Student ID#: ________________________

CPSC 410 ? Advanced Software Engineering
Mid-term Examination (Term I 2001-2002) SOLUTION
Instructor: Gail Murphy

Do NOT start until you are informed you can start!

This examination has 7 questions. The question and answer space is from page 2 to page 11. Page 12 provides a short UML reference. Check that you have a complete paper when you are told that you can start the examination.

You have **80 minutes** to complete this exam. Before you start, you must write your student id number on the top of **every** sheet of this exam. ***We will not mark your exam without this information***. You must also sign your name in the space provided below.

Write legibly and in **ink** (not pencil). We also have to be able to read it to mark it.

This exam is closed book and closed neighbour. Notes, book, or other materials are not allowed. Candidates guilty of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action:

?? Making use of any books, papers or memoranda, calculators or computer, audio or visual cassette players, or other memory aid devices, other than those authorized by the examiners.

?? Speaking or communicating with other candidates.

?? Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.

If you have a question during the exam, raise your hand and one of the invigilators will come and answer your question.

The exam is out of 100 marks. The value of each question is indicated. Use your time wisely. If you do not know the answer to a question, move on to the next question and come back to the question at the end.

The exam is double-sided. There are questions on **both** sides of the page.

Good luck!

Print Name:________________________________

Signature:_________________________________

1. ***(Total: 20 points. 2 points each.)*** True or false with justification. For each statement below, **circle** T if the statement is true or F if the statement is false. In the space provided for each statement, write at most **two** sentences explaining your choice.

(a) A system is designed well if its components are highly coupled.

T or [✓ F.] Justification:
A "good" design will have components that are highly cohesive and minimally coupled.

(b) In a system whose software architecture conforms to a 2-tier client/server style, the only responsibility of the client is to provide the graphical user interface to the system.

T or [✓ F.] Justification:

In such a system, the client is "fat", it likely contains business logic as well as GUI functionality

(c) The Software Architecture Analysis Method uses use-cases as a basis for evaluating a software architecture.

T or [✓ F.] Justification:

SAAM uses scenarios that differ from use-cases by encompassing more interactions with the system, such as the interaction of developers performing maintenance tasks.

(d) The repository and blackboard architectural styles share many similarities.

[✓ T] or F. Justification:

Yes, they are both sub-styles of the data centered architecture.

(e) The order in which components in a batch sequential architectural style shall execute must be determined at compile-time.

T or [✓ F.] Justification:

No, it can be determined at run-time.

(f) A client accesses a COM object through at most one interface.

T or [✓ F.] Justification:

No, a client can use multiple interfaces to access a COM object.

(g) When architecting a system using an object-oriented style, it is beneficial to create a view of the architecture showing the classes and the messages between objects of those classes.

[✓ T] or F. Justification:

Such a view can provide a useful dynamic view of how the system will execute.

(h) A system whose architecture conforms to a 3-tier client/server style can scale to handle a large number of users.

[✓ T] or F. Justification:

A number of application servers can be introduced to perform load sharing.

(i) The Observer Design Pattern is an example of an explicit invocation architectural style.

T or [✓ F.] Justification:

Observer supports an implicit invocation architectural style.

(j) When designing using an object-oriented approach, it is good design practice to ensure the class is closed for modification

[✓ T] or F. Justification:

Yes, classes should be closed for modification but open for extension.

2. (Total: 10 points. 2 points each.) Short Answer.
Define the following terms in five sentences or less each.

a. Stakeholder.

A stakeholder is a person or an organization that has an interest in the construction of a software system.

b. Package-oriented Programming..

In package-oriented programming, an application is created by gluing together existing (off-the-shelf) applications and components.

c. Information Hiding.

When an information hiding approach is used during design and implementation, modules are selected to hide design decision(s). Information hiding is achieved through the appropriate choice of interfaces to the module.

d. Architectural Style.

A description of a common way to architect a system, providing a framework on which to base design. A style typically consists of components, connectors, semantic constraints on the components and connectors, and a description of the typical system topology.

e. Product Line.

A collection of systems that share a managed set of features constructed from a common set of software assets.

3. (Total: 10 points) Architectural Views.

a) (8 marks) Briefly describe the 4 views that constitute the “4+1” approach to describing a software architecture.

1) Logical View – this view supports the functional requirements: how will the system perform the functions the user is interested in.

2) Development View – this view supports the developers: It shows the module organization and helps reason about configuration considerations, reuse, etc.

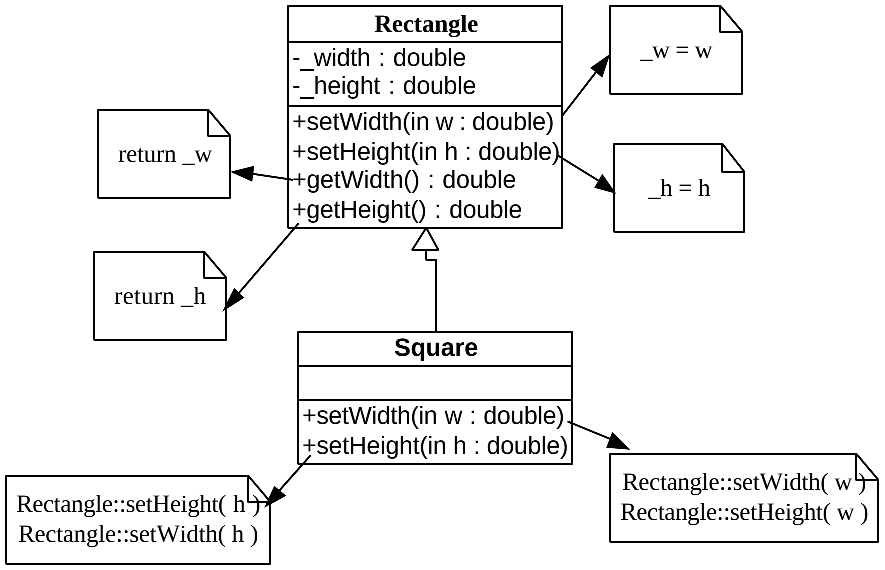
3) Process View – this view supports a run-time view of the system, considering how the system functionality is split into processes and tasks. It helps reason about performance, availability, etc.

4) Physical View – this view considers the hardware environment in which the system will be deployed and considers how the software maps to the hardware.

b) (2 marks) What is the “+1” in this approach?

Scenarios are the “+1”.

4. ***(10 marks)*** As part of an object-oriented design for a graphical drawing editor, similar to the UBCDraw example we worked on in lecture, the following fragment of a class diagram is proposed. Does this class diagram demonstrate "good" object-oriented design? Why or why not?



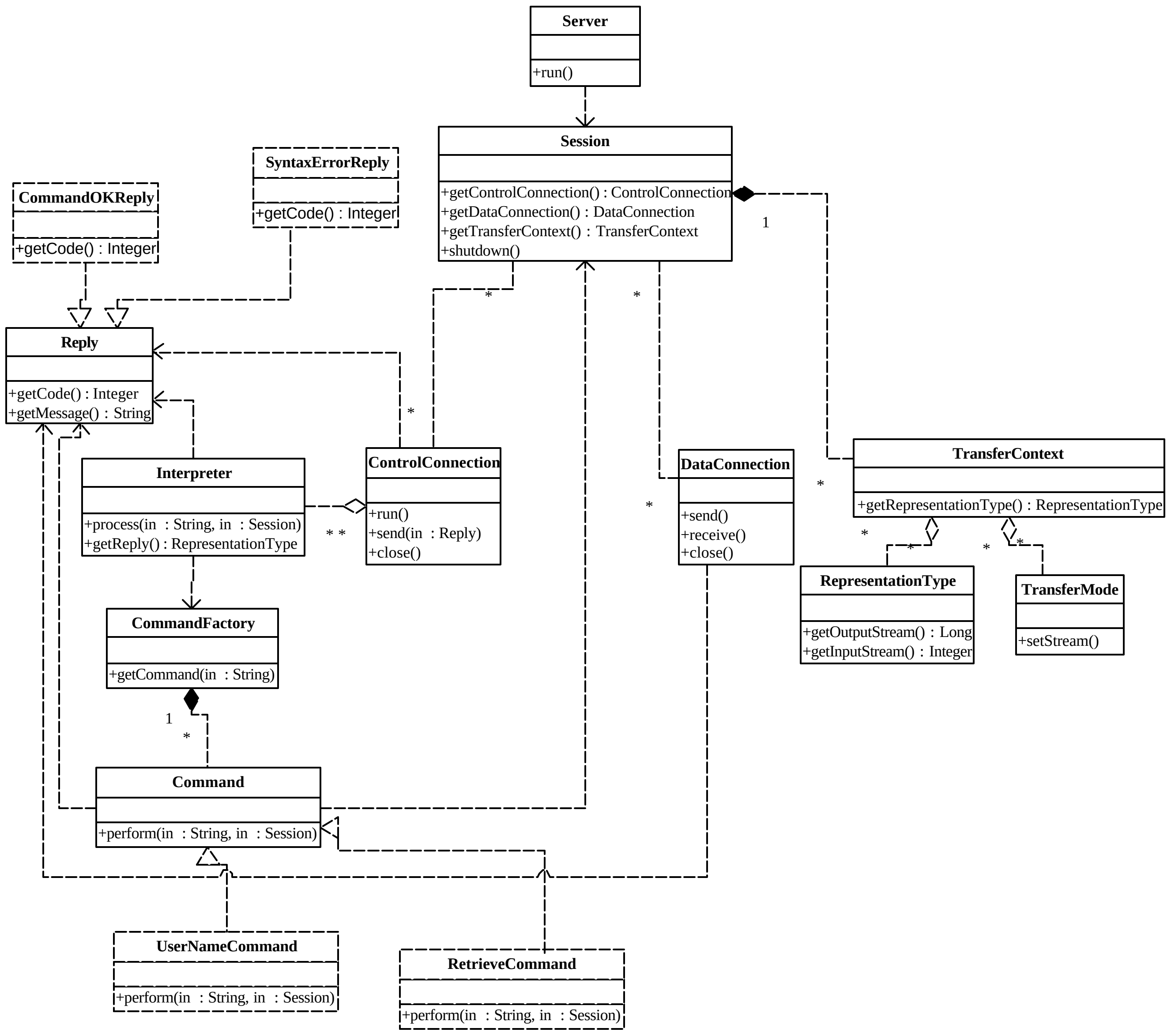
This design does not demonstrate "good" design according to the principles we studied. In particular, the design breaks the Liskov substitution principle. The problem is the redefinition of the setWidth and setHeight methods in Square. These methods have a weaker postcondition than their counterparts in Rectangle. The postcondition of setWidth in Rectangle is that after running setWidth, the_width variable will have the value of the given parameter **and** the _height variable will be unchanged. In Square, setWidth changes the value of _weight and _height! As a result, a Square cannot be substituted everywhere for a Rectangle. Consider the following code…

```
void f( Rectangle r ) {
  int i = 2;
  int j = 3;
  r.setHeight( i );
  r.setWidth(j);
  assert ( r.getHeight() == i );
  assert( r.getWidth() = j );
}

Square s = new Square();
f(s);  --- the first assert will fail because the call to setWidth changed the height value as well.
```

This page is intentionally blank.

5. ***(Total: 15 points***) We studied three design patterns. A sketch of a class diagram representing part of the design of an FTP server is given below.


Server
+run()
Session
+getControlConnection() : ControlConnection
+getDataConnection() : DataConnection
+getTransferContext() : TransferContext
+shutdown()
SyntaxErrorReply
+getCode() : Integer
CommandOKReply
+getCode() : Integer
Reply
+getCode() : Integer
+getMessage() : String
Interpreter
+process(in : String, in : Session)
+getReply() : RepresentationType
ControlConnection
+run()
+send(in : Reply)
+close()
DataConnection
+send()
+receive()
+close()
TransferContext
+getRepresentationType() : RepresentationType
RepresentationType
+getOutputStream() : Long
+getInputStream() : Integer
TransferMode
+setStream()
CommandFactory
+getCommand(in : String)
Command
+perform(in : String, in : Session)
UserNameCommand
+perform(in : String, in : Session)
RetrieveCommand
+perform(in : String, in : Session)
1
*
* *

A `Server` object listens on a defined port for connections from clients. When a connection request is received from a client, a new `Session` object is created to maintain the session. The `Session` object creates a `ControlConnection` (over which commands are sent by the client and responses received), a `DataConnection` (over which data is sent and received), and a `TransferContext` object (that describes how the data should be sent and received). The `ControlConnection` listens for commands from the client and passes each to the Interpreter, which encapsulates the syntax of the command protocol. The `Interpreter` reads and interprets the command to be executed and requests an object encapsulating the details of that command from the `CommandFactory`. Each `Command` class encapsulates the syntax of a particular command. When the `CommandFactory` returns an instance of `Command` to the `Interpreter`, the latter passes the arguments sent from the client and the current `Session` object to the `Command` object so that the appropriate actions may take place. The `Command` object executes the appropriate actions, creates an appropriate `Reply`, and through the `ControlConnection` returns the reply to the client. The `getMessage` operation on `Reply` encapsulates the knowledge of how to form a message from a code; each subclass of `Reply` must correctly implement the `getCode` method to satisfy the protocol setup by `getMessage`.

a. ***(5 points.)*** State which of the design patterns we studied are evident in the FTP design and what parts of the design contribute to those patterns.

The Command pattern is evident in the Command class and its subclasses.
The Template pattern is evident in the getMessage() code of Reply.

b. ***(10 points.)*** Explain the role of the pattern(s) you identified in the design (i.e., why were the pattern(s) used?)

The Command pattern makes it easier to add new commands into the system: the Interpreter will not need to be changed. The Command pattern ensures the implementation of a command is encapsulated.

The Template pattern captures the algorithm of replying and helps make sure it is adhered to consistently.

6. ***(Total: 10 points) Object-oriented Design Fundamentals***

   a. (3 marks) Most object-oriented designs include associations. What is an association and why are associations used in an object-oriented design?

An association is a relationship between two classes. Associations are used to represent data relationships (i.e., for every object of one class there are n objects of another class) and to represent interactions between classes.

   b. (2 marks) Do common object-oriented languages, such as C++ or Java, support the direct implementation of an association? If so, how? If not, how are associations implemented?

No, one must map an association onto object references in each (or one of) the classes depending on whether it is a bi-directional (or unidirectional) association.

   c. ***(3 marks)*** Most object-oriented designs describe invocations. What is an invocation and why are invocations used in an object-oriented design?

   An invocation refers to the execution of an operation on an object (e.g., calling a method on an object).

   d. ***(2 marks)*** Do common object-oriented languages, such as C++ or Java, support the direct implementation of an invocation? If so, how? If not, how are invocations implemented?

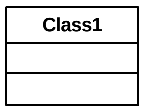
Yes. Method calls provide direct implementation of an invocation.

7. ***(Total: 25 points)*** Compare and contrast the pipe-and-filter and implicit invocation styles according to the following criteria: kinds of components, kinds of connectors, control topology, binding time of control, the flow of control and data, and the purported benefits of the style.

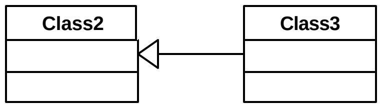
	Pipe & Filter	**Implicit Invocation**
Components	Filters	Arbitrary (processes, objects, etc.)
Connectors	Pipes	Events or messages
Control Topology	Acyclic (accepted arbitrary although its not quite right)	Arbitrary
Binding Time of Control	Invocation or run-time	Write-time, compile-time, run-time (full marks if you just said run-time)
Control Flow	Filters run asynchronously: A Ftiler starts when data arrives. Control follows data (and vice versa)	Components register interest in events. They are invoked when the event is announced.
Data Flow	Data follows control. Continuous data flow, high or low volume.	May not flow in the same direction as an event. Typically sporadic, low-volume data
Benefits	Reuse (of filters) and modifiability	Modifiability

Student ID#: ___________________________

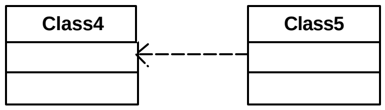
UML Reference



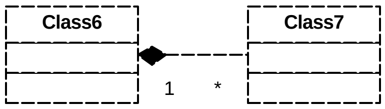
Class



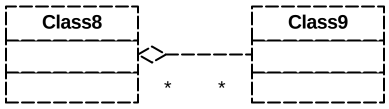
Generalization: Class3 inherits from Class2



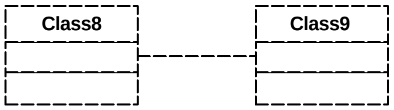
Dependency from Class5 to Class4



Composite Aggregation: Class6 aggregates Class7



Aggregation: Class8 aggregates Class9



Association between two classes

Student ID#: ___________________________