University of Waterloo

University of Waterloo
Final Examination

Term: Winter **Year:** 2007

Place sticker here.

Last Name: _____

First Name: _____

ID: _ _ _ _ _ _ _ _ _

Signature: _____

Course Abbreviation and Number:	CS246
Course Title:	Software Abstraction and Specification
Lecture Sections:	001
Instructors:	A. J. Malton
Date of Exam:	April 14, 2007
Time Period	**Start time:** 1600 **End time:** 1830
Duration of Exam:	2.5 hours
Number of Exam Pages:	14 (including this cover sheet)
Exam Type:	Closed book
Additional Materials Allowed:	None.

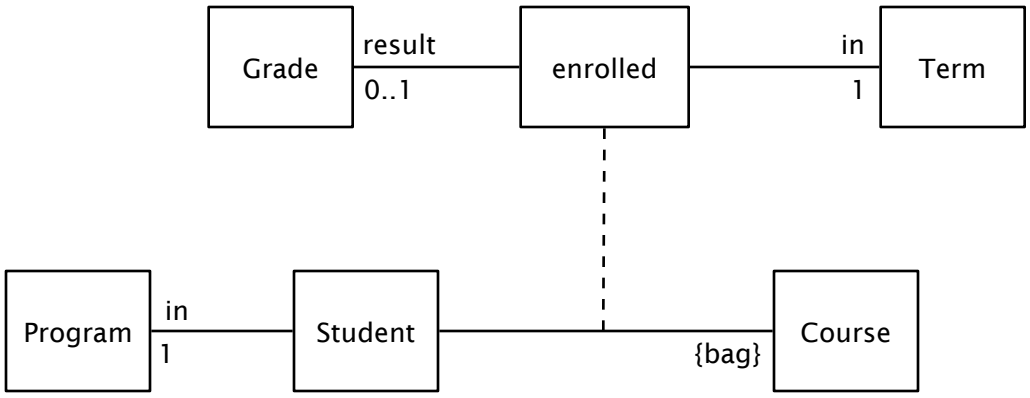
Instructions: (Read carefully before the exam begins):

1. Before you begin, make certain that you have one Exam Booklet with pages numbered 1–14 printed single-sided.
2. The marks assigned to each question are shown at the beginning of the question; use this information to organize your time effectively.
3. Place all your answers in the spaces provided on these pages.
4. You do not need to document your code unless it is specifically required by the question.
5. Questions will not be interpreted. Proctors will only confirm or deny errors in the questions. If you consider the wording of a question to be ambiguous, state your assumptions clearly and proceed to answer the question to the best of your ability. You may not trivialize the problem in your assumptions.
6. Cheating is an academic offense. Your signature on this exam indicates that you understand and agree to the University's policies regarding cheating on exams.

Question	Marks Given	Out Of	Grader Initials
1		12	
2a		10	
2b		10	
2c		5	
3a		10	
3b		10	
3c		10	
4		13	
5a		15	
5b		15	
Total		100	

Question 1: Modeling Associations

The diagram below shows part of the domain model of a university information system.
[12 marks = 3 marks each (right=3, not completely wrong=1, no answer=0, completely wrong=-1)



Choose the one ***best*** completion of each of the following statements.

1) For implementation in C++, there should be a constructor
   a) `Student ();`
   b) `Student (Term & in, Program & in);`
   c) `Student (Program & in);`
   d) `Student (Term & in, Grade* result);`
   e) `Student (std::multiset<Course*> & courses, Program & in);`

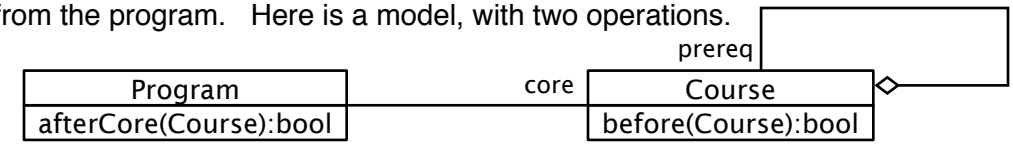
2) An operation to change a student's grade in a course in which he is enrolled should be declared
   a) `void setGrade (Student &);`
   b) `Grade setGrade (Student &);`
   c) `void setGrade (Student &, Course &, Grade &);`
   d) `void setGrade (Student &, Course &, Term &, Grade &);`
   e) `Term & setGrade (Student &, Course &, Term &, Grade &);`

3) A student *cannot*
   a) be enrolled in the same course twice in the same term.
   b) be enrolled in two different courses in the same term.
   c) have two different grades for the same course in the same term.
   d) be enrolled in a course without specifying which term.
   e) have the same grade in two courses in the same term.

4) The implementation of the association enrolled
   a) must be as a member variable of class Student.
   b) could be without using the Standard Template Library.
   b) could be using a member variable of type Course* .
   c) must provide navigation from Student to Course and from Course to Student.
   d) could provide navigation from Student to Term.

Question 2: Implementation of a Method

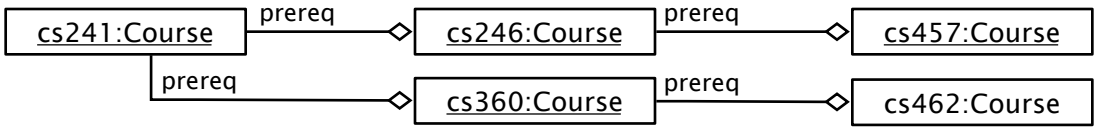
[25 marks] A course *p* is a *prerequisite* of another course *q* if a student must have credit for *p* in order to enroll in *q*. A course *p* is a *core course* of a program is a student must have credit for *p* in order to graduate from the program. Here is a model, with two operations.



A simple and familiar implementation of the associations above is as sets of pointers, as below:

```
class Course
{
    /* ... */
    std::set<Course*> _prereq;
    /* ... */
public:
    /* ... */
    bool before (Course &);
};
```

```
class Program
{
    /* ... */
    std::set<Course*> _core;
    /* ... */
public:
    /* ... */
    bool afterCore (Course &);
};
```



(a) [10 marks] A course *p* is *before* a course *q* if a student must have passed *p* before taking *q*, due to a sequence of one of more prerequisites. In the diagram above, cs241 and cs246 are before cs457. In the space below, complete the definition so that `p.before(q)` is true if and only if *p* is before *q*.

```
bool Course::before
```

Question 2: Implementation of a Method

(b) [10 marks] A student in a program can take courses which follow the program's core courses. In the space below, complete the definition so that `p.afterCore(c)` is true if and only if some core course of *p* is before *c*.

```
bool Program::afterCore
```

(c) [5 marks] Do your answers for (a) or (b) above depend on the fact that prereq is an aggregation? Why or why not?

Question 3: Dependency Inversion

It will probably help to read all the parts of this question before you start writing.

[30 marks] Read the Calculator Program on the last two pages of the exam. It is a simple program that behaves as a four-function (+-*/) calculator.

(a) [10 marks] In the space below draw a dependency diagram which shows the classes, types, and packaging in the Program, and the associations and dependency relationships ("uses") between them. Include standard classes used explicitly. **Note**: std::cout is of type std::ostream, and std::cin is of type std::istream, and std::setw(*n*) is of type of type std::_Setw. Assume main is of type Main.

Question 3: Dependency Inversion

(b) [10 marks] In the space below redesign the Calculator dependency graph from part (a) so that

1) the Calculator class is independent of the standard library, and
2) main and the Calculator class are in separate packages, and
3) the behaviour of main is the same as before.

Use dependency inversion in Calculator, and the adapter pattern in main. Show the revised dependencies, including on the standard library. Include associations and any specialization relationships you introduce.

Question 3: Dependency Inversion

(c) [10 marks] In the space below and on the next page, if required, rewrite the code of main and of Calculator to reflect your redesign from part (b). Preserve the source file organization. (To save needless copying you can refer to existing lines of code by line number, or describe specific changes to numbered lines. If you do this, please try very hard to be clear.)

Question 3: Dependency Inversion

This page contains more space for your answer to 3c, if required.

Question 4: Managing Change

[13 marks]

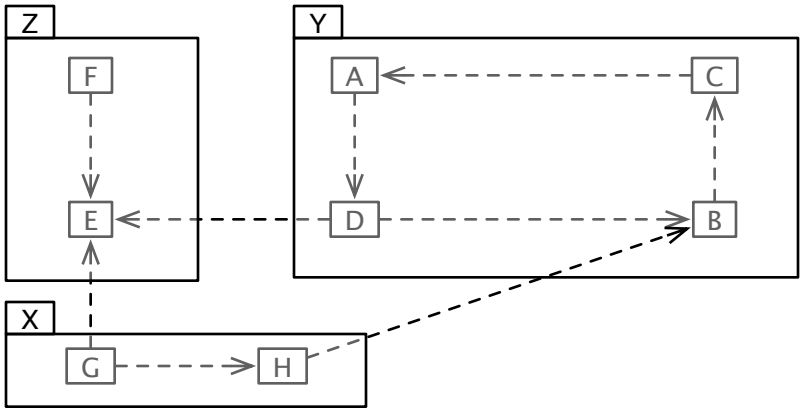
a) [3 marks] What are the three characteristics of good modularity?

1.

2.

3.

Now consider the dependency model below.

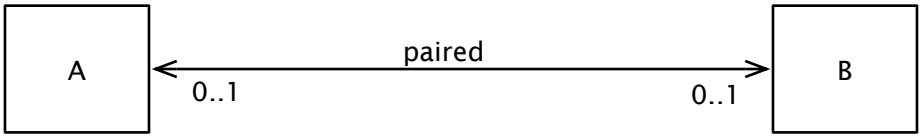


b) [5 marks] In the model above, if class E changes, which classes are potentially affected by change propagation? Why those classes?

c) [5 marks] A developer is responsible for package X. What action should he expect to take when revisions to class F are released, and why?

Question 5: Adding A New Association

[30 marks] It's required to provide the following association between classes A and B:



but unfortunately the code neither of A nor B can be changed, because they are shared with other projects that do not need the association.

a) [15 marks] Draw a class digram, showing navigation requirements explicitly, which could be used to implement **paired** without modifying A or B. Declare operations to

1) set up a link between an A and a B which are not currently paired to anything;
2) remove a link between an A and a B which are currently linked together;
3) navigate from A to B, as shown; and
4) navigate from B to A, as shown.

Hint: you need to introduce one or more new classes.

Question 5: Adding A New Association

b) [15 marks] Write C++ code, organized into a header file and an implementation file, to implement your design from (a) above. You can assume that A.h and B.h are available by inclusion.

Question 5: Adding A New Association

This page contains more space for your answer to 5b, if required.

Calculator Program 1: Calculator.h, main.cc
The line numbers are for reference, you can mention them instead of recopying lines.
You can detach this page if you want.

Calculator.h

```
1    # ifndef CALCULATOR_H
2    # define CALCULATOR_H
3
4    class Calculator
5    {
6        float r1;
7        float r2;
8        float r3;
9        float r4;
10       bool inError;
11
12   public:
13       Calculator ();
14       void enter (float);
15       void add ();
16       void subtract ();
17       void multiply ();
18       void divide ();
19       void clear ();
20   };
21
22   # endif
```

main.cc

```
1    # include "Calculator.h"
2    # include <iostream>
3    # include <sstream>
4
5    int main ()
6    {
7        Calculator c;
8        std::string t;
9        while (std::cin >> t)
10       {
11           if (t == "+")
12               c. add ();
13           else if (t == "-")
14               c. subtract ();
15           else if (t == "*")
16               c. multiply ();
17           else if (t == "/")
18               c. divide ();
19           else if (t == "C")
20               c. clear ();
21           else
22           {
23               std::istringstream s (t);
24               float v;
25               if (s >> v)
26                       c. enter (v);
27           }
28       }
29   }
```

Calculator Program 1: Calculator.cc
The line numbers are merely for reference, of course.
You can detach this page if you want.

Calculator.cc

```
1    # include "Calculator.h"
2
3    # include <iostream>
4    # include <iomanip>
5
6    Calculator::Calculator ()
7    {
8       clear ();
9    }
10
11   void Calculator::enter (float v)
12   {
13      if (!inError)
14      {
15            r4 = r3;
16            r3 = r2;
17            r2 = r1;
18            r1 = v;
19            std::cout << "[" << std::setw (10) << v << "]\n";
20      }
21   }
22
23   void Calculator::add ()
24   {
25      if (!inError)
26      {
27            r1 = r1 + r2;
28            r2 = r3;
29            r3 = r4;
30            std::cout << "[" << std::setw (10) << r1 << "]\n";
31      }
32   }
33
34   void Calculator::subtract ()
35   {
36      if (!inError)
37      {
38            r1 = r1 - r2;
39            r2 = r3;
40            r3 = r4;
41            std::cout << "[" << std::setw (10) << r1 << "]\n";
42      }
43   }
44
45   void Calculator::multiply ()
46   {
47      if (!inError)
48      {
49            r1 = r1 * r2;
50            r2 = r3;
51            r3 = r4;
52            std::cout << "[" << std::setw (10) << r1 << "]\n";
53      }
54   }
55
56   void Calculator::divide ()
57   {
58      inError = (r2 == 0);
59      if (!inError)
60      {
61         r1 = r1 / r2;
62         r2 = r3;
63         r3 = r4;
64            std::cout << "[" << std::setw (10) << r1 << "]\n";
65      }
66      else
67            std::cout << "ERROR\n";
68   }
69
70   void Calculator::clear ()
71   {
72      r1 = 0.0;
73      r2 = 0.0;
74      r3 = 0.0;
75      r4 = 0.0;
76      inError = false;
77      std::cout << "[" << std::setw (10) << 0.0 << "]\n";
78   }
```