CS 3321 or INFO 3307/5307
Homework 2 – Design Principles Solutions

Solution Key

Assigned: September 16, 2019
Due: October 04, 2019 @ 2300

Grading: * Each of the following questions has either one or two principle violations. The grading for this assignment works as follows: - In the case of one violation: * 2.5 Points for identifying the violation * 2.5 Points for the reasoned argument alluding to the same cause I have identified * 5 Points for either a description of the fix (which should result in something similar to mine) or for similar code as mine (it must solve the fix, and I probably went overboard). - In the case of two violations: * 1.5 Points for identifying each violation * 1 Point each for the reasoned argument alluding to the same cause I have identified (if they get it backwards, - 1 point total) * 5 points for either a description of the fix (which should result in something similar to mine) or for similar code as mine (it must solve the fix, and I probably went overboard).

Questions

In lecture, we discussed a number of software principles, including Encapsulate What Varies, DRY, LSP, OCP, ISP, and SRP. Identify in each of the following cases which if any of these principles are violated and if so, (a) justify your reasoning as to why that principle applies in a sentence or two, and (b) outline how the code can be corrected (words is fine for the latter if it is clear, or you can give revised code).

Note, there may be more than one principle violated. In the case of more than one, explain each principle; but, don't identify a design principle without very good cause.

1. (10 Points) This class is a snippet from a console-based 2D board game implementation (think e.g. Chess or Checkers)

```
class Game {
    private Piece[][] board;
    public void move() {
        /* Read user input */
        Point source = readCoordinateFromInput();
        Point dest = readCoordinateFromInput();
        /* Does the source square have a piece? Is the desination empty? */
        if (board[source.x][source.y] == null || board[dest.x][dest.y] != null)
            throw new IllegalMoveException();
```

```
        /* ... More rules here .. */
        /* Print out the current state of the board */
        printBoard();
    }

    private Point readCoordinateFromInput() {
        /* Read input from the terminal and convert to a point */
    }
    private void printBoard() {
        /* Printing code here */
    }
}
```

Violations:

- Single Responsibility Principle - The game class has been relegated to managing the data underlying the board, handling the logic of a players move and finally coordinating this with the print update. I argue the Game class is responsible for coordinating all of this behavior and this is it. Everything else should be moved into classes that contain the logic.
- OCP/Encapsulate what Varies - There are several rules which evaluate the viability of the move, these should be extracted out and encapsulated. Furthermore, rules should be moved into a list or array witch can be modified at runtime and evaluation code replaced with a loop which processes them.

Fix:

```
class Board {
    private Piece[][] board;

    private void printBoard() {
        /* Printing code here */
    }

    Piece get(Point loc) { return board[loc.x][loc.y]; }
}

interface GameRule {
    boolean evaluate(Board b, Point s, Point d);
}

class ValidMove implements GameRule {
    boolean evaluate(Board b, Point s, Point d) {
      return isEmpty(b, s) || !isEmpty(b, d);
    }

    private boolean isEmpty(Board b, Point p) {
        return board[p.x][p.y] == null
    }
}

class MoveHandler {
    private GameRules rules[];
```

```
    void makeMove() {
        /* Read user input */
        Point source = readCoordinateFromInput();
        Point dest = readCoordinateFromInput();
        /* Test game rules */
        for (rule r : rules)
            if (r.evaluate(b, s, d)) throw new IllegalMoveException();
    }

    private Point readCoordinateFromInput() {
      /* Read input from the terminal and convert to a point */
    }
}

class Game {
    private Board board;
    private MoveHandler handler;

    public void move() {
        /* Handle the move */
        hanlder.makeMove();
        /* Update the board */
        board.printBoard();
    }

}
```

2. (10 Points) Here is another board game example, this time just code for the board and there are both 2D and 3D boards.

```
public class Square  {
    int value;
}
public class Board {

    private Square[][] squares;
    public void updateSquare(int x, int y, Square aSquare) {
        squares[x][y] = aSquare;
    }

    public Square getSquare(int x, int y) {
        return squares[x][y];
    }

    // ... more methods
}

public class ThreeDBoard extends Board {

    private Square[][][] squares;
    public void updateSquare(int x, int y, int z, Square aSquare) {
        squares[x][y][z] = aSquare;
```

```
    }

    public Square getSquare(int x, int y, int z) {
        return squares[x][y][z];
    }

    // ... more methods
}
```

Violation:

- Liskov Substitution Principle: A 3D Board overloads the updateSquare() and getSquare() methods by adding the extra z component, but the original methods are still there. Unfortunately, they do nothing and are thus not applicable to the 3D board. Meaning that technically anywhere that I have a reference to a Board type, I cannot use a ThreeDBoard class, as currently defined.

Fix:

```
public class Square  {
    int value;
}

public interface Board {
  void updateSquare(int[] location, Square aSquare);

  Square getSquare(int[] location);
}

public class TwoDBoard implements Board {

    private Square[][] squares;
    public void updateSquare(int[] location, Square aSquare) {
        squares[location[0]][location[1]] = aSquare;
    }

    public Square getSquare(int[] location) {
        return squares[location[0]][location[1]];
    }

    // ... more methods
}

public class ThreeDBoard implements Board {

    private Square[][][] squares;
    public void updateSquare(int[] location, Square aSquare) {
        squares[location[0]][location[1]][location[2]] = aSquare;
    }

    public Square getSquare(int x, int y, int z) {
        return squares[location[0]][location[1]][location[2]];
    }
```

```
    // ... more methods
}
```

3. (10 Points) This code defines an API for interacting with a batch computing cluster

```
public interface ClusterManager {
    public int getJobCount();
    public List<JobId> getActiveJobIds();
    public JobId startJob(Job job);
    public double getJobProgress(JobId id);
    public void waitForJobToFinish(JobId id);
    public void pauseJob(JobId id);
    public void resumeJob(JobId id);
    public List<MachineId> getMachineIds();
    public MachineId getMachineIdForJob(JobId id);
    public List<JobId> getJobsOnMachine(MachineId id);
    public double getClusterLoadPercentage();
    public DiagnosticReport runClusterDiagnostics();
    public void restartCluster();
}
```

Violation:

- Interface Segregation Principle: If this interface were to be implemented it would lead to a class with far too many responsibilities, or where each class would implement only the methods necessary and leave a large number with empty method bodies. This may also lead to a later violation of the Single Responsibility Principle.

Fix: Something similar to the following (either 3 or 4 interfaces) I could argue for the first two being composed together.

```
public interface ClusterJobManager {
    public int getJobCount();
    public List<JobId> getActiveJobIds();
}

public interface Job {
    public JobId startJob(Job job);
    public double getJobProgress(JobId id);
    public void waitForJobToFinish(JobId id);
    public void pauseJob(JobId id);
    public void resumeJob(JobId id);
}

public interface ClusterMachineManager {
    public List<MachineId> getMachineIds();
    public MachineId getMachineIdForJob(JobId id);
    public List<JobId> getJobsOnMachine(MachineId id);
}

public interface ClusterManager {
    public double getClusterLoadPercentage();
```

```
    public DiagnosticReport runClusterDiagnostics();
    public void restartCluster();
}
```

4. (10 Points) This is a code fragment for a multi-sensor HVAC (heating/air conditioning) system display.

```
class HVACController {
    // ...
    void updateDisplay () {
        double tenInsideTemps = 0;
        double tenOutsideTemps = 0.0;
        // implemented in/out temp so far,
        // still have in/out humidity and 2nd floor temps to go
        for(int i=0; i<10; i++){
            tenInsideTemps = tenInsideTemps + insideSensor.getTemp();
            tenOutsideTemps = tenOutsideTemps + outsideSensor.getTemp();
            TimeUnit.SECONDS.sleep(1); // ten readings to average
        }
        double aveInsideTemp = tenInsideTemps / 10;
        double aveOutsideTemp = tenOutsideTemps / 10;
        display.setInsideTemp(aveInsideTemp);
        display.setOutsideTemp(aveOutsideTemp);
    }
}
```

Violations:

- Encapsulate What Varies. The code for updating the display is similar for each sensor, but also dependent upon the sensor itself. Since there can be many sensors, it would be wise to initially place these into an array or list which would facilitate handling them together. We would then want to extract out the specific display.set*() calls which are specific to each of the individual sensors to the sensor class. The fix for this is to have a specific updateDisplay for each sensor.
- Don't Repeat Yourself. The code to calculate the averages are the same for each sensor. Thus, we should probably construct an interface for the sensors which allows this to be processed. That is, we can have a reset which resets the average for the sensor, a collect which collects a temperature from the sensor, and a calcAverage which calculates the average of the sensor for a given number of readings. We also probably want to modify HVACController.updateDisplay to include the number of readings and how long to sleep, rather than using magic numbers.

fix:

```
interface Sensor() {
    void updateDisplay(Display disp, int readings);
    float getTemp();
    void resetAverage();
    void collect();
}

abstract class AbstractSensor implements Sensor {
    double average = 0.0d;
```

```
    float temp;

    void resetAverage() { average = 0.0d; }
    void collect() { average += getTemp(); }
    void getTemp() { return temp; }
}

class InsideSensor extends AbstractSensor {
    void updateDisplay(Display disp, int readings) {
        disp.setInsideTemp(average / readings);
    }
}

class OutsideSensor extends AbstractSensor {
    void updateDisplay(Display disp, int readings) {
        disp.setOutsideTemp(average / readings);
    }
}

// other sensor classes

class HVACController {
    // ... Sensor[] sensors set by constructor or addSensor method
    void updateDisplay (int readings, int sleep) {
        for (int j = 0; j < sensors.length; j++)
            sensors[j].resetAverage();

        for(int i = 0; i < sensors.length; i++){
            for (int j = 0; j < readings; j++) {
                sensors[i].collect();
            }
            TimeUnit.SECONDS.sleep(sleep);
        }

        for (int j = 0; j < sensors.length; j++)
            sensors[j].updateDisplay(display, readings);
    }
}
```

5. (10 Points) This is code for a simple customer accounts system.

```
public class Customer {
    private String firstName;
    private String lastName;
    private String custNumber;
    private double balance = 0;
    // ... getters and setters too
}

public class Transaction {
    private Date date;
    private String description;
```

```
    private double amount;
    // ... getters and setters too
}

public class Account {
    private String acctId;
    private Customer customer;
    private Transaction[] transactions;

    public void deposit(double amount) { // ...
    }

    public void withdraw(double amount) { // ...
    }

    public double getBal() { // ...
    }

    public String getAcctId() { // ...
    }

    public void printStatement() { // ... print out all transactions
    }
}
```

Violations:

- Single Responsibility Principle: For the life of me I can't figure out why the customer maintains the balance field rather than the account. Also why does the customer not have an accounts field containing multiple accounts. It seems that the relationship between customer and accounts should be 1..*, balance should be in account, and account should not be managing the balance in customer. Also, customer should probably have the deposit and withdraw functions which then call an update action on account by passing a transaction statement (after verifying the balance).

```
public class Customer {
    private String firstName;
    private String lastName;
    private String custNumber;
    private List<Account> accounts;
    // ... getters and setters too

    public void deposit(Account acct, double amount) { // ...
        acct.update(Transaction.create("deposit", amount));
    }

    public void withdraw(Account acct, double amount) { // ...
        if (acct.getBal() > amount)
            acct.update(Transaction.create("withdrawl", amount));
        else
            throw new OverdraftException("Balance to low to withdraw that amount");
    }
}
```

```
public class Transaction {
    private Date date;
    private String description;
    private double amount;
    // ... getters and setters too

    public static Transaction create(String desc, double amt) {
        return new Transaction();
    }

    private Transaction(String desc, double amt) {
        date = Date.Now;
        description = desc;
        amount = amt;
    }
}

public class Account {
    private String acctId;
    private double balance = 0;
    private Transaction[] transactions;

    public void update(Transaction t) { // ...
    }

    public double getBal() { // ...
    }

    public String getAcctId() { // ...
    }

    public void printStatement() { // ... print out all transactions
    }
}
```