COMP/EECE 7012

Final Exam

Spring 2012

Name: Answer Key

Instructions:

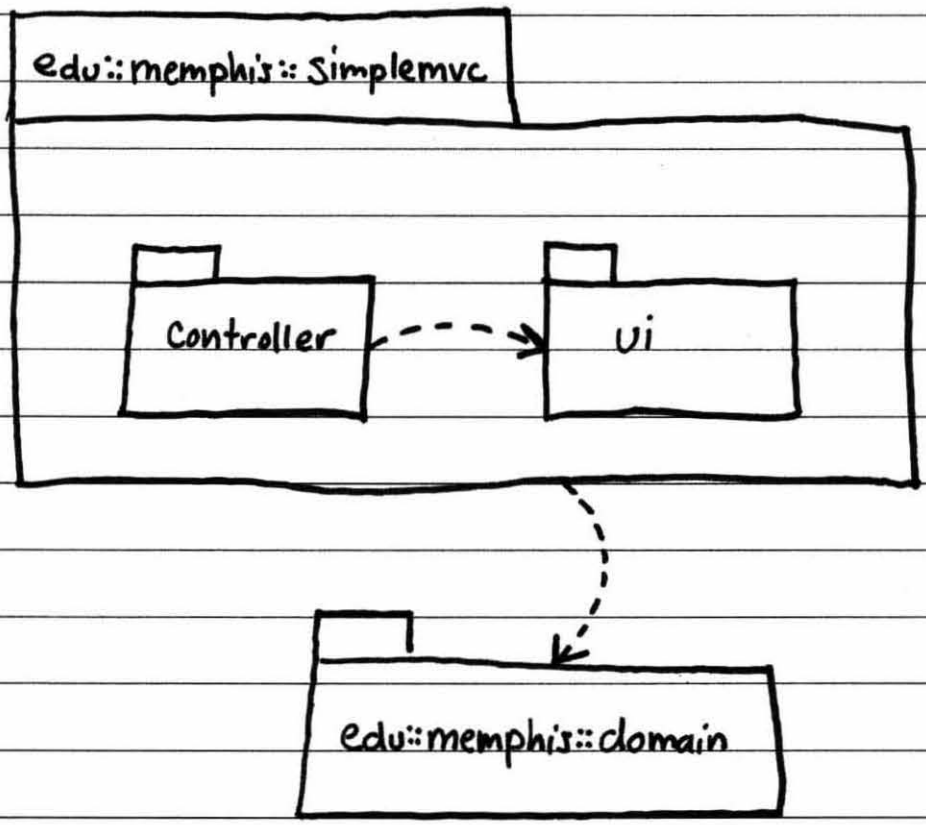
- No bathroom breaks.

- No calculators.
- No cellphones—turn them off.
- No other devices.

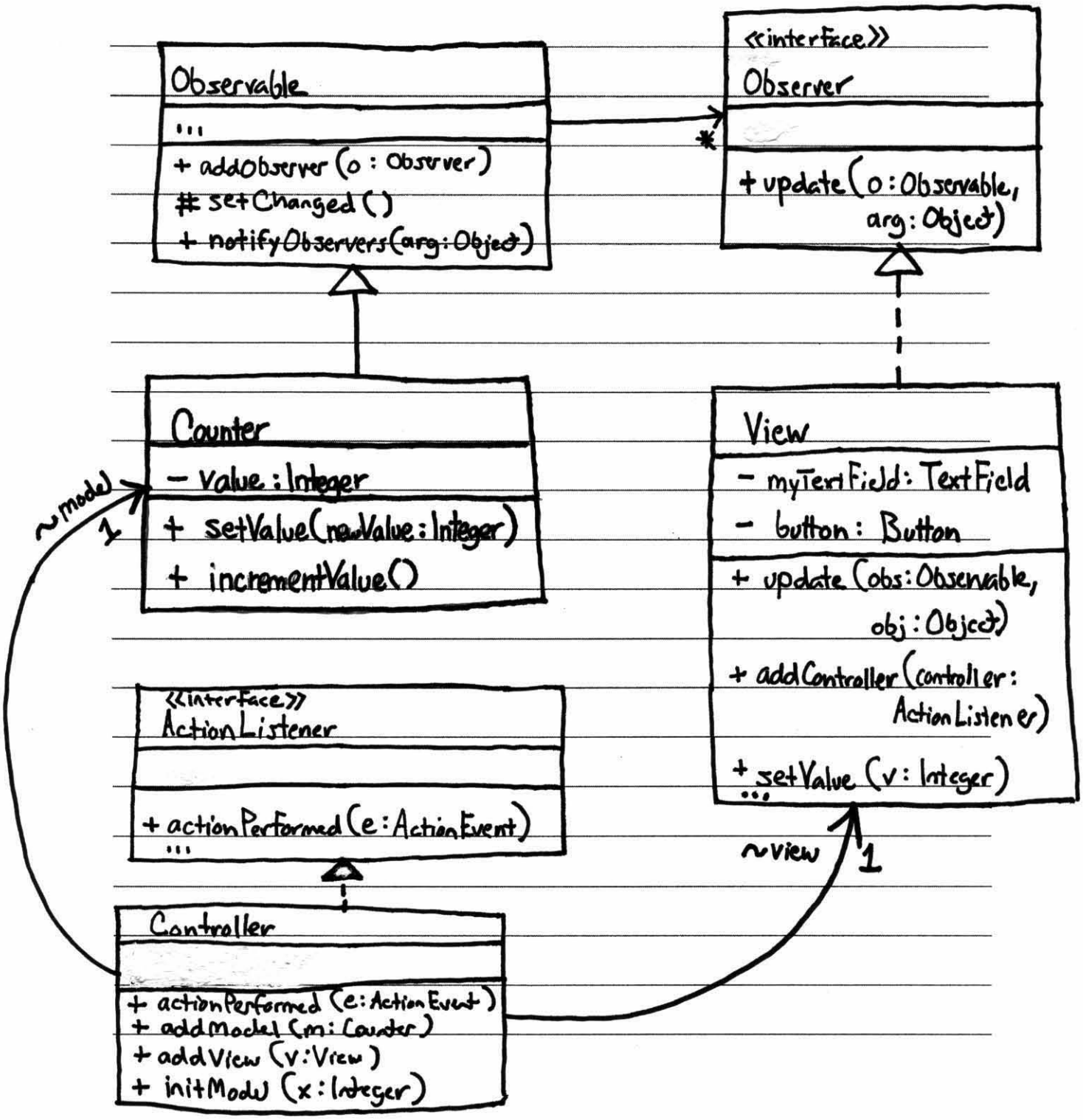
- Closed book.
- Closed note.
- Closed neighbor.

- Verify that you have all the pages.
- Don't forget to write your name.
- Read each question carefully.
- Don't forget to answer every question.

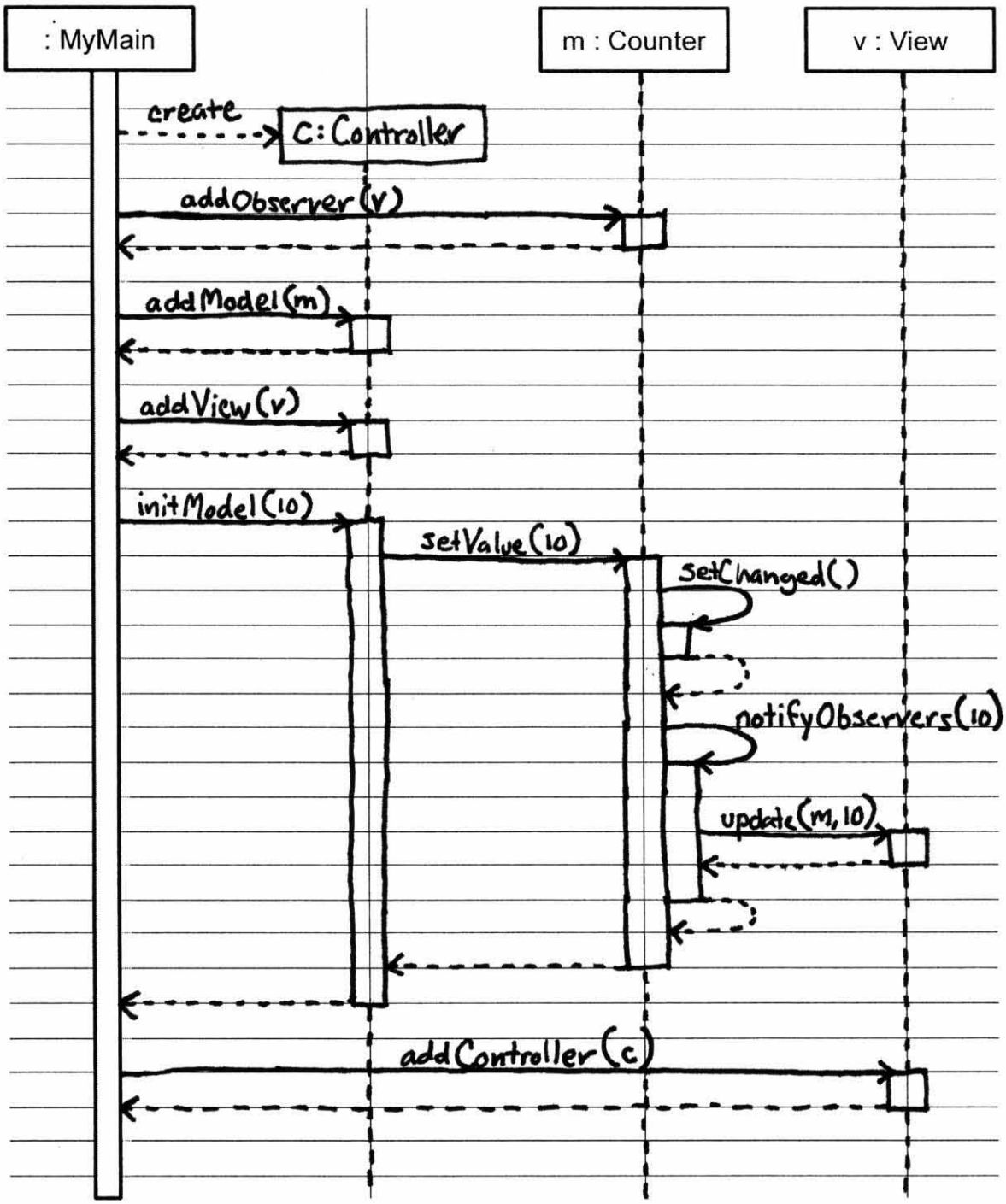
1. [15pts] Draw a package diagram representing the logical architecture of the SimpleMVC application.
   - Exclude all Java API packages (i.e., all packages that begin with "java.").
   - Do not explicitly model the "edu" and "edu.memphis" packages (because they just aren't interesting).
   - Model all dependencies.



2. [18pts] Draw a Design Class Diagram (DCD) that models the SimpleMVC application.
   - Model only the following classes: Counter, View, Controller, Observable, ActionListener, and Observer.
   - Don't forget to include all generalizations and associations among these classes.
   - Don't forget to include all attributes and operations of these classes, including types and visibilities.



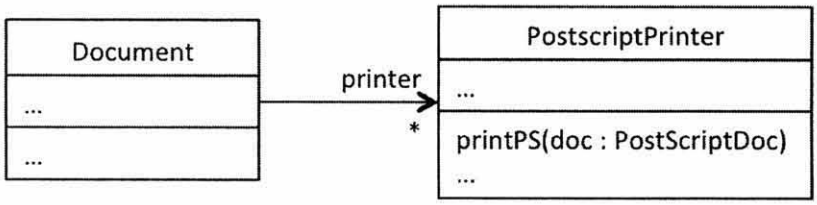
3. [18pts] Complete the sequence diagram below by modeling the object interaction that results from executing lines "Q3 START" through "Q3 END" in the `MyMain.main` method.
   - Model all objection creations, operation calls, and operation returns.
   - Include all operation arguments and execution specifications.



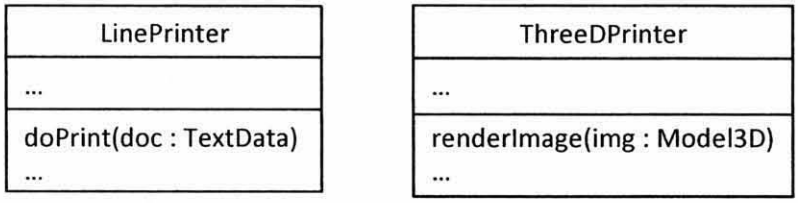
4. [3pts] Which of the following best characterizes the Law of Demeter?
   a. Thou shalt not kill.
   b. Don't talk to strangers.
   c. Build bridges, not walls.
   d. A bird in the hand is worth two in the bush.
   e. A chain is as strong as its weakest link.

5. [3pts] What is the expected relationship between coupling and cohesion?
   a. As coupling increases, cohesion decreases.
   b. As coupling increases, so does cohesion.
   c. Making the classes within a package more tightly coupled increases the packages cohesion.
   d. Only classes have cohesion, and only packages have coupling.
   e. None of the above. Coupling and cohesion are independent.

6. [15pts] Consider the following design for a document-editing system. The Document class represents a document, and Document objects know how to print themselves using a PostscriptPrinter object.



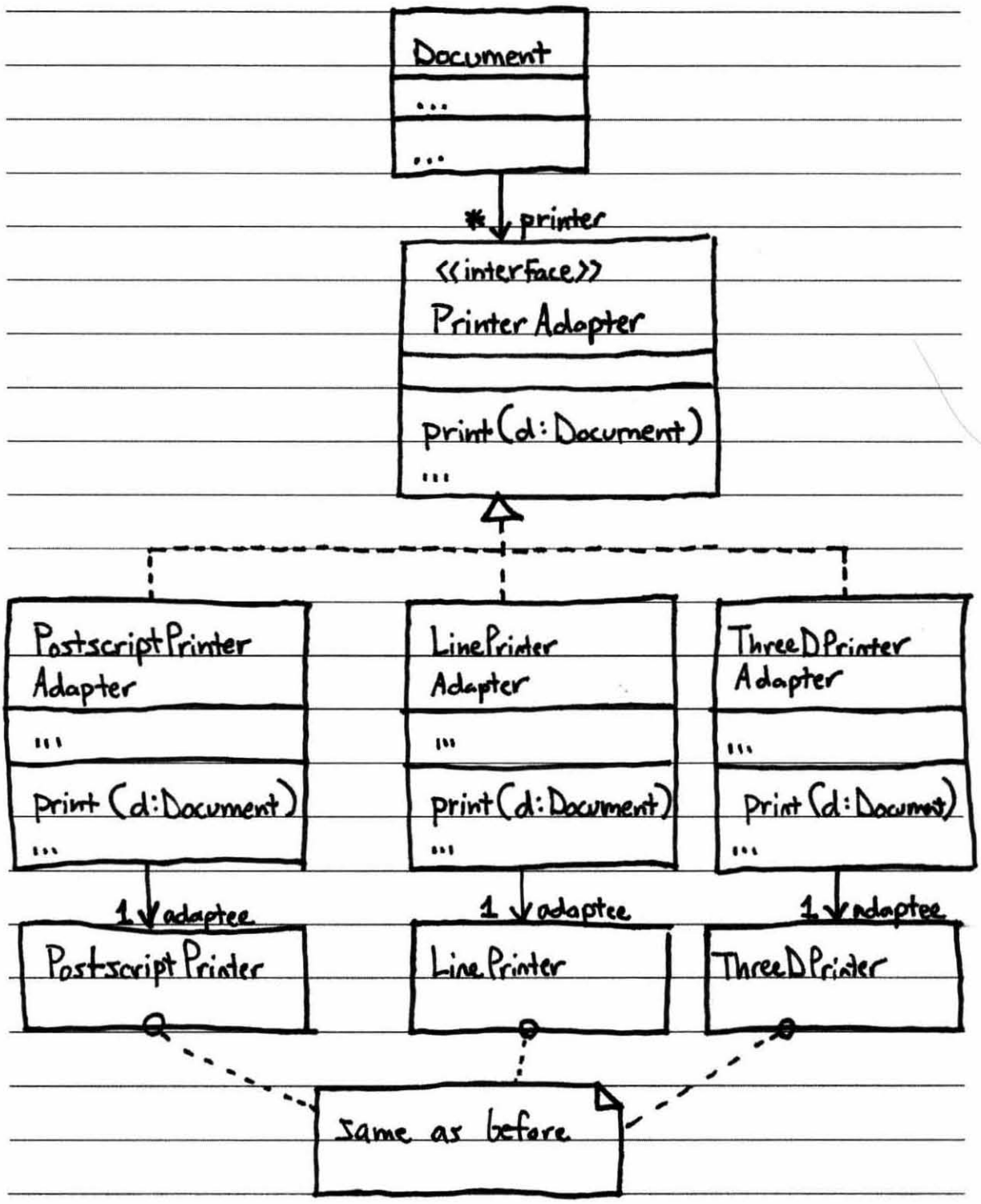
However, there are other types of printers that a document might want to print itself on, but these printers have slightly different interfaces than the Postscript printer, for example:



Using Larman's Polymorphism and Protected Variations patterns, refactor the design, so that the different types of printers can be easily swapped in and out.

Draw a design class diagram for your design on the next page.

Question 6 answer:



7. [3pts] If package **ui** contains package **web** contains package **jsp**, then what is the fully qualified name for **jsp** in UML syntax?

ui::web::jsp

8. [3pts] How are the Domain Model and the Domain Layer of a layered logical architecture related?
   a. They have exactly the same classes.
   b. Classes in the Domain Layer inspire classes in the Domain Model.
   (c.) Classes in the Domain Model inspire classes in the Domain Layer.
   d. Classes in the Domain Model become packages in the Domain Layer.
   e. They both contain only non-fabricated classes.

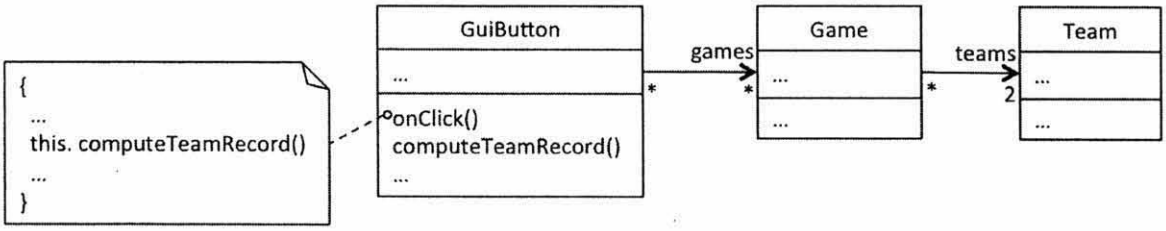
9. [3pts] According to the Creator Pattern, a class B should be responsible for creating instances of A if _____(fill in the blank)_____.
   a. B "contains" A
   b. B records A
   c. B closely uses A
   d. B has initializing data for A
   (e.) Any of the above.

10. [3pts] According to the Information Expert pattern, you should assign a knowing responsibility to a class that _____(fill in the blank)_____.
    a. is from the Domain Model
    b. is a fabricated class
    (c.) has the information necessary to fulfill the responsibility
    d. has polymorphic operations and implements a general interface
    e. Any of the above.

11. [3pts] System Operations defined in a System Sequence Diagram (SSD) largely form the interface of a _____(fill in the blank)_____.
    a. Master Switch
    (b.) Facade Controller
    c. Demultiplexer
    d. Creator Class
    e. None of the above.

12. [3pts] Which of the following is true of coupling?
    a. Less coupling is generally better.
    b. Java class C is coupled to class D if C invokes operations of D.
    c. Java class C is coupled to class D if C inherits from D.
    d. Coupling is a measure of how strongly one element is connected to others, has knowledge of others, and/or relies on others.
    (e.) All of the above.

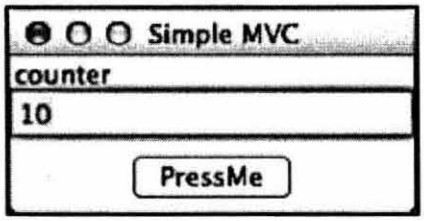
13. [10pts] Does the following design obey the Model-View Separation Principle? Explain your answer.



No. The View class, GuiButton, has a method computeTeamRecord(). Computing a teams win/loss record is domain concern. Thus, a view class contains model code, which violates the Model-View Separation Principle.

SimpleMVC Program

In this exam, we will use as a running example a small program that purports to use the MVC pattern. The UI for the program looks like this:



Initially, the counter is set to 10. The user can press the "PressMe" button to increment the counter by one.

Relevant Source Code

MyMain.java

```
package edu.memphis.simplemvc;

import edu.memphis.domain.Counter;
import edu.memphis.simplemvc.controller.Controller;
import edu.memphis.simplemvc.ui.View;

public class MyMain {

	private static int start_value = 10;

	public static void main(String[] args) {
		Counter myModel = new Counter();
		View myView = new View();
		Controller myController = new Controller(); ////// Q3 START

		myModel.addObserver(myView);
		myController.addModel(myModel);
		myController.addView(myView);
		myController.initModel(start_value);
		myView.addController(myController); ////////////// Q3 END
	}

}
```

Counter.java

```
package edu.memphis.domain;

import java.util.Observable;

public class Counter extends Observable {

	private int value = 0;

	public void setValue(int newValue) {
		this.value = newValue;
		setChanged();
		notifyObservers(newValue);
	}

	public void incrementValue() {
		++value;
		setChanged();
		notifyObservers(value);
	}

}
```

Controller.java

```
package edu.memphis.simplemvc.controller;

import java.awt.event.*;
import edu.memphis.domain.Counter;
import edu.memphis.simplemvc.ui.View;

public class Controller implements ActionListener {

	Counter model;
	View view;

	public void actionPerformed(ActionEvent e) {
		model.incrementValue();
	}

	public void addModel(Counter m) { this.model = m; }
	public void addView(View v) { this.view = v; }

	public void initModel(int x) { model.setValue(x); }

}
```

View.java

```
package edu.memphis.simplemvc.ui;

import java.awt.*;
import java.awt.event.*;
import java.lang.Integer;
import java.util.*;


public class View implements Observer {

      private TextField myTextField;
      private Button button;

      public View() {
            Frame frame = new Frame("Simple MVC");
            frame.add("North", new Label("counter"));

            myTextField = new TextField();
            frame.add("Center", myTextField);

            Panel panel = new Panel();
            button = new Button("PressMe");
            panel.add(button);
            frame.add("South", panel);

            ...
            frame.setSize(200, 100);
            frame.setLocation(100, 100);
            frame.setVisible(true);

      }

      public void update(Observable obs, Object obj) {
            myTextField.setText("" + ((Integer)obj).intValue());
      }

      public void addController(ActionListener controller) {
            button.addActionListener(controller);
      }

      public void setValue(int v) { myTextField.setText("" + v); }

      ...

}
```

Relevant Excerpts from the Java API

public class Observable

An observable object can have one or more observers. An observer may be any object that implements interface `Observer`. After an observable instance changes, an application calling the `Observable` object's `notifyObservers` method causes all of its observers to be notified of the change by a call to their `update` method.

- `public void addObserver(Observer o)`
  - Adds an observer to the set of observers for this object, provided that it is not the same as some observer already in the set.
- `protected void setChanged()`
  - Marks this `Observable` object as having been changed
- `public void notifyObservers(Object arg)`
  - If this object has changed, as indicated by the `hasChanged` method, then notify all of its observers, and then call the `clearChanged` method (not shown) to indicate that this object has no longer changed. Each observer has its `update` method called with two arguments: this observable object and the `arg` argument.

public interface Observer

A class can implement the `Observer` interface when it wants to be informed of changes in observable objects.

- `void update(Observable o, Object arg)`
  - This method is called whenever the observed object is changed. An application calls an `Observable` object's `notifyObservers` method to have all the object's observers notified of the change.

public interface ActionListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's `addActionListener` method. When the action event occurs, that object's `actionPerformed` method is invoked.

- `void actionPerformed(ActionEvent e)`
  - Invoked when an action occurs.