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
Midterm Exam
Spring 2007

Instructions: This is a closed-book, closed-notes exam. All work is to be done on these pages.

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

There are a total of 70 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 judgements 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.

1. (6 pts) What are the major phases of the Unified Process Model for software development?

   inception, elaboration, construction, transition

2. (5 pts) The terms "design", "implementation", and "validation" occur in both the waterfall and unified process models, but the two models use these terms differently. Explain that difference.

   In the waterfall model, these are the names of major phases, but in the Unified model these are names of some of the workflows that pervade all phases of development.

3. (6 pts) In OOP, the C++ "switch" or other multi-way branch functions tend to be replaced by instances of dynamic binding. Why is this considered a good thing to do?

   Application code is simplified because special cases are taken out of the application and grouped into the various subclasses.

   Each special case in the subclass code tends to be comparatively simple.

   New special cases can be added by creating additional subclasses, with no change required to the application code.

   (Full credit for any two of the above 3 reasons.

   No credit for inventing your own question and answering that one instead of the one I asked! This question asks why the use of dynamic binding is good. Many people responded by telling me the definition of dynamic binding. Some simply repeated the question back at me as a statement ("because we replace the branch statement by dynamic binding"). A few people did a "mind dump" of every detail they could remember about dynamic binding, telling me at great length about virtual functions, inheritance hierarchies, accessing objects via pointers, etc., without ever addressing the question I really asked.)

4. * (15 pts) Consider the analysis of a small library.

    The library's holdings include copies of a variety of publications including books, audio books, CDs and DVDs, periodicals and newspapers. There may be many copies of the same publication in the holdings.

    Patrons may check out an item and return an item. There is a predetermined limit on the number of items that may be simultaneously checked out by any patron. Library staff may view a list of items currently checked out by a patron.

    Library staff may add/remove items to/from the library.

    Patrons and staff may search the holdings for a list of items with a particular title, by a particular author, or in a particular subject area.

    Based upon this information, draw CRC cards for the 5 most important classes in this world. ("Importance" should be judged by the number of responsibilities and collaborators.)

    The library's holdings include copies of a variety of publications

    - Library has a collection of copies of publications as an attribute
    - I don't regard "holdings" as a separate class - it seems to be simply the name of an attribute of the library. But I would accept it.

    including books, audio books, CDs and DVDs, periodicals and newspapers. There may be many copies of the same publication in the holdings.

    - Copies of publications are not the same thing as a publication. (Each copy is related (N:1) to some publication.)

    Patrons may check out an item

    - "Item" is a synonym for "copy of a publication".
    - Library has Patrons (attributes)
    - Responsibility: checkOut(item), /research/java/j2sdk1.4.2/
        – probably responsibility of Library, but would accept checkout(Library) as a responsibility of Item, or even checkOut(Item) as a responsibility of LibraryStaff. The one thing I would not accept is a claim that this is a responsibility of Patron.
    - Patron collaborates with (sends messages to) Library (or Item or Staff)

    and return an item.

    - Responsibility: return(item).
        – probably responsibility of Library, with same alternate possibilities as for checkOut
    - Patron collaborates with Library

    Library staff may view a list of items currently checked out by a patron.

    - Library has Staff (attributes)
    - Responsibility of Patron: getCheckedOutItems(),
    - Staff collaborates with Patron

    or

    - Responsibility of Library: getCheckedOutItems(Patron),
    - Staff collaborates with Library

    Library staff may add/remove items to/from the library.

    - Responsibility of library: add/remove items, *NOT* a responsibility of the staff.
    - Staff collaborates with Library

    Patrons and staff may search the holdings for a list of items with a particular title, by a particular author, or in a particular subject area.

- Publications have authors, titles, subject as attributes.
- Responsibility of Library: search (criteria) or separate searches on various fields, Staff and patrons collaborate with Library
  - Also plausible, introduce the notion of a catalog that gets searched (not mentioned in problem, but might be considered common knowledge about libraries). Catalog would have to appear as an attribute of the library.

So, most active classes would be Library, Patron, Staff, Publication, Item. Holdings/Catalog may be used instead of either of the last two.

Library	
add/remove Item	
maintain set of Patrons	
maintain set of Staff	
checkOut (Item)	
return (Item)	
search (criteria)	

Item	
getPublication	

Publication	
get/set Author	
get/set Title	
get/set Subject	

Patron	
getCheckedOutItems	Library

Staff	
	Library
	Patron

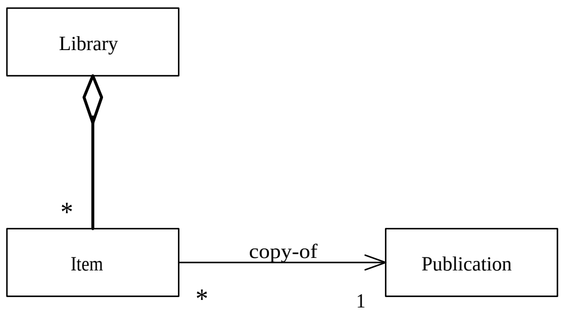
5. * (12 pts) For each of the following statements about the library, draw a separate UML diagram that captures the sense of the statement.

   If a portion of the statement is italicized, pay particular attention to portarying that portion.

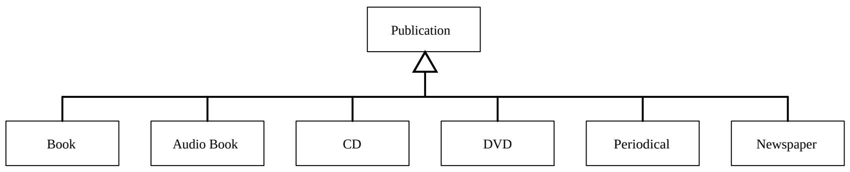
   (a) *The library's holdings include copies of a variety of publications* including books, audio books, CDs and DVDs, periodicals and newspapers. *There may be many copies of the same publication in the holdings.*

   (b) The library's holdings include copies of *a variety of publications including books, audio books, CDs and DVDs, periodicals and newspapers*. publication in the holdings.

   (c) Patrons may check out an item and return an item. There is a predetermined limit on the number of items that may be simultaneously checked out by any patron.

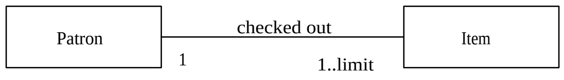


(a)



(b)

(c)



The remaining questions are concerned with a To-Do List, which can be viewed as a container of tasks. There will be many different kinds of tasks (Projects, Calls, Errands, etc.) organized into an inheritance hierarchy.

```
class Task {
public:
   ⋮
  Task(String title, Interval when);

  Interval getInterval() const;
  void setInterval (Interval);

  string getTitle() const;
  void setTitle (string);

  virtual Task* clone() = 0;
   ⋮
};

class ToDoList {
public:
   ⋮
  ToDoList();

  void addTask(const Task& e);
  // Add a task to the calendar

  ToDoList tasksDuring (Interval interval);
  // create a smaller to-do list containing only the tasks
  // overlapping the indicated interval.
   ⋮
private:
  int numberOfTasks;
  Task** tasks; // array of pointers to tasks
};
```

Note: The date/time concepts employed here are as follows:

- A **time**, **date**, or **date-time** are instants in time. Examples are, respectively, “1:30PM”, “3/5/2007” and “3/5/2007 at 1:30PM”.

- A **duration** is a length of time. For example, consider a task that runs from 3:00PM to 4:15PM. This task has a **duration** of one hour, 15 minutes.
- An **interval** is a date plus a starting time and an ending time, or, alternatively, a date plus a starting time and a duration. An example of an interval would be March 5, 2007, from 3:00PM-4:15PM.

The Interval class mentioned in the interfaces above looks like:

```
class Interval {
public:
  //
  // An interval is a period of time beginning with a starting time
  //  and running up to, but not including, its stop time.
  //
  Interval ();
  Interval (DateTime start, DateTime stop);
  Interval (DateTime start, Duration length);

  DateTime getStart();
  DateTime getStop();
  Duration getDuration();

  bool contains (DateTime d);
  bool overlaps (Interval intrvl);

private:
  DateTime theStart, theStop;
};

ostream& operator<< (ostream&, Interval);
```

6. (10 pts) What failings does the `Interval` class have, according to our class designer's checklist?

    Post-conditions are not documented. In particular, just what is created by the default constructor is far from obvious.

    It is not const-correct. The "get" operations and the two boolean operations should be marked as const.

    No comparison operations (== and <) are provided.

    Note, however, that there is *not* a big-3 violation here. None of the data members are pointers, so the default compiler-generated big-3 should be just fine.

7. * (10 pts) Write the function body for `ToDoList::tasksDuring`.

    Honestly, I messed up this question a bit. I intended to set up a situation where you would need to exploit dynamic binding, but somewhere during several revisions of the code, I lost all the virtual functions without noticing it. So this became a more basic problem in working with ADTs. Still gave some people trouble.

```
ToDoList ToDoList::tasksDuring (Interval interval)
  // create a smaller to-do list containing only the tasks
  // overlapping the indicated interval.
{
   ToDoList results;
   for (int i = 0; i < numberOfTasks; ++i)
     if (interval.overlaps(tasks[i]->getInterval()))
       results.addTask(*tasks[i]))
   return results;
}
```

8. (6 pts) In the `ToDoList` class of the preceding problem, `tasks` is an array of pointers to `Task`s, rather than an array of `Task`s. `Task` is an abstract class, so an array of `Task`s would not be legal for that reason, but even if it were not abstract, we would still have wanted an array of pointers. Why?

   If we did not use pointers, then assignments of tasks into the array would truncate the data members added by each subclass.

   Alternatively, we can note that the use of pointers facilitates the use of dynamic binding, but this is not quite as important because, as it happens, there's no dynamic binding going on that we can see and, in any case, there are easy ways to work around the rule that dynamic binding can only take place when using a pointer or reference.