Object-Oriented Software Design
Assignment 2: Object-Oriented Design

Li-Yi Lin / llin34@jhu.edu

1 Question 1: Fitness Center Software Design

I) Feature List:

(a) Membership:

(1) Allow the center to create a new member account for a new customer.
(2) Allow the center to set the membership of a member account.
(3) The membership can be bought for one month, six months, and one year at a time.
(4) The membership must have two kinds of memberships: basic and premium memberships.
(5) Allow the center to renew a membership.
(6) Allow a member to switch from basic to premium membership by paying a prorated membership fee.
(7) Ensure switching from premium to basic is allowed only during renewal.
(8) Be able to calculate the membership fee.
(9) Be able to keep track of customers and their memberships including historical information.
(10) Allow the center to deactivate, suspend, activate a membership.

(b) Access control:

(1) Have access controls for facilities according to different memberships.
(2) The basic and premium memberships shall be able to access most facilities (including the weight room and the pool).

(c) Personal trainer:

(1) Allow the premium membership to have five free appointments with a personal trainer per month.
(2) Allow a member to pay small fee to get personal training appointments.
(3) Be able to keep track of personal trainer appointments and avoid double-booking.
(4) Be able to prevent a member from booking more than one appointments for the same time period.
(5) Allow a member to cancel her/his personal trainer schedule.

(d) Membership fee:

(1) Allow the center to update the pricing of the memberships.

(2) The change in price should not affect the current memberships until it is renewed.

(3) Allow members to lapse in their payments.

(4) Allow members to be late up to a month without being refused entry to the fitness center. But this applies only to customers who have had a membership (continously) for at least the last six months and have not lapsed in payment during that time frame.

(5) Be able to automatically suspend a membership when some conditions are met (e.g. not pay membership fee one month later after the membership expiration date).

II) Use Cases: (x.x means alternate path)

(a) Create member account:
1. The staff opens the "Create Account" function.
2. The staff fills in information about a new member.
3. The staff sets the membership (basic or premium) of the new account.
4. The staff sets the membership duration (one month, six months, or one year).
5. The staff clicks the "Create" button and the data is sent to backend systems.
6. The system replies success message.
6.1. The system replies error message if creating account failed.

(b) Query member:
1. The staff opens the "Manage Member" function.
2. The staff enter the member account in the search area and clicks the "Search" button.
3. The system replies member's data.
3.1. The system replies "member not found."

(c) Query members' membership history:
Precondition: The staff has found the member using "Query member" function.
1. In the query member's page, the staff clicks "Membership History" button.
2. The system replies the result and shows the result on a membership history page.

(d) Renew membership:
Precondition: The staff has found the member using "Query member" function.
1. The staff clicks the "Renew Membership" button.
2. The system shows that the member could renew the membership. The member can renew her/his membership only when the remainder duration of the membership is no more than one month (this is an assumption).
2.1. If the member's status is not qualified for membership renewal, the system will show a warning message. Stop this operation.
3. The staff chooses a membership type (basic or premium).
4. The staff chooses a membership duration (one month, six months, or one year). 5. The staff clicks "Renew" button.
6. The system shows a page about the renewal information and asks the staff to confirm this renewal.

7. The staff clicks ”Confirm” button and the data is sent to the backend systems.
8. The system replies success message.
 8.1. The system replies error message.

(e) Suspend a membership:
Precondition: The staff has found the member using ”Query member” function and the member’s status is neither suspended nor inactive.
1. The staff clicks ”Change Member Status” button and the system shows a status management page.
2. The staff chooses a new status for this member and enters a reason for this change (could select from a reason list).
3. The staff clicks ”Change” button.
4. The system shows a confirmation page and the staff clicks ”Confirm” button.
5. The system replies success message.
 5.1. The system replies error message.

(f) Deactivate a membership:
Precondition: The staff has found the member using ”Query member” function and the member’s status is not inactive.
1. The staff clicks ”Change Member Status” button and the system shows a status management page.
2. The staff chooses a new status for this member and enters a reason for this change (could select from a reason list).
3. The staff clicks ”Change” button.
4. The system shows a confirmation page and the staff clicks ”Confirm” button.
5. The system replies success message.
 5.1. The system replies error message.

(g) Switch membership type:
Precondition: The staff has found the member using ”Query member” function and the membership type is basic and the member status is ”current”.
1. The staff clicks ”Switch Membership Type” button and the system shows a new page for switching membership type.
 1.1. If the current status of the member is not suitable for swithcing membership type, the system will show a warning message. Stop this operation.
2. The staff chooses a new membership type for this member.
3. The staff clicks ”Switch” button.
4. The system shows a confirmation page and the staff clicks ”Confirm” button.
5. The system replies success message.
 5.1. The system replies error message.

(h) Book personal Trainer:
Precondition: The staff has found the member using ”Query member” function and the member status is ”current.”
1. The staff clicks ”Book Personal Trainer” button and the system shows a page for booking personal trainer.
2. The staff choose a time slot.
3. The staff choose a personal trainer if available.
 3.1 There is no personal trainer available. Go back to step 2 and choose another time slot.

4. The staff clicks ”Create” button.
5. The system shows a confirmation page.
 5.1 The system shows error message if the member has no enough personal trainer quota to use. Stop this operation.
6. The staff clicks ”Confirm” button.
7. The system replies success message and the personal trainer hours will be substracted from the member’s account.
 7.1 The system replies error message.

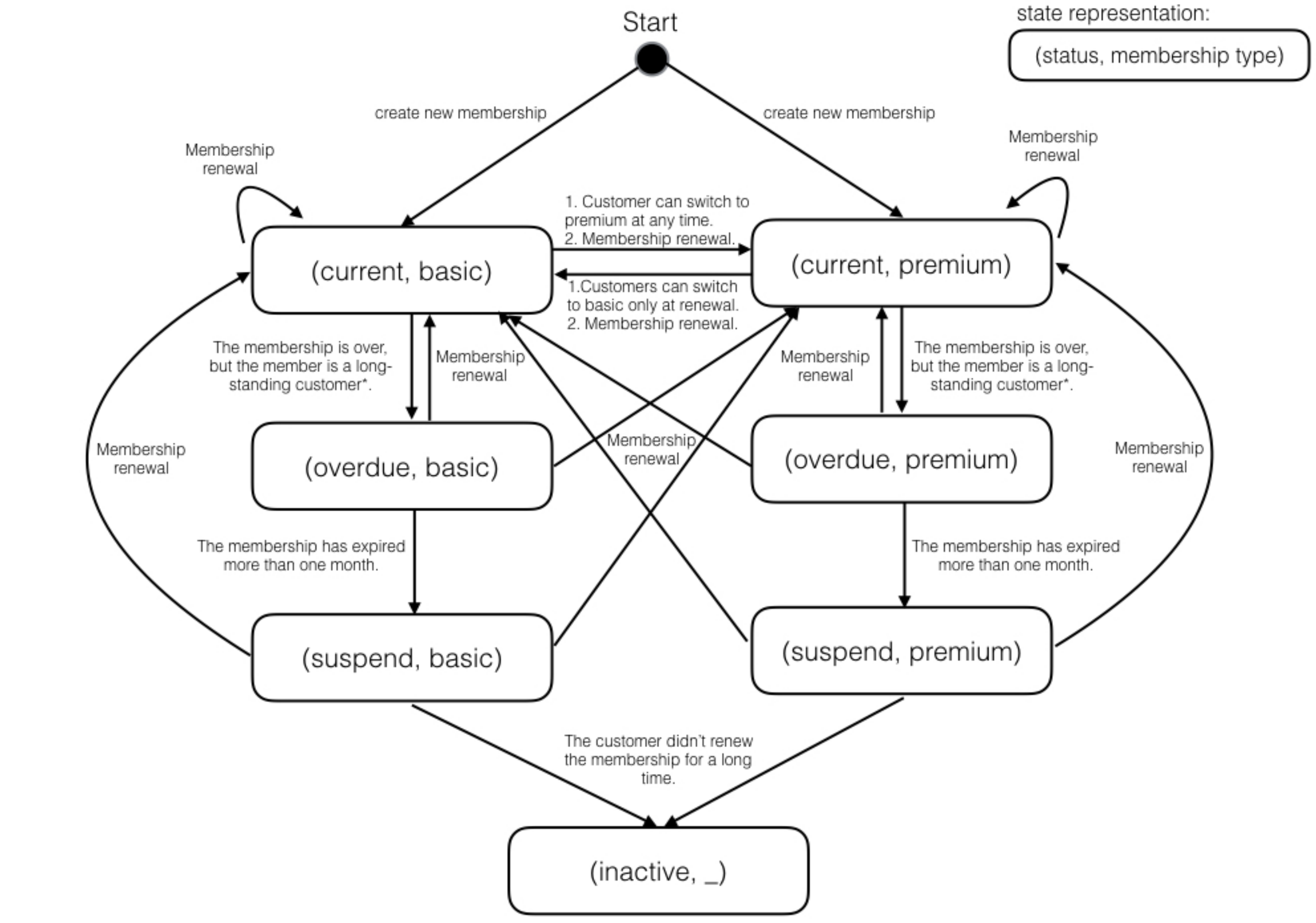
(i) Cancel personal trainer schedule:
Precondition: The staff has found the member using ”Query member” function.
1. The staff clicks ”Personal Trainer Schedule” button and the system shows a page of the member’s schedules for personal trainer.
2. The staff choose a unexpired schedule and the system shows a page about this detailed personal trainer schedule.
 3.1 There is no personal trainer available. Go back to step 2 and choose another time slot.
4. The staff clicks ”Create” button.
5. The system shows a confirmation page.
 5.1 The system shows error message if the member has no enough personal trainer quota to use. Stop this operation.
6. The staff clicks ”Confirm” button.
7. The system replies success message and the personal trainer hours will be added back to the member’s account.
 7.1 The system replies error message.

(j) Pay for personal trainer:
Precondition: The staff has found the member using ”Query member” function and the member status is ”current.”
1. The staff clicks ”Add Personal Trainer Hour” button and the system shows a new page for adding hours for this member.
 1.1. If the member status is not suitable for this operation, the system will show a warning message. Stop this operation.
2. The staff enters the number of hours the member want to add for the personal trainer.
3. The system shows the total amount of fee the member should pay for it.
4. The staff clicks ”Add” button.
5. The system shows a confirmation page and the staff clicks ”Confirm” button.
5. The system replies success message and the personal trainer hours will be added to the member’s account.
 5.1. The system replies error message.

(k) Update the pricing of the memberships:
1. The staff clicks ”Membership Configuration” button and the system shows a new page for configuring membership setting.
2. The staff clicks ”Membership Pricing” button and the system shows a new page for changing membership pricing.
3. The staff enter a new membership pricing for the membership she/he want to change.
4. The staff clicks ”Update” button.

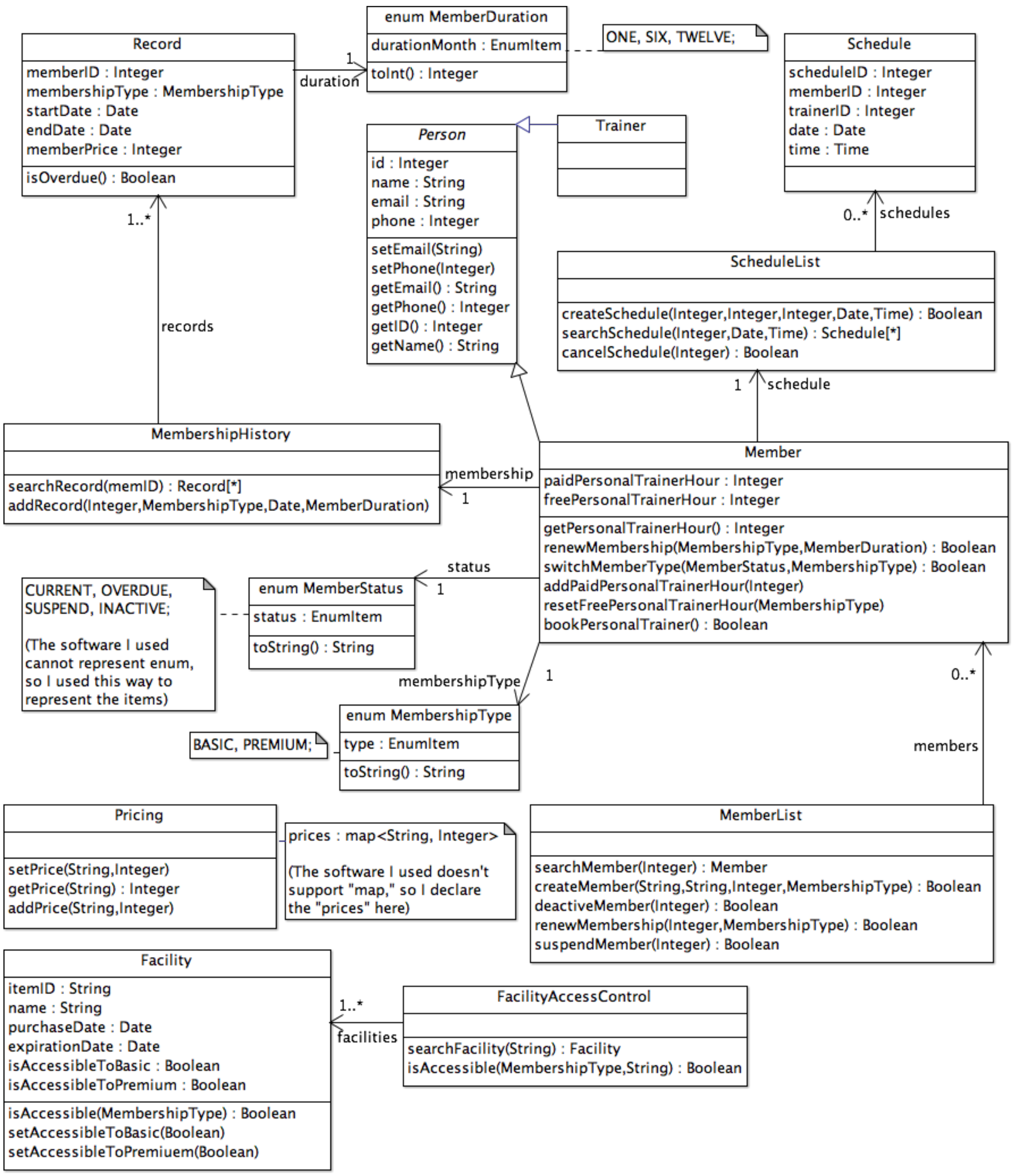
5. The system shows a confirmation page and the staff clicks "Confirm" button.
6. The system replies success message.
 6.1 The system replies error message.

III) State Diagram:


Start
state representation:
(status, membership type)
create new membership
create new membership
Membership renewal
Membership renewal
1. Customer can switch to premium at any time.
2. Membership renewal.
1.Customers can switch to basic only at renewal.
2. Membership renewal.
(current, basic)
(current, premium)
The membership is over, but the member is a long-standing customer*.
Membership renewal
Membership renewal
The membership is over, but the member is a long-standing customer*.
Membership renewal
Membership renewal
Membership renewal
(overdue, basic)
(overdue, premium)
The membership has expired more than one month.
The membership has expired more than one month.
(suspend, basic)
(suspend, premium)
The customer didn't renew the membership for a long time.
(inactive, _)

* Long-standing customer is who have had a membership (continously) for at least the last six months and have not lapsed in payment during that time frame.

IV) UML Class Diagram:


Record
memberID : Integer
membershipType : MembershipType
startDate : Date
endDate : Date
memberPrice : Integer
isOverdue() : Boolean
1
duration
enum MemberDuration
durationMonth : EnumItem
toInt() : Integer
ONE, SIX, TWELVE;
Schedule
scheduleID : Integer
memberID : Integer
trainerID : Integer
date : Date
time : Time
Person
id : Integer
name : String
email : String
phone : Integer
setEmail(String)
setPhone(Integer)
getEmail() : String
getPhone() : Integer
getID() : Integer
getName() : String
Trainer
1..*
records
0..*
schedules
ScheduleList
createSchedule(Integer,Integer,Integer,Date,Time) : Boolean
searchSchedule(Integer,Date,Time) : Schedule[*]
cancelSchedule(Integer) : Boolean
1
schedule
MembershipHistory
searchRecord(memID) : Record[*]
addRecord(Integer,MembershipType,Date,MemberDuration)
membership
1
Member
paidPersonalTrainerHour : Integer
freePersonalTrainerHour : Integer
getPersonalTrainerHour() : Integer
renewMembership(MembershipType,MemberDuration) : Boolean
switchMemberType(MemberStatus,MembershipType) : Boolean
addPaidPersonalTrainerHour(Integer)
resetFreePersonalTrainerHour(MembershipType)
bookPersonalTrainer() : Boolean
status
1
CURRENT, OVERDUE, SUSPEND, INACTIVE;
(The software I used cannot represent enum, so I used this way to represent the items)
enum MemberStatus
status : EnumItem
toString() : String
membershipType
1
0..*
enum MembershipType
type : EnumItem
toString() : String
BASIC, PREMIUM;
members
Pricing
setPrice(String,Integer)
getPrice(String) : Integer
addPrice(String,Integer)
prices : map<String, Integer>
(The software I used doesn't support "map," so I declare the "prices" here)
MemberList
searchMember(Integer) : Member
createMember(String,String,Integer,MembershipType) : Boolean
deactiveMember(Integer) : Boolean
renewMembership(Integer,MembershipType) : Boolean
suspendMember(Integer) : Boolean
Facility
itemID : String
name : String
purchaseDate : Date
expirationDate : Date
isAccessibleToBasic : Boolean
isAccessibleToPremium : Boolean
isAccessible(MembershipType) : Boolean
setAccessibleToBasic(Boolean)
setAccessibleToPremiuem(Boolean)
1..*
facilities
FacilityAccessControl
searchFacility(String) : Facility
isAccessible(MembershipType,String) : Boolean

V) Extended Requirements:

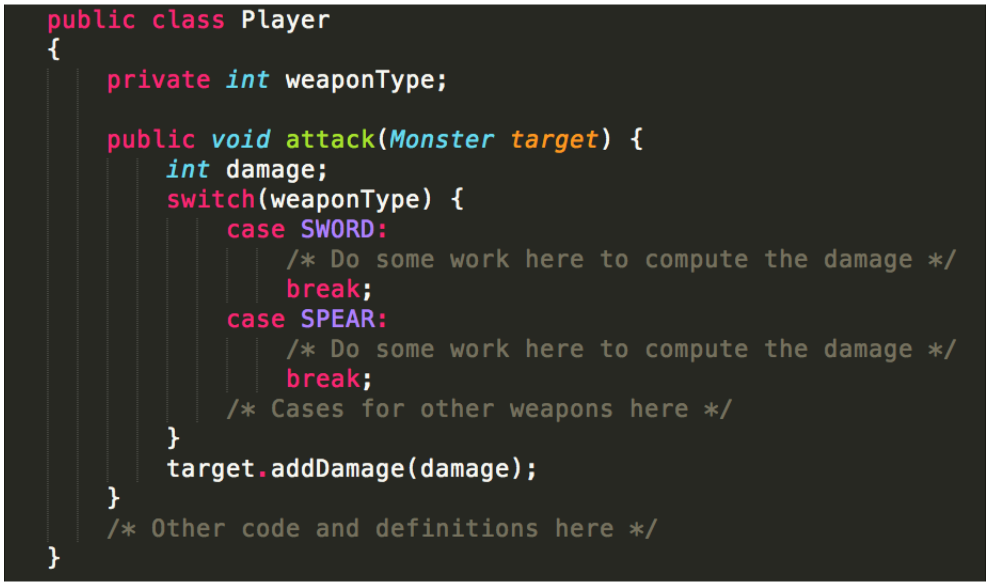
1. The system can allow the center to manage facilities.
2. The system can allow the center to change the access control for different kinds of memberships.

3. The system can allow the center to manage the personal trainers' information.
4. The system can allow the center to manage the time slots of personal trainers.
5. The system can automatically renew the membership if the member provides her/his credit card information and permit the center to do so.
6. The system can repeat the personal trainer schedule periodically for members.
7. The system can send personal trainer schedule reminder (using SMS or email) to members.
8. The system can a have a member page for members to reserve the personal trainer online.

2 Question 2: Object-Oriented Design Principles

Case 1:
In case 1, the code violates the ”Liskov Substitution Principle (LSP)” because when ”MyMultiSet” class is replaced with its subclass ”Set,” the result of calling ”add” method will be different. The ”add” method of ”MyMultiSet” will always add the item into its list ”data.” However, the subclass ”Set” will add a new itme into its list ”data” only when its list ”data” has no item equals to the new item. In some cases, the subclass will override the method inherited from its superclass on purpose (like the ”add” method in ”Set”). If we really want to correct the code so that it will not violate LSP, we can change the method name of ”add” in class ”Set” to another name so that when ”MyMultiSet” is replaced with subclass ”Set”, the result will not change.
This case could also be ”Open Close Principle (OCP)” violation because subclass can modify the superclass' methods. To close the superclass' methods for modification, we can declare its methods as ”final.”

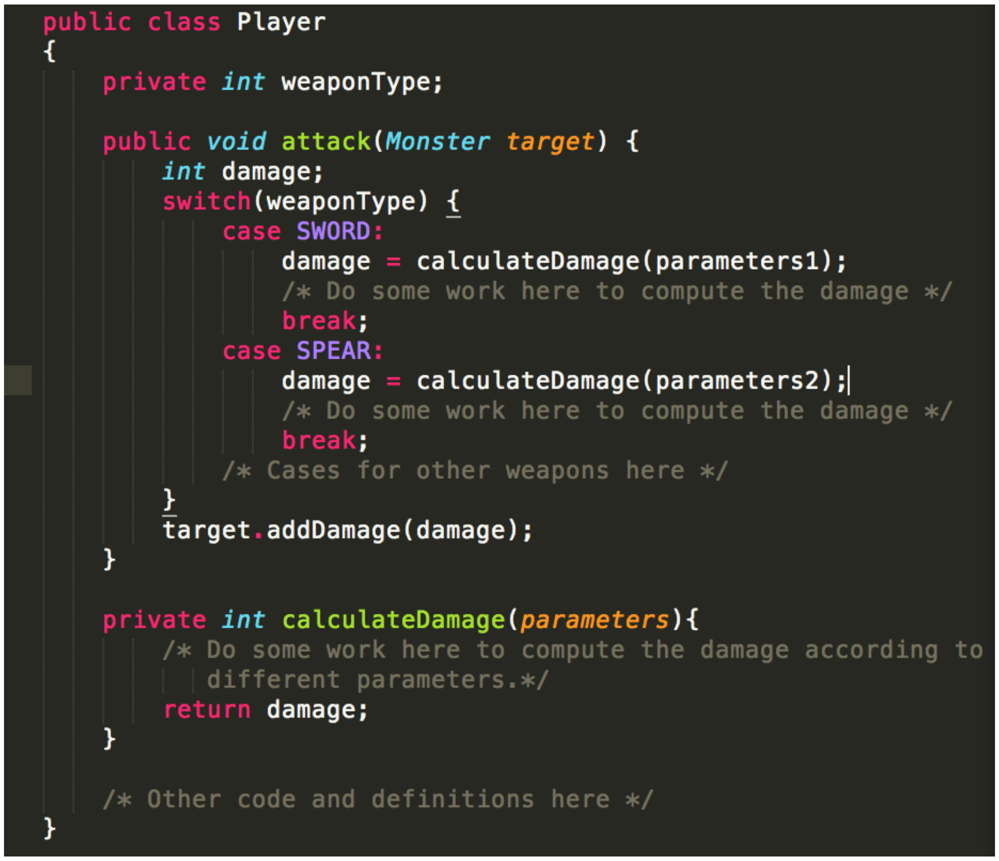
Case 2:
In case 2, the code violates the ”Don't Repeat Youself (DRY)” principle. The actions of ”slashWithSword(Monster target)” and ”pokeWithSpear(Monster target)” are almost the same except the calculation of differenet damages. This DRY violation can be corrected if we calculate the damage value according to different weapon types in the switch section. After the switch section, add ”target.addDamage(damage);” before the end of the ”attack” method. So the code will become



```
public class Player
{
    private int weaponType;

    public void attack(Monster target) {
        int damage;
        switch(weaponType) {
            case SWORD:
                /* Do some work here to compute the damage */
                break;
            case SPEAR:
                /* Do some work here to compute the damage */
                break;
            /* Cases for other weapons here */
        }
        target.addDamage(damage);
    }
    /* Other code and definitions here */
}
```

Thus, the ”slashWithSword(Monster target)” and ”pokeWithSpear(Monster target)” methods can be deleted. Furthermore, if the damage calculations are very similiar between different weapon types, we can create a new method to calculate the damage value according to different parameters. The code will become



```
public class Player
{
    private int weaponType;

    public void attack(Monster target) {
        int damage;
        switch(weaponType) {
            case SWORD:
                damage = calculateDamage(parameters1);
                /* Do some work here to compute the damage */
                break;
            case SPEAR:
                damage = calculateDamage(parameters2);
                /* Do some work here to compute the damage */
                break;
            /* Cases for other weapons here */
        }
        target.addDamage(damage);
    }

    private int calculateDamage(parameters){
        /* Do some work here to compute the damage according to
           different parameters.*/
        return damage;
    }

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

Case 3:
In case 3, the code violates the ”Single Responsibility Principle (SRP)” because the ”Game” class are responsible for getting user input, moving a piece, and printing the board, which are not necessarily the responsibilities the ”Game” class should do. To correct the SRP violation, we can create a ”Board” class that deals with moving pieces and printing the board. And the ”Game” class deals with getting user input and interacting with the ”Board” class. Therefore, each class can focus on the responsibilities it should do.

Case 4:
In case 4, we assume the ”Game” class is responsible for moving a piece. The code violates the ”Don’t Repeat Youself (DRY)” principle because it lists all the ckecks for all the possible moves for a piece. This will create many duplicate codes. To correct the DRY violation, it should analyze the possible moves and simplify the checks. To simplify the checks, it should first check a rule that will reduce following checks as much as possible and so on. By doing so, the move check will become much simpler and will not have duplicate codes.