, Cash_available);

PROCESS
   if Invalid_card(PIN) then
      Retain_card;
      Print ("Card retained - please contact your bank");
   else
   repeat
      Print_operation_select_message;
      Button := Get_button;
      case Get_button is
         when Cash_only =>
            Dispense_cash (Cash_available, Amount_dispensed);
         when Print_balance =>
            Print_customer_balance (Account_balance);
         when Statement =>
            Order_statement (Account_number);
         when Check_book =>
            Order_checkbook (Account_number);
      end case;
      Print ("Press CONTINUE for more services or STOP to finish");
      Button := Get_Button;
   until Button = STOP;

OUTPUT
   Eject_card;
   Print ("Please take your card);
   Update_account_information (Account_number, Amount_dispensed);
end loop;
```

The author asserts that "An object-oriented design would be similar and would probably not be significantly more maintainable."

(a) Explain why, in fact, a good OO design would *not* be similar to this one.

(b) Describe a circumstance in which the OO approach would be noticably easier to maintain.

(c) What design pattern/technique would you use to implement the functionality of the ATM buttons?

8. * (18 pts) Consider the construction of a system for inventory tracking and control:

> Fancy Oriental Objects (FOO), Incorporated is a mail-order company specializing in overpriced knickknacks and gimcracks imported from Eastern Asia. FOO processes orders via a number of regional warehouses. Each warehouse has responsibility for inventory management and order processing. Because demand of some products varies by region, the inventory at the warehouses may vary considerably. Also, the products offered tend to vary considerably from year to year.
>
> FOO would like to have a common inventory- and order-tracking system across all of its warehouses. Key functions include:
>
> - Tracking inventory as it enters each warehouse, shipped from a variety of suppliers.
> - Tracking customers' orders as they are received from the the central mail-order handling facility.
> - Generating packing slips (instructing the warehouse personnel indicates what items to ship in an order and where to ship it) and invoices (bills) for each order.
> - Generating supply requests for each warehouse.

Based upon this information, draw CRC cards for classes "`Warehouse`", "`Customer Order`", and "`Packing Slip`".

9. * (18 pts) Consider the scenario:

> The mail-order facility (MOF) receives an order from customer Jones. The order is for three items. After checking on warehouse stock levels, the MOF determines that the Norfolk warehouse can supply the first two items, and the Memphis warehouse can supply the third. the order is divided accordingly and the packing slips for the two parts of the order are sent to the warehouses.
>
> Each warehouse receives the packing slip, assembles the order, and ships it to the customer address listed on the slip.
>
> The MOF prints a single invoice, which is mailed to the customer.

Draw a sequence interaction diagram for this scenario.

10. (15 pts) FOO frequently runs sales and complicated sales promotions (e.g., "Buy two pewter figurines from the 'Famous Chinese Poets' series and receive free shipping if the order was placed on the 1st or 3rd Thursday of the month.")
Each promotion is coded as a subclass of

```
class Promotion {
public:
   Promotion (int code) thePromotionCode = code;
   int promotionCode() const return thePromotionCode;
   virtual string canBeAppliedTo (const Invoice&) const = 0;
   virtual void applyTo (Invoice&) const = 0;
private:
   int thePromotionCode;
};
```

The `applyTo` function will modify item prices and shipping prices in the invoice. The `canBeAppliedTo` function is used to check and be sure that a particular promotion is applicable or not. If applicable, an empty string is returned. If not applicable, the function returns a non-empty string containing an explanation suitable for adding to the message area of the invoice.
The `Invoice` class is considerably more complicated. A portion of its interface is:

```
class Invoice {
   ⋮
   Customer getCustomer() const;
   void addToMessageArea (string);
   Dollars getPrice (int itemNumber) const;
   void setPrice (int itemNumber, Dollars price);
   Dollars getShipping () const;
   void setShipping (Dollars price);
   ⋮
   void applyPromotions (int numberOfPromotions,
                         Promotion** promotions);
   ⋮
};
```

(the `Promotion**` type is an array of pointers to `Promotion`).
Write the body for `Invoice::applyPromotions`.

11. (10 pts) Rewrite the `Promotion` class declaration from the previous problem in Java.

12. (10 pts) Assume that the `Invoice` class has also been rewritten into Java, and that `Invoice.applyPromotions` now takes a single parameter: `(Vector promotions)`. Write the declaration and body for `Invoice.applyPromotions`.

13. * (30 pts) FOO also accepts orders by telephone. The telephone ordering system is multi-threaded. Each telephone operator is represented by a thread that pops up a form for the operator to fill in, then submits the order when the operator has completed all the required information. An `OrderProcessing` class provides the code for processing orders, dispatching new packing slips to the warehouses and creating the appropriate invoices.

    In the original design for the system, the telephone operator threads look like:

```
public class TelephoneOperator extends Thread {
    ⋮
  public void run()
  {
    while (true) {
      displayOrderForm();
      if (formFilledOutCorrectly())
        {
           orderProcessing.processOrder(form.getOrder());
        }
     }
   }
```

    and the order processing object looks like:

```
public class OrderProcessing {
    ⋮

  public synchronized
  void processOrder(CustomerOrder co)
    {
      ⋮ // (may take several minutes)
    }
```

    However, in practice it turned out that the telephone operators often submitted clusters of orders in short bursts (e.g., after one of the company's television commercials had just aired). With this design, no operator could complete an order until the order processor was done with its current order, and even then the operator might have to wait if some other operator's thread got the CPU first.

    To smooth out the system performance, it has been suggested that the order processor be turned into a separate thread, with a queue of orders used as a "buffer" to collect any temporary clusters of orders until the could be processed.

    Below are several proposals for the interactions among these threads. (The `sleep(k)` command that appears in some of these simply places the thread into a blocked state for a period of at least k milliseconds.)

    Which of these will work correctly, and, for those that do not, what is the nature of the problem?

(a) The order queue is monitored and the is empty/full status tests are hidden.

```
public class OrderQueue {
  public OrderQueue () {...}

  public synchronized
  void addOrder(CustomerOrder co)
    {
      while (isFull()) wait;
       ⋮
      notifyAll();
    }

  private synchronized
  boolean isEmpty() {...}

  private synchronized
  boolean isFull() {...}

  public synchronized
  CustomerOrder getOrder()
    {
      while (isEmpty()) wait;
       ...
      notifyAll();
    }
}
```

The telephone operator threads look like:

```
public class TelephoneOperator extends Thread{
    ⋮
  public void run()
  {
    while (true) {
      displayOrderForm();
      if (formFilledOutCorrectly())
           orderQueue.addOrder(form.getOrder());
     }
   }
```

and the order processing thread looks like:

```
public class OrderProcessing extends Thread{
    ⋮
  public void run()
  {
    while (true) {
         CustomerOrder co = orderQueue.getOrder();
         process (co);
    }
   }
```

(b) The order queue is monitored with public tests on its status:

```
public class OrderQueue {
  public OrderQueue () {...}

  public synchronized
  void addOrder(CustomerOrder co) {...}

  public synchronized
  boolean isEmpty() {...}

  public synchronized
  boolean isFull() {...}

  public synchronized
  CustomerOrder getOrder() {...}
}
```

The telephone operator threads look like:

```
public class TelephoneOperator extends Thread{
    ⋮
  public void run()
  {
    while (true) {
      displayOrderForm();
      if (formFilledOutCorrectly())
        {
           while (!orderQueue.isFull()) {sleep(100);}
           orderQueue.addOrder(form.getOrder());
        }
     }
   }
```

and the order processing thread looks like:

```
public class OrderProcessing extends Thread{
    ⋮
  public void run()
  {
    while (true) {
      if (!orderQueue.isEmpty()) {
         CustomerOrder co = orderQueue.getOrder();
         process (co);
      } else {
         sleep(100);
      }
     }
    }
```

(c) The order queue is hidden inside the order processing object, which is monitored.

```
public class OrderQueue {
  public OrderQueue () {...}

  public
  void addOrder(CustomerOrder co) {...}

  public
  boolean isEmpty() {...}

  public
  boolean isFull() {...}

  public
  CustomerOrder getOrder() {...}
}
```

The telephone operator threads look like:

```
public class TelephoneOperator extends Thread{
    ⋮
  public void run()
  {
    while (true) {
      displayOrderForm();
      if (formFilledOutCorrectly())
        {
           orderProcessing.addOrder(form.getOrder());
        }
   }
```

and the order processing object looks like:

```
public class OrderProcessing extends Thread{
    ⋮
  private OrderQueue q;

  public synchronized
  void addOrder(CustomerOrder co) {q.addOrder(co);}

  private synchronized void processOrder() {
     while (q.isEmpty()) sleep(100);
     CustomerOrder co = q.getOrder();
       ⋮
  }

  public void run()
  {
    while (true) {
      processOrder();
    }
   }
```

References: Some Java Classes and Interfaces

Some of these may be useful to you. In many cases, I have omitted functions that we have not discussed or that are clearly irrelevant to the questions on this exam.

interface Enumeration

```
boolean hasMoreElements()
Object nextElement()
```

class Object

```
Object clone()
boolean equals(Object obj)
void notifyAll()
void wait()
```

class Observable

```
Observable()
void addObserver(Observer o)
void deleteObserver(Observer o)
void notifyObservers()
```

interface Observer

```
void update(Observable o, Object arg)
```

class Thread

```
static void sleep(long milliseconds)
void run()
void start()
```

class Vector

```
boolean add(Object o)   // appends to end
void clear()
Object clone()
boolean contains(Object elem)
Object get(int index)
Enumeration elements()
boolean equals(Object o)
boolean isEmpty()
Object remove(int index)
Object set(int index, Object element)
```

Some UML symbols