Name: ____________________________________

Quiz (Thu 18 Oct)

Create a system sequence diagram (SSD) for the following use case.

UC Checkout Bicycle

1. Mobile/Web Customer provides identification
2. System authenticates Mobile/Web Customer
3. System presents Mobile/Web Customer's available credits
4. Mobile/Web Customer sends checkout request with Station's ID
5. System identifies ID of bicycle in Station to checkout
6. System instructs Station to release bicycle with that ID
7. Station unlocks bicycle
8. Station sends confirmation to System that bicycle was unlocked
9. System debits Mobile/Web Customers account and records checkout
10. System tells Mobile/Web Customer which bicycle to collect and presents updated credit information

Name: ______________________________________

Quiz (Thu 25 Oct)

Create a domain model for the following problem[1]:

> An international airport requires a system to keep track of flight details for customers. Each flight has a flight number, destination airport, departure time, departure gate, airline, and flight cost. An airport has a code (i.e., airport code) and a country and city where it's located. Of course, an aircraft carries out the flight. An aircraft has a make, model, and capacity (number of passengers that it can carry).

[1] Excerpt adapted from http://titan.cs.unp.ac.za/~nelishiap/comp301/lectures/ood_exercises.pdf.

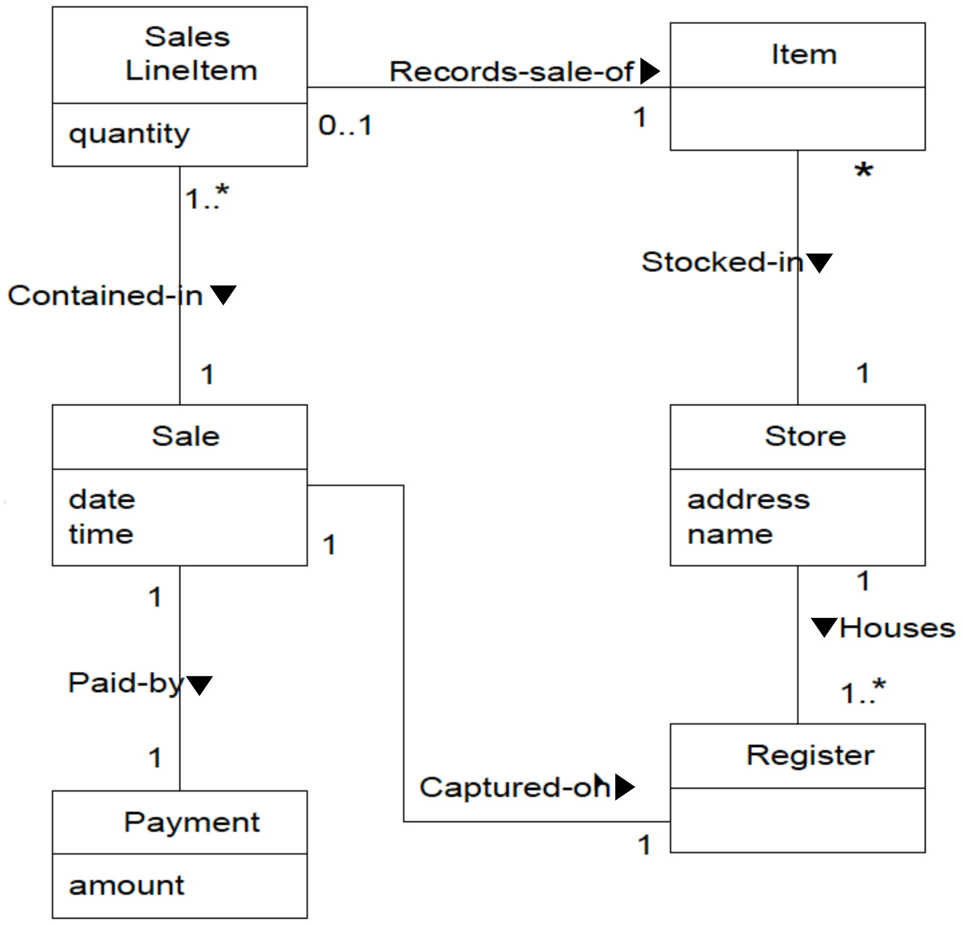


Figure 1. Example domain model from Larman.

Name: ________________________________

Quiz (Tue 30 Oct)

Create a domain model for the following problem[1]:

> **Old part:** An international airport requires a system to keep track of flight details for customers. Each flight has a flight number, destination airport, departure time, departure gate, airline, and flight cost. An airport has a code (i.e., airport code) and a country and city where it's located. Of course, an aircraft carries out the flight. An aircraft has a make, model, and capacity (number of passengers that it can carry).
>
> **New part:** Some flights are direct flights (i.e. they fly non-stop to the destination), and some are indirect flights (i.e., they fly via another airport to the destination). An indirect flight stops at an airport en route to its destination to refuel. Information regarding the refueling airport must also be stored. On some flights, additional passengers can board the plane at the transit airport. The system needs to keep track of whether boarding will take place at the transit airport or not.

Figure 1 provides an example domain model for a point-of-sale system.

---

[1] Excerpt adapted from http://titan.cs.unp.ac.za/~nelishiap/comp301/lectures/ood_exercises.pdf.

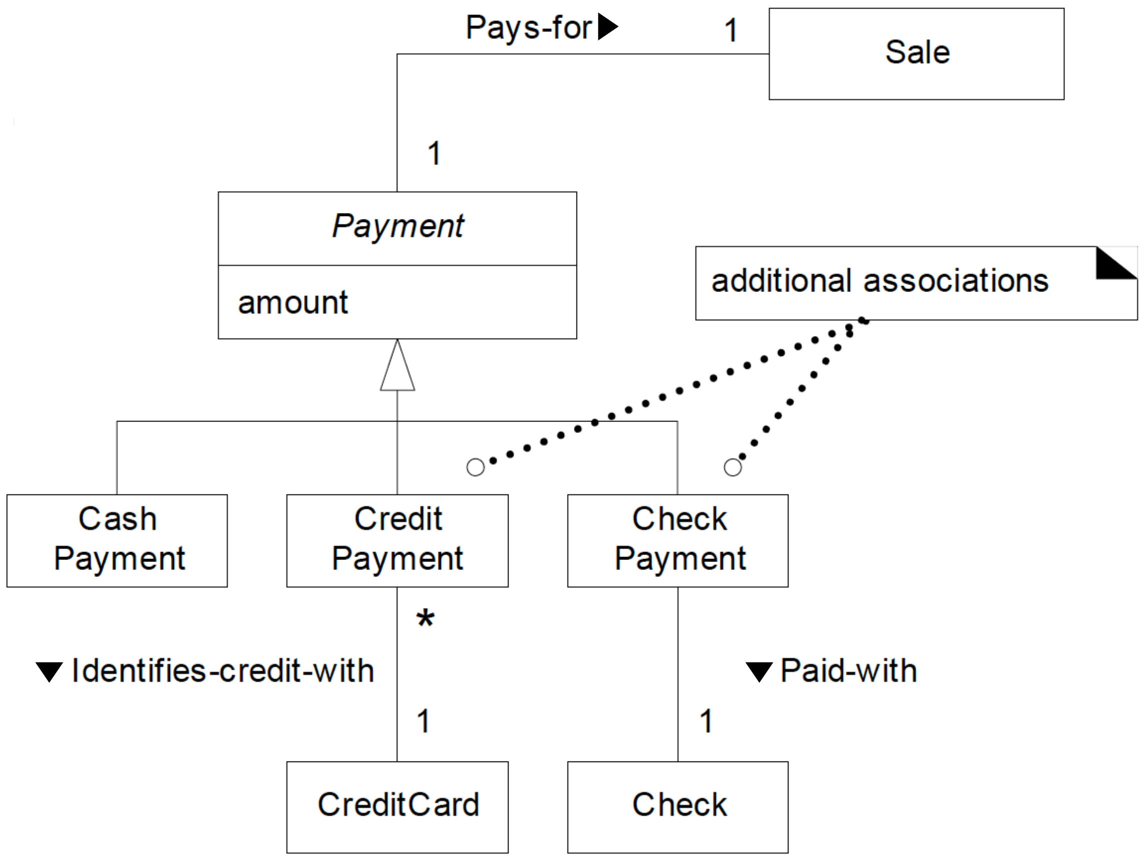


Figure 1. Example domain model from Larman.

Name: ____

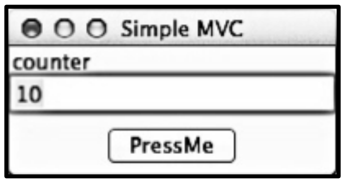
Quiz (Thu 1 Nov)

Draw a Design Class Diagram (DCD) that models the SimpleMVC application below.

- 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.

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();

            myModel.addObserver(myView);
            myController.addModel(myModel);
            myController.addView(myView);
            myController.initModel(start_value);
            myView.addController(myController);
      }
}
```

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.

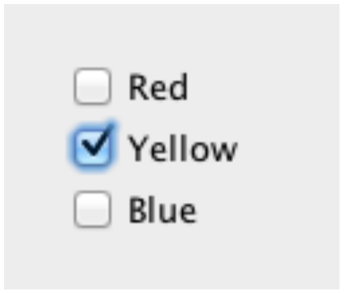
- `void actionPerformed(ActionEvent e)`
    - Invoked when an action occurs.

Name: ____

Quiz (Thu 8 Nov)

Radio buttons are a type of graphical user interface element that allows the user to choose one of a predefined set of options. They were named after the physical buttons used on older car radios to select preset stations. When you press one of the buttons, the other buttons pop out, leaving the one you pressed as the only button in the ”pushed in” position.

Here’s an example, which implements a set of radio buttons using checkboxes:



The Java code on the next page implements a set of radio buttons like those above.

Using the code, draw a sequence diagram representing the following interaction:

- Assume that the application is running, and that Yellow is currently selected.
- Show the interaction that occurs when the user clicks the Blue button.

Here are some additional tips/instructions:

- The interaction should begin with a call to `itemStateChanged` that comes “out of nowhere,” and end when the method returns.
    - Note that when a user clicks a button, the GUI subsystem first sets the button state to “selected,” and then calls `itemStateChanged` on any registered listeners.
- Model <u>all</u> method calls.
- Model the initial state of all checkboxes and all state changes to those checkboxes using state symbols (rounded rectangles).
- When the user clicks a checkbox, the GUI subsystem automatically sets the checkbox’s state to selected, and then executes the callback method (i.e., `itemStateChanged`).

```java
public class SquareRadioButtonsDemo extends JPanel
                                    implements ItemListener {
      JCheckBox redBox;
      JCheckBox yellowBox;
      JCheckBox blueBox;

      public SquareRadioButtonsDemo() {
            super(new BorderLayout());

            redBox = new JCheckBox("Red");
            redBox.setSelected(false);
            yellowBox = new JCheckBox("Yellow");
            yellowBox.setSelected(false);
            blueBox = new JCheckBox("Blue");
            blueBox.setSelected(false);

            //Register a listener for the check boxes.
            redBox.addItemListener(this);
            yellowBox.addItemListener(this);
            blueBox.addItemListener(this);

            //Put the check boxes in a column in a panel
            JPanel checkPanel = new JPanel(new GridLayout(0, 1));
            checkPanel.add(redBox);
            checkPanel.add(yellowBox);
            checkPanel.add(blueBox);

            add(checkPanel, BorderLayout.LINE_START);
            setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
      }

      /** Listens to the check boxes. */
      public void itemStateChanged(ItemEvent e) {
            Object source = e.getItemSelectable();

            if (source == redBox && redBox.isSelected()) {
                  yellowBox.setSelected(false);
                  blueBox.setSelected(false);
            } else if (source == yellowBox && yellowBox.isSelected()) {
                  redBox.setSelected(false);
                  blueBox.setSelected(false);
            } else if (source == blueBox && blueBox.isSelected()) {
                  redBox.setSelected(false);
                  yellowBox.setSelected(false);
            }
      }

      public static void main(String[] args) { … }
}
```

Name: ________________________________

Quiz (Tue 13 Nov)

Consider a simple shopping cart application that uses a CartContents class to keep track of the items in the shopping cart and an Order class for processing a purchase. The Order needs to determine the total value of the contents in the cart. Here's one possible implementation.

```
public class CartEntry {
    public float price;
    public int quantity;
}

public class CartContents {
    public CartEntry[] items;
}

public class Order {
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax) {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float orderTotal() {
        float cartTotal = 0;
        for (int i = 0; i < cart.items.length; i++) {
            cartTotal += cart.items[i].price * cart.items[i].quantity;
        }
        cartTotal += cartTotal*salesTax;
        return cartTotal;
    }
}
```

1. Why would these classes be considered tightly coupled?

2. What consequence would this coupling have if we were to try to change this logic to allow for discounts on individual items and/or total orders?

3. Propose an alternative design with looser coupling.

Name: ______________________________________

Quiz (Thu 15 Nov)

1. Cohesion is a qualitative indication of the degree to which a module
    a. can be written more compactly.
    b. focuses on just one thing.
    c. is able to complete its function in a timely manner.
    d. is connected to other modules and the outside world.

2. Coupling is a qualitative indication of the degree to which a module
    a. can be written more compactly.
    b. focuses on just one thing.
    c. is able to complete its function in a timely manner.
    d. is connected to other modules and the outside world.

3. Polymorphism reduces the effort required to extend an object system by
    a. coupling objects together more tightly.
    b. enabling a number of different operations to share the same name.
    c. making objects more dependent on one another.
    d. removing the barriers imposed by encapsulation.

4. Since modularity is an important design goal it is not possible to have too many modules in a proposed design. T OR F?

5. Information hiding makes program maintenance easier by hiding data and procedure from unaffected parts of the program. T OR F?

Name: ______________________________________

Quiz (Tue 27 Nov)

Read the following passage from Quinn on the Principle of Utility.

> **Principle of Utility:** An action is right (or wrong) to the extent that it increases (or decreases) the total happiness of the affected parties.
>
> *Utility* is the tendency of an object to produce happiness or prevent unhappiness for an individual or community. Depending on the circumstances, you may think of "happiness" as advantage, benefit, good, or pleasure, and "unhappiness" as disadvantage, cost, evil or pain.
>
> We can use the Principle of Utility as a yardstick to judge all actions in the moral realm. To evaluate the morality of an action, we must determine, for each affected person, the increase or decrease in that person's happiness, and then add up all of these values to reach a grand total. If the total is positive (meaning the total increase in happiness is greater than the total decrease in happiness), the action is moral; if the total is negative (meaning the total decrease in happiness is greater than the total increase in happiness), the action is immoral.

Now consider this scenario (also from Quinn).

> Over a period of seven years, about 500 residents of Freeport, Texas were overbilled for their water usage. Each resident paid on average about $170 too much, making the total amount of the overbillings about $100,000. The city council decided not to issue refunds, saying that about 300,000 bills would have had to have been examined, some residents had left town, and the individual refunds were not that large.

Did the city council make a moral decision with respect to the Principle of Utility? Justify your answer.

Name: ______________________________________

Quiz (Thu 29 Nov)

1. Sort unit, integration, and system testing in order from smallest subset of the system under test to largest.

2. Which of the following is generally a difference between unit and integration testing.
    a. Unit tests are repeatable/deterministic; integration tests may not be so.
    b. Unit tests are “in-memory”; integration tests may not be so.
    c. Unit tests are “fast”; integration tests may not be so.
    d. A unit test tests only one property; an integration test may test multiple properties.
    e. All of the above.

3. What is the difference between blackbox and whitebox testing?

4. Given the following method, how might you select blackbox tests for it? Give examples.

```
public int myMethod(int i, String s)
```

5. In test-driven development, when do you write each test (relative to the code it tests)?

Name: ________________________________

Quiz (Tue 4 Dec)

Consider the `myF()` method.

1. Draw a control flow graph for `myF()`.
2. Define test suites for `myF()` that provide:
    a. Statement coverage
    b. Condition coverage
    c. Path coverage (1-loop or less paths only)

```
int myF(int x, int y) {
  while (x > 10) {
    x = x - 10;
    if (x == 10) {
      break;
    }
  }
  if (y < 20 && x%2 == 0) {
    y = y + 20;
  } else {
    y = y - 20;
  }
  return 2*x + y;
}
```