CS130: Software Engineering (Fall 2019)

Assignment 1

Due Date: October 23, 11:59 PM
Late Policy: There is a 10% penalty per every late day. Late assignments are not accepted after 48 hours. Once a solution is released, there's no credit.

Submission Instruction: Please submit **a single typed-out** report file (.pdf, .doc, .docx) to CCLE before the deadline.

Q1: Probing Patterns [10pts, 5*2]

Identify the name of the design pattern from the given descriptions:

1. In many software repositories, it is recommended to use Camel case with JavaScript but Snake case with Python for defining classes or variables. In applications with a JavaScript front-end making requests to a Python backend, there can be incompatibility of class or variable names. This design pattern can be used for ensuring compatibility by converting JavaScript interfaces to those expected by the Python backend. (***Answer:*** *Adapter design pattern)*

2. This pattern defines a set of algorithms that can be used interchangeably. It also captures the abstraction in an interface, buries implementation details in derived classes. (***Answer:*** *Strategy design pattern)*

3. You're building the server-side REST API for a social sharing application in which a user can write some text and share it on many social media sites simultaneously. You're the backend engineer and you want a single endpoint '/share' where a client would send some text (via a POST request). The backend handles the logic for sharing that text on Facebook, Twitter, Linkedin etc. Which design pattern is used for '/share'? (***Answer:*** *Facade design pattern)*

4. For certain tasks such as Logging, there is a requirement of having only one instance of a resource (eg. Logger). This design pattern ensures that a class has only one instance and provides a globalpoint of access to that instance. (***Answer:*** *Singleton design pattern)*

5. This design pattern defines an object that controls how a set of objects interact. Also, loose coupling between colleague objects is achieved by having colleagues communicate with this pattern, rather than with each other. (***Answer:*** *Mediator design pattern*)

Q2: Diagramming DoorDart [15pts]

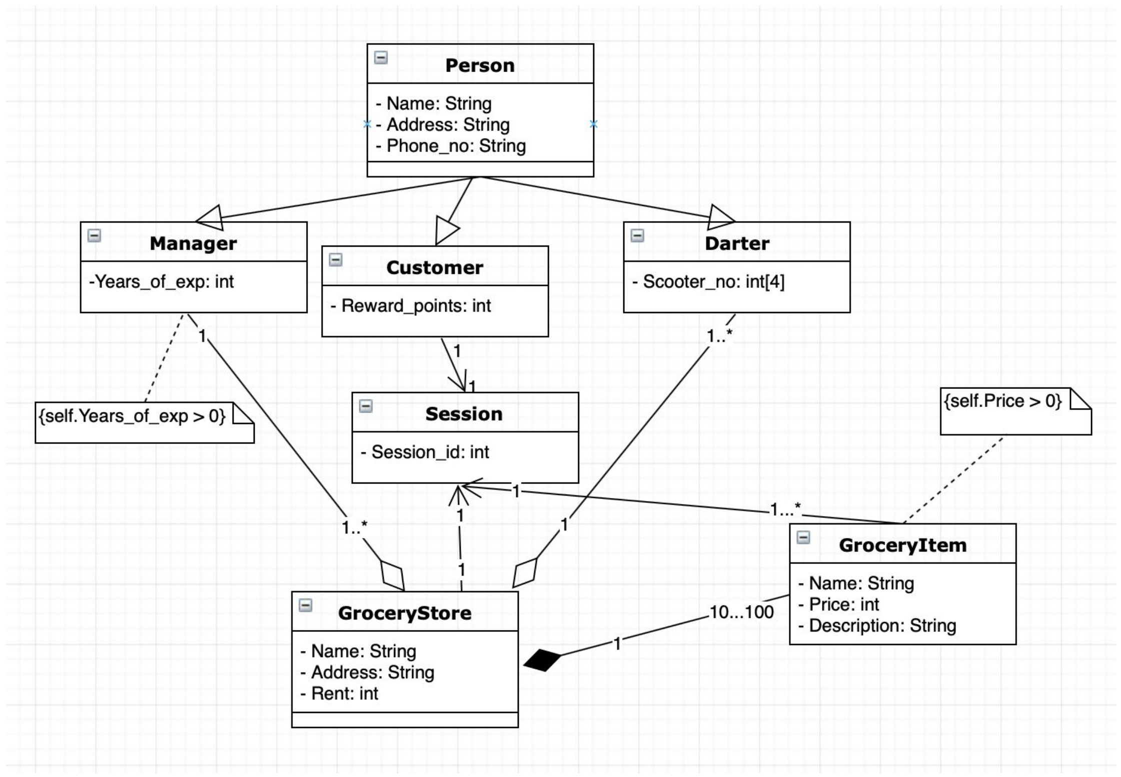
You were hired as a software architect at DoorDart, a new startup that delivers groceries to people's doors! DoorDart's code-base is very poorly organized and you want to rewrite a lot of things. So in this problem, you will design a UML Class Diagram for communicating the design of your proposed system with other engineers.

Here's what you were told:

- DoorDart has on-boarded 5 Grocery stores and plans to onboard more
- Each grocery store has a name, an address, a store manager, and at least one "Darter" (people who deliver the ordered groceries)
- Each grocery store has 10 - 100 grocery items.
- Grocery items have a name, price and description
- Every store manager has certain years of experience. Store managers can manage multiple grocery stores.
- Customers can start a Session
- Every Session has a store name, some grocery items, and a total price
- Customers, Darters and Managers are people so they have names, phone numbers and addresses.

You might've been given incomplete information. So you should make educated guesses and add 2 more constraints in your UML class diagram. Write 2-3 sentences explaining the additional things you added. There can be multiple correct answers.

Answer:



Grading: Partial credit guidelines:

- 7 classes: 7*1 = 7pts (1 point per correct implementation of class)
- Any 7 correct relationships (arrows): 7*1 = 7pts
- 1 point for additional constraints that make sense, eg:
    - Positive value constraint
    - Session-darter relationship
    - Params for Customers (like rewards)
    - Coupons etc.

There's no single mapping from requirements in English to UML diagram, so we won't deduct the points unless the answer or the relationship is obviously inconsistent with the requirement.

Also, it's okay if they use simple association instead of aggregation & composition

Q3: Let's Log [15pts]

Multithreading is usually implemented in a system to enhance its throughput. Sometimes multiple threads may request accesses to a single "logger" resource. Therefore, when we are designing a class for the access control, we wish to do it in the following way:

- First, we will check whether the resource instance exists. If there does not exist a resource instance, then the program should run into asynchronized block.
- Once it goes into the synchronized block, it should check if it is still null.
- If the resource is null, we create such an instance.

This solution ensures that only the first few threads that try to acquire your object have to go through the process of acquiring its lock.

Please use **Java** code to implement such a Logger following the logic above.

Answer:

```
public class Logger {

        private volatile static Logger uniqueLogger; // 4pts, 2 for "static", 2 for
"volatile"
        private Logger(){ // 2pt for a "private" constructor
                System.out.println("Initializing an Logger instance");
              }
        public static Logger getInstance(){ // 2pt for static
              if(uniqueLogger == null){ // 1pt for checking if uniqueLogger is null
                synchronized(Logger.class){    // 3pts for using a "synchronized" block.
                  if(uniqueLogger == null){ // 2pt for double check
                       lockinguniqleLogger = new Logger(); // 1pt for new a Logger
                       System.out.println("First time getInstance was invoked");
                     }
              }
       }
      System.out.println("Returning the Logger instance"); return uniqueLogger;
    }

 }
```

Q4: Judging JHotDraw [22 pts]

This program comprehension question intends to provide an experience for you to examine the use of design patterns in a real world application. Please download the latest version of JHotDraw from this sourceforge link (Version: JHotDraw 7.6). If the following link does not work, visit this URL: `https://sourceforge.net/projects/jhotdraw/files/JHotDraw/JHotDraw%207.6/jhotdraw-7.6.nested.zip/download`

a. In package org.jhotdraw.color, please investigate the interface ColorSliderModel. This interface is implementing a strategy design pattern.

   (1) What is the name of concrete strategy classes that are implementing ColorSliderModel interface? Please list all the names of classes that are generalizations of ColorSliderModel. [Hint: please also look at org.jhotdraw.gui]
   Answer:
   AbstractColorSliderModel implements ColorSliderModel,
   DefaultColorSliderModel extends AbstractColorSliderModel

PaletteColorSliderModel extends DefaultColorSliderModel ***(3 X 1 pt)***

(2) ColorSliderModel also uses Observer pattern. Look at the following code (In org.jhotdraw.color.ColorWheelChooser, line 51) and the class diagram for Observer pattern in the Lecture 3 slides. Please indicate which method in this code correspond to the "registerObserver()" method in lecture slides. Please also indicate which method corresponds to "update()" method in lecture slides.

```
ccModel.addChangeListener(new ChangeListener() {
    @Override
    public void stateChanged(ChangeEvent evt) {
        setColorToModel(ccModel.getColor());
    }
});
```

Answer:
addChangeListener() correspond to registerObserver() **(3pts)**
stateChanged correspond to update() **(3pts)**

b. In package org.jhotdraw.draw.io, the class ImageInputFormat is in charge of loading an image from a file and adding the figure to drawing. A function read() is implemented as follows:

```
public void read(File file, Drawing drawing, boolean replace)
throws IOException {
        ImageHolderFigure figure = (ImageHolderFigure) prototype.clone();
        figure.loadImage(file);
        figure.setBounds(
                new Point2D.Double(0, 0),
                new Point2D.Double(
                figure.getBufferedImage().getWidth(),
                figure.getBufferedImage().getHeight()));
        if (replace) {
            drawing.removeAllChildren();
            drawing.set(CANVAS_WIDTH, figure.getBounds().width);
            drawing.set(CANVAS_HEIGHT, figure.getBounds().height);
        }
        drawing.basicAdd(figure);
    }
```

Please identify all statements from the above code blocks that demonstrate the use of a drawing strategy (the use of interface org.jhotdraw.draw.Drawing)

Answer:
All statements that use drawing strategy (4pts, 1pt for each)
drawing.removeAllChildren();
drawing.set(CANVAS_WIDTH, figure.getBounds().width);
drawing.set(CANVAS_HEIGHT, figure.getBounds().height);
drawing.basicAdd(figure); **(4pts)**

c. Please discuss the benefit of using the strategy design pattern from the perspective of Information Hiding principle. **(3pts)**
Answer:
ImageInputFormat's code of method "read()" does not need to change, when the implementation of drawing.set,drawing.basicAdd, drawing.removeAllChildren changes. (Partial Breakdown: 1pts for read() does unchanged, 2pts for other 3 methods change, 3pts in total.)

d. In package org.jhotdraw.draw, the DrawingEditor class in the package acts as a *mediator* for objects implementing interface DrawingView. A mediator is a class that controls the status of different components and help objects decouple from each other.
(1) What concrete class implements DrawingView? **(2pts)**
Answer: `DefaultDrawingView in org.jhotdraw.draw`

(2) Please list the method in class DrawingEditor to register a new DrawingView object **(2pts)**
Answer: `add()`

(3) Please list the method to deregister an existing DrawingView object **(2pts)**
Answer: `remove()`

Q5. Sequence Sketching [15 points]

The Frobenius norm is the square root of the sum of squares of each element in a matrix. It is defined as:

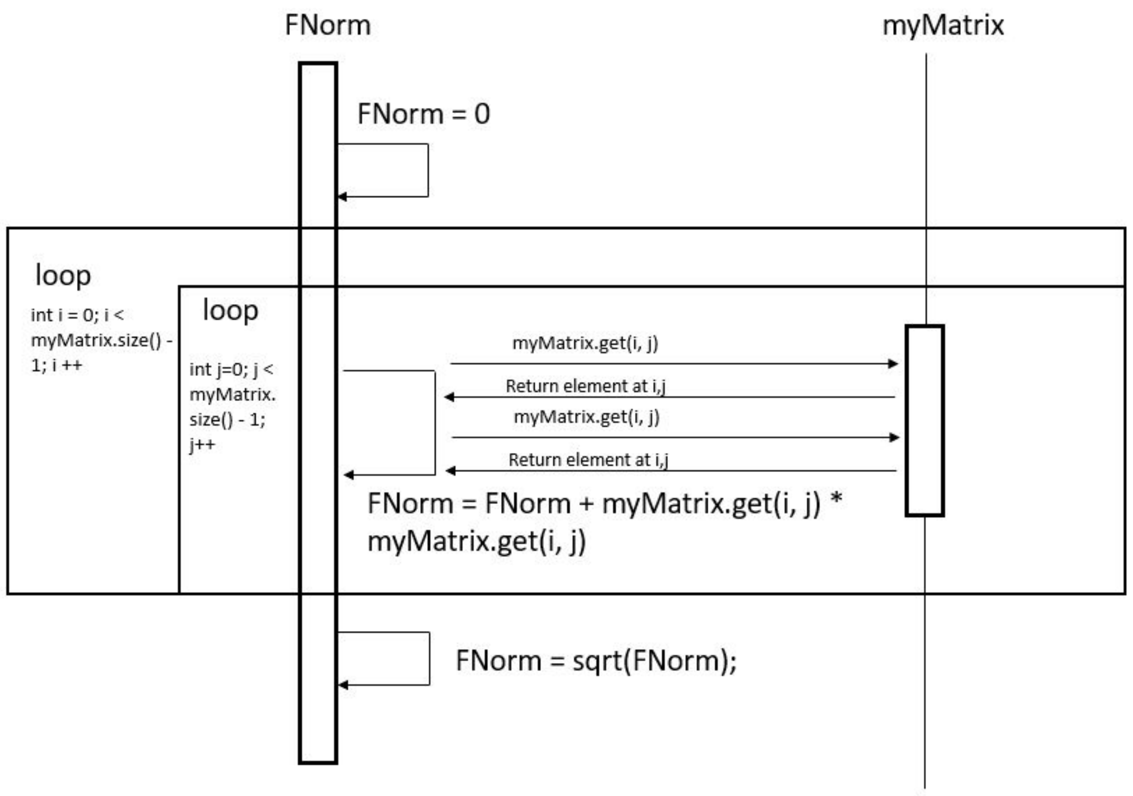
$$||\mathrm{A}||_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{i\,j}|^2}$$

Please draw a sequence diagram for the following code blocks calculating the Frobenius norm of a square matrix of size *n x n*:

```
public class Matrix{
    private double[][] _matrix = new double[n][n];
    ...
    public double get(int i, int j){ return _matrix[i][j];}
    public double size(){ return _matrix[0].length;}  // returns 'n'
    ...
}

public class FNorm{
      ...
      public double compute(Matrix myMatrix){
             double FNorm = 0;
             for(int i = 0; i < myMatrix.size() - 1; i ++) {
                    for(int j=0; j < myMatrix.size() - 1; j++) {
                    FNorm = FNorm + myMatrix.get(i, j) * myMatrix.get(i, j);
                    }
             }
             FNorm = sqrt(FNorm);
             return FNorm;
      }
      ...
}
```

Answer:


FNorm
myMatrix
FNorm = 0
loop
int i = 0; i < myMatrix.size() - 1; i ++
loop
int j=0; j < myMatrix. size() - 1; j++
myMatrix.get(i, j)
Return element at i,j
myMatrix.get(i, j)
Return element at i,j
FNorm = FNorm + myMatrix.get(i, j) * myMatrix.get(i, j)
FNorm = sqrt(FNorm);

Partial Grade guidelines:

- 5 points for recognizing FNorm and my Matrix as the two “actors” in the sequence(3 for showing the two actors, 2 for showing when they are active i.e. using a block vs a line).
- 5 points for depicting the loops (Can be done using one rectangle for both loops or as above, or anything that shows that some operations are occurring inside the loop).
- 5 points for correct ordering of statements and either labelling each step with the code statement or an English description of what happens at each step.

Q6. Problematic Python Piece (15pts)

One fine day, I was making a program to compute the distances between two vectors. My plan was to implement the following metrics:

1. Euclidean distance
2. Manhattan distance
3. Cosine similarity
4. *Maybe more later…*

The hasty person I am, I wrote the following code:

```
class Vector:
    def __init__(self, name, value):
        self.name = name
        self.value = value
        self.length = len(value)

class DistanceFinder:
    def square_rooted(x):
        return round(sqrt(sum([a*a for a in x])),3) # L2 norm of vector

    def find_distance(self, v1, v2, type):
        if type == 'Euclidean':
           return sqrt(sum(pow(a-b,2) for a, b in zip(v1.value, v2.value)))
        elif type == 'Manhattan':
            return sum(abs(a-b) for a,b in zip(v1.value,v2.value))
        elif type == 'Cosine':
            numerator = sum(a*b for a,b in zip(v1.value,v2.value))
            denominator = square_rooted(v1.value)*self.square_rooted(v2.value)
            return round(numerator/float(denominator),3)
        else:
            raise ValueError(type) # Throws error for unknown types
# Example:
# v1 = Vector('x', [5, 5])
# v2 = Vector('y', [2, 2])
# df = DistanceFinder()
# print(df.find_distance(v1, v2, 'Euclidean'))          Output: 4.24264068712
```

a) Briefly list drawbacks of the above style of coding (*Hint: Think about possible modifications which would make it necessary to change DistanceFinder*) **(3pts)**
b) Rewrite the above code with a suitable design pattern **(8pts)**
a) Which pattern did you use and how were the drawbacks listed in part a) addressed by your solution to part b)? **(4pts)**

Answer:

A. *The drawbacks of the above coding style are (Any 2 for full points):*

   a. ***Hard to Maintain:*** *class will have to change if we add more distance functions*
   b. ***Hard to Read:*** *too much logic within one method makes it hard to follow*
   c. ***Hard to Modify:*** *class will have to change if the definition of a distance function needs to be modified*
   d. ***Tightly coupled with class Vector:*** *If the class Vector is modified so that the field storing the numbers is not called 'value' anymore, the class DistanceFinder will have to be changed too!*

B.

```
class Vector:
    def __init__(self, name, value):
      self.name = name
      self.value = value
      self.length = length

class DistanceFinder:
      # 4 points for Factory Method
      def find_distance(self, v1, v2, type):
            finder_function = get_distance_finder_function(type)
            return finder_function(v1, v2)

def get_distance_finder_function(type):
      # 2 points for abstracting out logic
      if type == 'euclidean' or type == 'Euclidean':
         return euclidean
      elif type == 'manhattan' or type == 'Manhattan':
         return manhattan
      elif type == 'cosine' or type == 'cosine':
         return cosine
      else:
        Raise ValueError(type)


def square_rooted(x):
      return round(sqrt(sum([a*a for a in x])),3) # L2 norm of vector
```

```
# 2 points for separate functions for all distances
def euclidean(v1, v2):
      return sqrt(sum(pow(a-b,2) for a, b in zip(v1.value, v2.value)))

def manhattan(v1, v2):
      return sum(abs(a-b) for a,b in zip(v1.value,v2.value))

def cosine(v1, v2):
    numerator = sum(a*b for a,b in zip(v1.value,v2.value))
    denominator = square_rooted(v1.value)*square_rooted(v2.value)
    return round(numerator/float(denominator),3)
```

C. Factory Method Design Pattern.

Now, with these changes, **the DistanceFinder class will not have to be changed** in the following scenarios:

- **Addition of a new distance function**
- **Modification of a particular distance function**
- **Changes in the class Vector**

Furthermore, each individual distance function can be used now in other parts of the code using function calls thereby **decreasing code duplication. (2pts X 2 explanations)**

Note: The above problems can also be solved using the Strategy pattern which is also correct.

Q7. Knowing KWIC [8pts]

KWIC is a little program covered in the class. Its function is to output all circular shifts of a given sentence in an alphabetical order. For example, give ***"take software engineering class"*** as an input, it returns four circular shifted version sentences which are ordered alphabetically as:

- "**c**lass take software engineer"
- "**e**ngineering class take software"
- "**s**oftware engineering class take"
- "**t**ake software engineering class"

Now, Joe has a design of implementation of this program as follows:

```
public class Input {
      private static char[] chars_;
      private static int[] line_index_;
      public static char[] getChars() {return chars_;}
      public static int[] getLineIndex() { return line_index_;}

      public static void execute(String file) {
            ... // Loading input from file
      }
}

public class CircularShift {
      private static int[][] circular_shifts_;
      public static int[][] getCircularShifts() { return circular_shifts_; }

      public static void execute(){
            int[] line_index = Input.getLineIndex();
            char[] chars = Input.getChars();
            ... // Execute circular shift
      }
}

public class Alphabetizing {
      private static int[][] alphabetized_;
     public static int[][] getAlphabetized() { return alphabetized_; }

      public static void execute(){
            int[][] circular_shifts = CircularShift.getCircularShifts();
            int[] line_index = Input.getLineIndex();
            char[] chars = Input.getChars();
            ... // Alphabetizing job
      }
}

public class Output {
      public static void execute(){
            int[][] alphabetized = Alphabetizing.getAlphabetized();
            int[] line_index = Input.getLineIndex();
            char[] chars = Input.getChars();
            ... // Output files
      }
}

public class MasterControl {
      public static void main(String[] args){
        if(args.length != 1){
            System.err.println("KWIC Usage: java kwic.ms.MasterControl
file_name");
            System.exit(1);
        }
        Input.execute(args[0]);
        CircularShift.execute();
        Output.execute();
      }
}
```

As you can see, there are two I/O components (Input class and Output class), a CircularShift class, an Alphabetizing class, and a MasterControl class.
Please read the skeleton code and answer the following questions.

Please indicate whether the following statement is True of False.. **(2pts * 4 = 8pts)**

I. Having a public getter method of `public static int[][] getCircularShift` and keeping the field `int[][] Circularshift` as private is directly following the information hiding principle
II. Using a static call invocation in MasterControl class `Input.execute(args[0])` is a violation of the information hiding principle
III. If `int[] line_index` in Input is changed to `Integer[]`, then CircularShift, Alphabetizing, and Output should be modified.

IV. If the internal data layout in CircularShift is changed, then `Alphabetizing.execute()` should also be changed is a violation of Information Hiding Principle.

Answer: False, False, True, True