Object Oriented Software Engineering

600.421

Assignment 2: Object-Oriented Design

Submitted By: Sindhuula Selvaraju

JHED : sselvar4

In Collaboration With(Question 2): Anchit Pandya (apandya5)

Question-1:

1. Feature List

a. Fitness Center can add new members to the database
b. The app maintains a unique Member ID for each member
c. Fitness Center can update membership type for a member at anytime
d. App calculates additional cost for the changing membership type
e. Period of membership is entered for each customer and can be 1,6 or 12months
f. Number of personal training sessions for each member is saved in the database
g. App automatically gives 5 free personal training sessions to premium members
h. Fitness Center can add number of personal training sessions for each member
i. App decides what all a member has access to based on his membership type
j. All previous memberships of a member is saved in the database ie when a membership is renewed it adds a new record to the database keeping the old record intact
k. Fitness Center can create a personal trainer session for a member
l. Database stores the information of the personal training sessions
m. App checks if the trainer is available at the time requested for personal training session
n. Application keeps tab on if a membership needs renewal and alerts the Fitness Center
o. Application should also be able to check all future training sessions of the member
p. Fitness center checks when a member visits if their membership is inactive
q. Fitness Center allows 1 late month based on how long a member has been in the Fitness Center
r. Fitness Center can update pricing of the Memberships
s. Fitness Center allows Members to renew their membership

2. Use Cases

1) New Member Creation

Actors : Fitness Center, Database, Application

Goal : Add a new member record to the database

1.1 Fitness Center enters the new member record to the app
1.2 Application asks if the new member needs to be added to the database

1.3 Application also adds additional data to the database like the free training sessions(if membership type is Premium), facilities accessible along with data it gets from the Fitness Center
1.4 The member is given a new internal member ID

2) Updating Membership Type:

Actors : Fitness Center, Database, Application

Goal : Update record in the database

2.1 Fitness Center asks for change in MembershipTypeApplication calculates the additional fees payable by the member for the change
2.2 Application makes the change in the database
2.3 Application returns the status of the change to the Fitness Center

3) Add Personal Training Sessions

Actors : Fitness Center, Database, Application

Goal : Update record in the database with new number of sessions

3.1 Fitness Center asks for adding number of training sessions for a member
3.2 Application adds the new number of sessions to the already existing data in the database

4) Member access:

Actors : Fitness Center, Database, Application

Goal : check record in the database

4.1 Member visits Fitness Center
4.2 Fitness Center asks Application to check what the member has access to
4.3 Application queries the Database
4.4 Application responds to Fitness Center
4.5 Fitness Center checks if member is trying to access restricted areas and responds accordingly

5) Scheduling Personal Training Session

Actors : Fitness Center, Database, Application

Goal : Add new training session record in the database

5.1 Fitness Center asks for adding new training session record
5.2 Application checks if the member has personal training sessions
    5.2.1 If the member has sessions:
        5.2.1.1 Application checks if the trainer is available
            5.2.1.1.1 If the trainer is available then create new training session
            5.2.1.1.2 If the trainer is unavailable send failure message to Fitness Center
    5.2.2 If the member does not have sessions:
        5.2.2.1 Send failure message to the fitness center

6) Checking for Overdue:

Actors : Fitness Center, Database, Application

Goal : Check database for overdue

6.1 Member visits Fitness Center
6.2 Fitness Center asks application to check status
6.3 Application queries the database.
6.3.1 If status is current then the member can continue
6.3.2 If the status is overdue:
6.3.2.1 If the member has been a member for more than 6 months continuously then a grace period of 1 month is given
6.3.2.2 If the member has not been a member for more than 6months then the account status is set to suspended
6.3.2.3 If the member has gone beyond the grace period then the account status is set to suspended
6.3.2.4 If the member has not paid for a long time the status is set to inactive
6.3.3 If the status suspended or inactive send response to Fitness Center

7) Update Membership Pricing:

Actors : Fitness Center, Database, Application

Goal : Update Membership Type record in database

7.1 Fitness Center sends new price list to the Application
7.2 Application updates corresponding Membership Type records in the database
7.3 Application does not update any member records

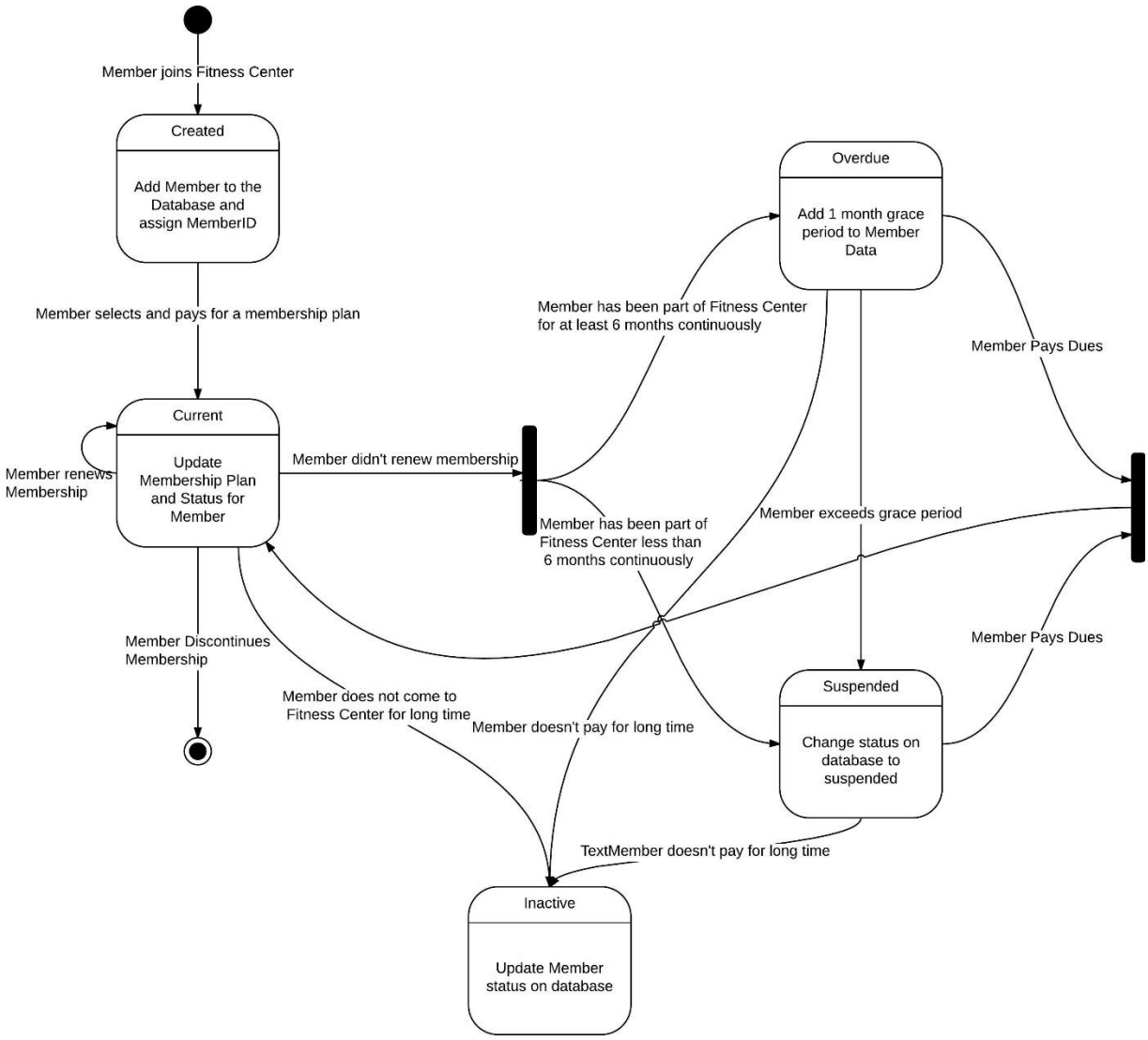
8) Renew Membership:

Actors : Fitness Center, Database, Application

Goal : Update Member Record in Database

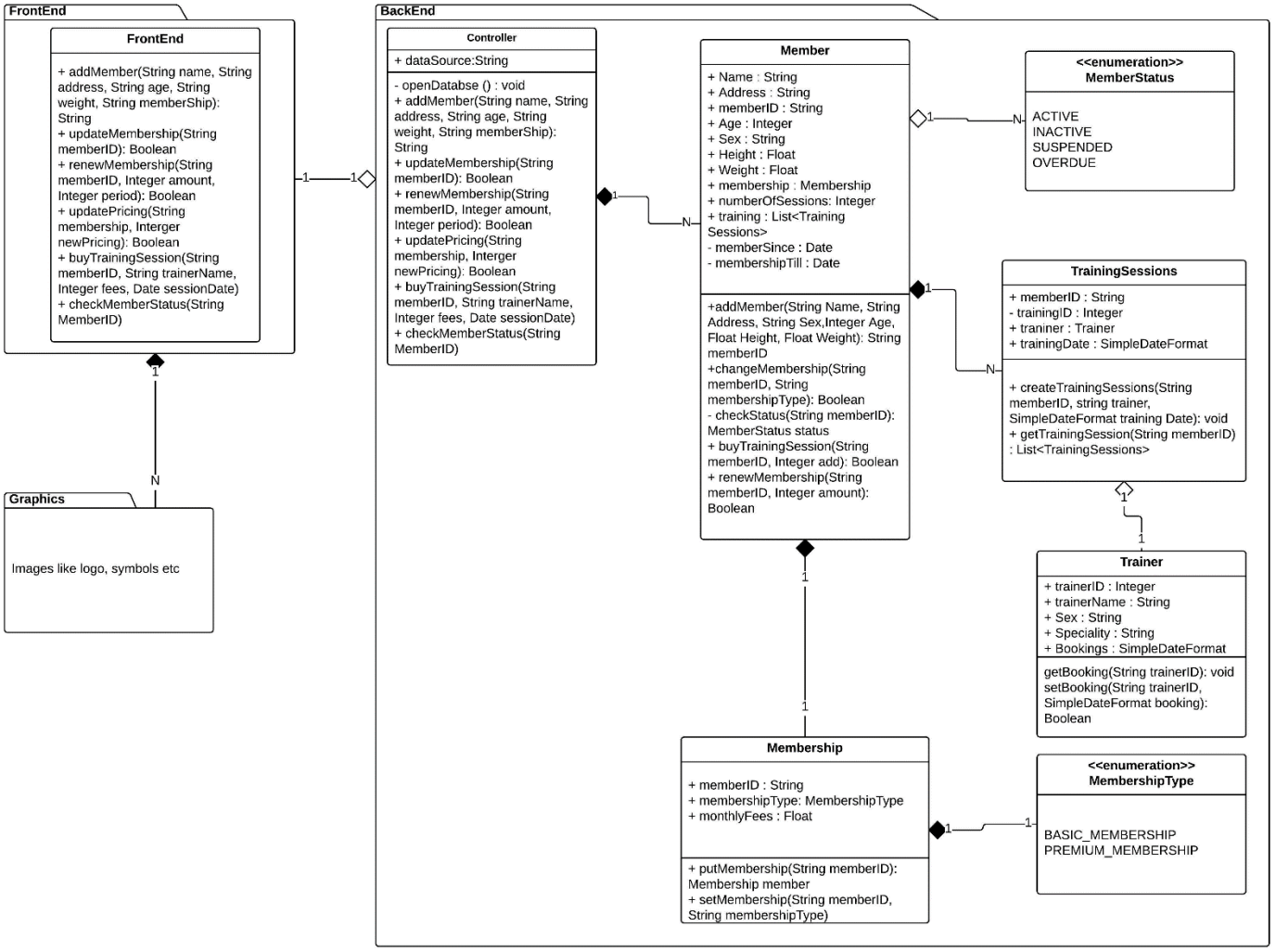
8.1 Fitness Center sends renewal request to Application
8.2 Application queries the Database to check if the member needs to renew:
8.2.1 If member does not require renewal send failure to Fitness Center
8.2.2 If member needs to renew check status:
8.2.2.1 If the status is current add a new record to the historical data and update membership data
8.2.2.2 If the status is overdue
8.2.2.2.1 Check if the amount being payed covers overdue. If it does update status to current and send success response add a new record to the historical data and update membership data.(Also add free training sessions for Premium members)
8.2.2.2.2 If the amount payed does not cover it respond with failure message asking for overdue amount
If the status is suspended or inactive create a new record for the member in historical data and update membership data. (Also add free training sessions for Premium members)

3. State Diagram



4. Class Diagram

Since, we are writing a piece of software for this question I am assuming the Front-end to be a form. So instead of HTTP methods I am directly using the backed methods. If HTTP methods are used there would simply be another class in the Back-end that would just add Endpoints and will instead lead to the controller class.



5. Additional Requirements

There are several requirements that can be added to the requirements can be adding like:

- Even though not mentioned it would be necessary to maintain the trainer-member relationship specially, since members would prefer being in sessions with the same trainer. I have included this in my class diagram but there can also be a use case for this
- For members:
    - Giving discounted rates for long term members – this will affect the use cases with an added case for calculation when renewing membership
    - Sending reminder mails to the members who've been lapse in their payments – this will add data member email to the Member class and use case for checking lapse payments and sending reminder mail
    - Saving special conditions for members like if they are diabetic or have a heart condition and suggesting trainers and training sessions accordingly – this would require a sub-class under member to save and monitor these conditions. There will also be an additional use case for the same

- Added to this the Fitness Center may also want to maintain payroll of employees and condition of the fitness equipment where the class diagram will also have added classes

Question-2:

OCP: Open-Closed Principle

DRY: Don't Repeat Yourself

LSP: Liskov Substitution Principle

SRP: Single Responsibility Principle

1)Collection Library:

Principle	OCP	DRY	LSP	SRP
Violates	No	Yes	Yes	No

The code violates LSP because the base class, super class relation is incorrect. The class Set that takes only single instance of value is inheriting the class MyMultiSet that takes multiple instances of value. Instead I think there should be a reversal of roles where MyMultiSet either inherits from Set or uses Set as its member.

```
/* A generic interface for collections */
interface Collection<T>
{
   public void add(T v);

   public void remove(T v);

   public int count(T v);
}

/* Sets on the other hand only a single instance of a value */
class Set<T>
{
  private T data;

  public Set() {
     data = new T();
  }

 public void add(T v) {
     if (count(v) == 0) {
        data = v;
     }
}
```

```
/* Multisets can hold more than one instance of a value */
class MyMultiSet<T> implements Collection<T>
{
  private List<Set> data;

  public MyMultiSet() {
     data = new LinkedList<Set>();
  }

  public void add(Set v) {
     data.add(v);
  }

  public void remove(Set v) {
     data.remove(v); //Removes the first element equal to v
  }

  public int count(Set v) {
     int c = 0;
     for(T i : data) {
        if (v.equals(i)) c++ ;
     }
     return c;
  }
}
```

2)Hack–and-slash Game:

Principle	OCP	DRY	LSP	SRP
Violates	Yes	Yes	No	No

The code violates OCP because there is no chance of expansion in case if new weapon types need to be added. For this I would have weaponType as its own class so that sword, spear etc can be objects of the class.

The code violates DRY because identical functions are being implemented for different weapon types. I would instead suggest passing weaponType also as a parameter so that I get both the type and the damage that the weapon does.

Corrected Code:

```
public class weaponType
{
   public Int weaponType
   public Int weaponDamage
}
```

```
public class Player
{
   private weaponType weapon;

   public void attack(Monster target) {
            attackWithWeapon(target,weapon);
   }

   public void slashWithSword(Monster target, weaponType weapon) {

  /* Do some work here to compute the damage */
      target.addDamage(weapon.weaponDamage);
   }
}

   /* Other code and definitions here */
}
```

3)Terminal Version of Hare and Hounds:

Principle	OCP	DRY	LSP	SRP
Violates	No	No	No	Yes

This code violates SRP because there is a single function that check, move, print state etc. So instead have 2 functions, 1 that checks the positions and one that make the move

Corrected Code:

```
class Game {
   private Piece[][] board;
   public Boolean check()
  {
      /* Does the source square have a piece? Is the desination empty? */
       if (board[source.x][source.y] == null || board[dest.x][dest.y] != null)
          return false;
       return true;
   }
   public void move() {
       /* Read user input */
       Point source = readCoordinateFromInput();
       Point dest = readCoordinateFromInput();
       Boolean result  = Check(dest);
       if(result)
  {

       /* ... More rules here .. */
```

```
      /* Print out the current state of the board */
      printBoard();
  }
     else {throw new IllegalMoveException();}
  }

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

4) Hare and Hounds Game:

Principle	OCP	DRY	LSP	SRP
Violates	No	Yes	No	Yes

This code violates DRY because after every check there is code for move. This move code would essentially be identical in every case save for the parameters which can be passed as part of the move function

This code also violates SRP because a single method is checking also and moving also. Instead I would have 1 method that moves and one that checks so that the responsibility is split between the two. Also the conditions can be compacted better like the if and the second else if conditions vary only in 1 parameter so why not include it as an or condition.

Corrected Code:

```
class Game {
   /* Other code and definitions here */
   private void check(Point src, Point dest) {
      Piece piece = getPieceAt(src);

      /* Check if we are moving along a valid edge. Also ensure that hounds cannot move
backwards */
Piece destPiece = getPieceAt(dest);
      if (src.x == 0 && src.y == 1 && dest.x == 1 && (dest.y == 0|| dest.y == 2 )&& destPiece
== null) {
            move(src,dest,destPiece);
      }
 else if (src.x == 1 && src.y == 0 && dest.x == 0 && dest.y == 1
               && piece != null && piece.getPieceType() != PieceType.HOUND) {
      move(src,dest,destPiece);
```

```
        }
        /* More else-if conditions here */
    }
    private void move(Point src, Point Dest, Piece piece)
{
            /* Code to move the piece to the new position */

}
    private Piece getPieceAt(Point pt) {
        /* Code here to return the piece at the given position. Returns null
           if none exist */
    }
}
```