CSSE220: Example Design Problems Relating to Principles 3, 4, and 5

For maximum benefit, I encourage you to attempt to solve these problems yourself before peeking at the solutions document.

All of these problems relate to Design Principle 3, 4, and 5

This document includes example problems and space for you to write in solutions. In every case, the instructions are:

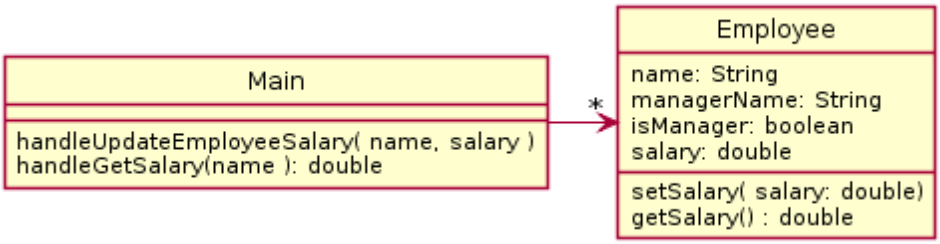
1. Identify the problems with Solution A & B using your design principles
2. Design a new solution that solves all problems
    a. **To Do #1** Identify all the *primary nouns*
    •A *primary noun* is a noun in the problem that has *attributes (other nouns*)
    •Nouns that designate *actors* of the system (i.e. The *user* can click…) can be **excluded**
    b. **To Do #2** Write down the *attributes* (*other nouns)* associated with the *primary nouns*
    c. **To Do #3** Identify all the *verbs*
    d. **To Do #4** Identify which *primary nouns* are worked on by the *verbs*
    e. **To Do #5** Design a system using UML to handle this problem

Table of Contents

Employee Salary (in-class exercise SETechniques)

There is a company which has employees, each of which has a salary. There are specific employees which oversee other employees called managers. While employees have their salaries which can be updated from time to time. Unlike employees, a manager's salary is always 10% more than the salary of their top paid employee. What is wrong with the following design? Design your own improved UML diagram.

Bad Design A



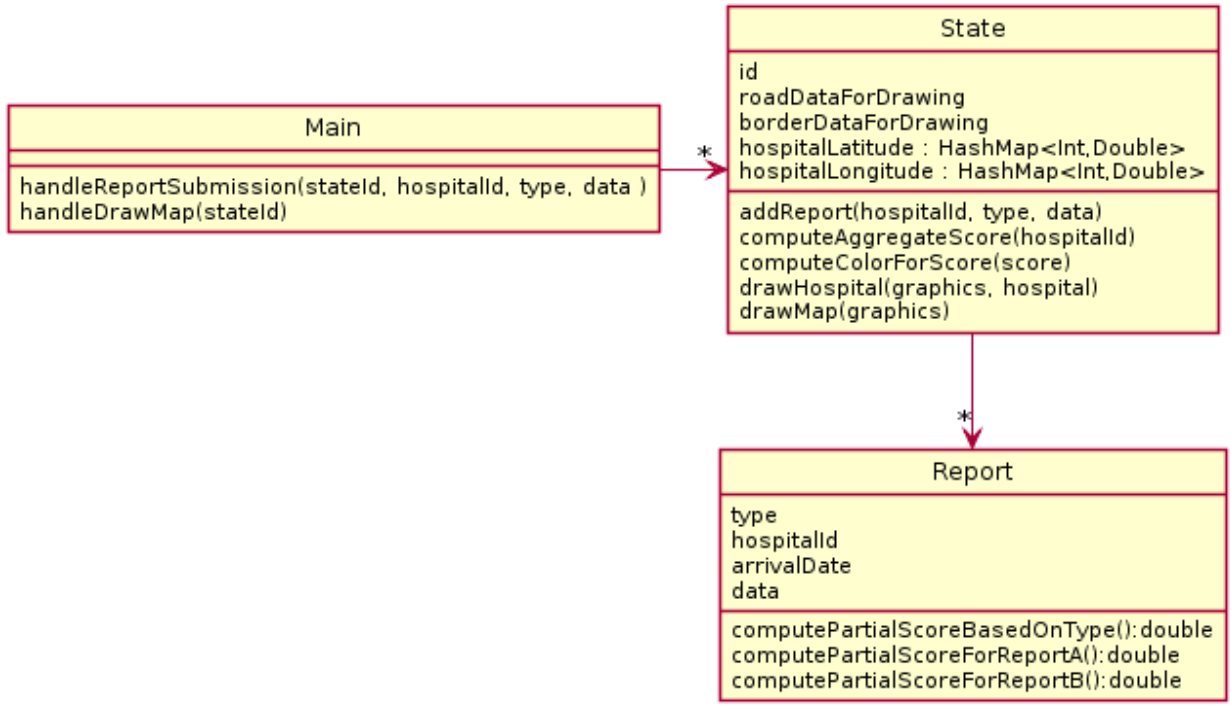
1) Describe the problems with Bad Solution

2) Design your own solution

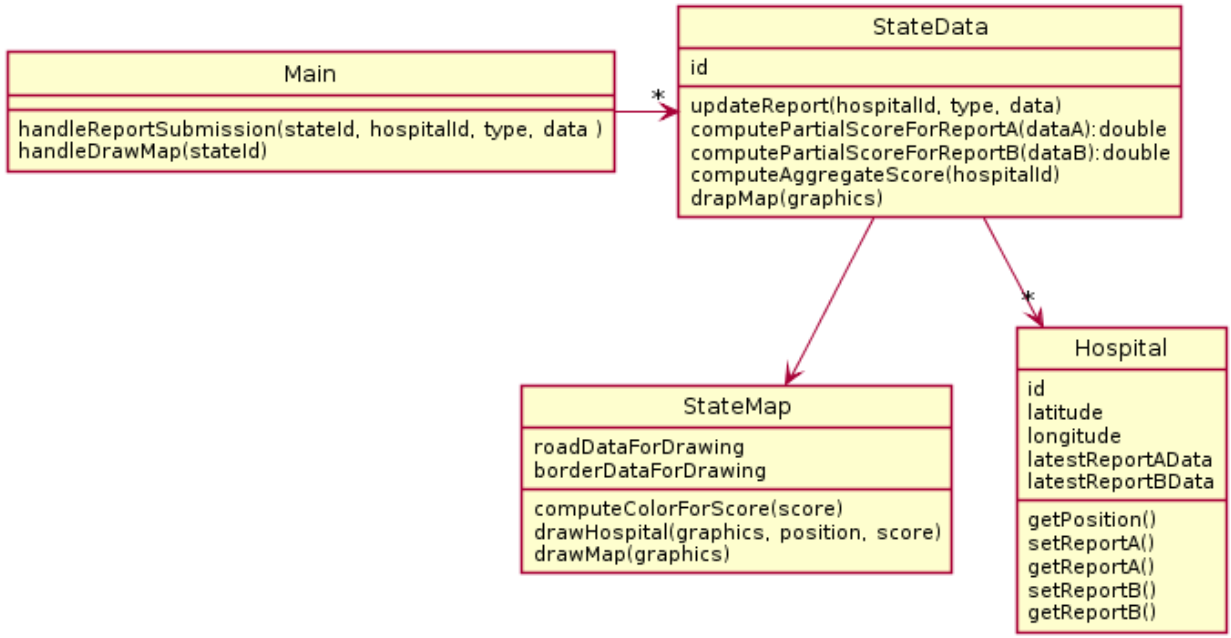
State Hospital

A government agency tracks hospital data to make a monthly report. The highlight of the report is a set of colorful maps that show every hospital in a particular state with colors indicating the hospital's overall status. To make this report, hospitals submit two different kinds of reports (reportA, reportB) that come in at different times. Each of these reports is analyzed differently and produces a partial score. The final map color is produced by combining these three scores into an aggregate score, using the most up-to-date version of each report available at that time. Color on the map is positioned according the the hospital's specific latitude and longitude which is stored in the system and does not change. The drawn map also includes the states borders and major roads.

Design A



Design B



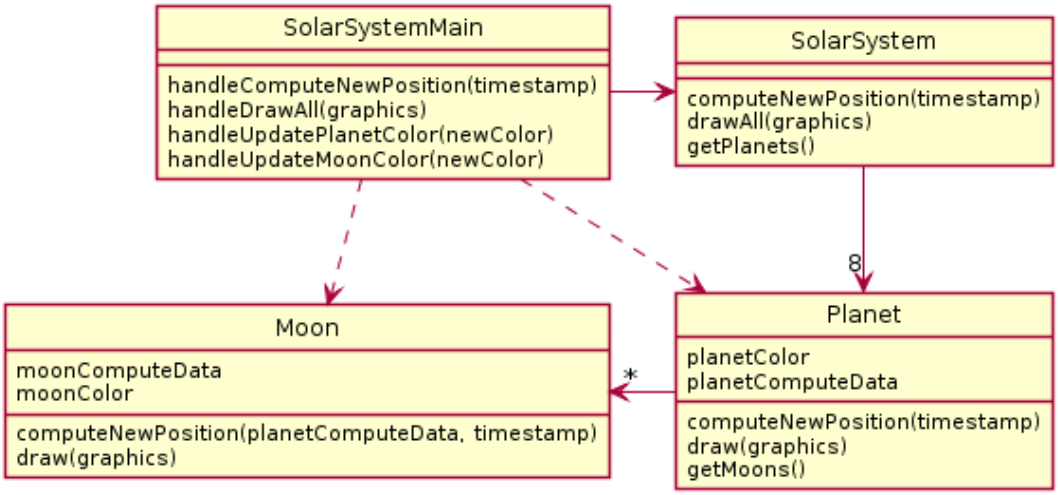
1) Describe the problems with Design A and B

2) Design your own solution

Solar System (in-class exercise SETechniques)

A Java program draws a minute-by-minute updated diagram of the solar system including all planets and moons. To update the moon's position, the moon's calculations must have have the updated position of the planet it is orbiting. The diagram is colored - all planets are drawn the same color and all moons are drawn the same color. However, it needs to be possible to reset the planet color or the moon color and the diagram should reflect that.

Bad Design



Note that I won't always be kind and draw those dependency lines.

1) Describe the problems with Bad Design

2) Design your own solution

Online Store with Coupons

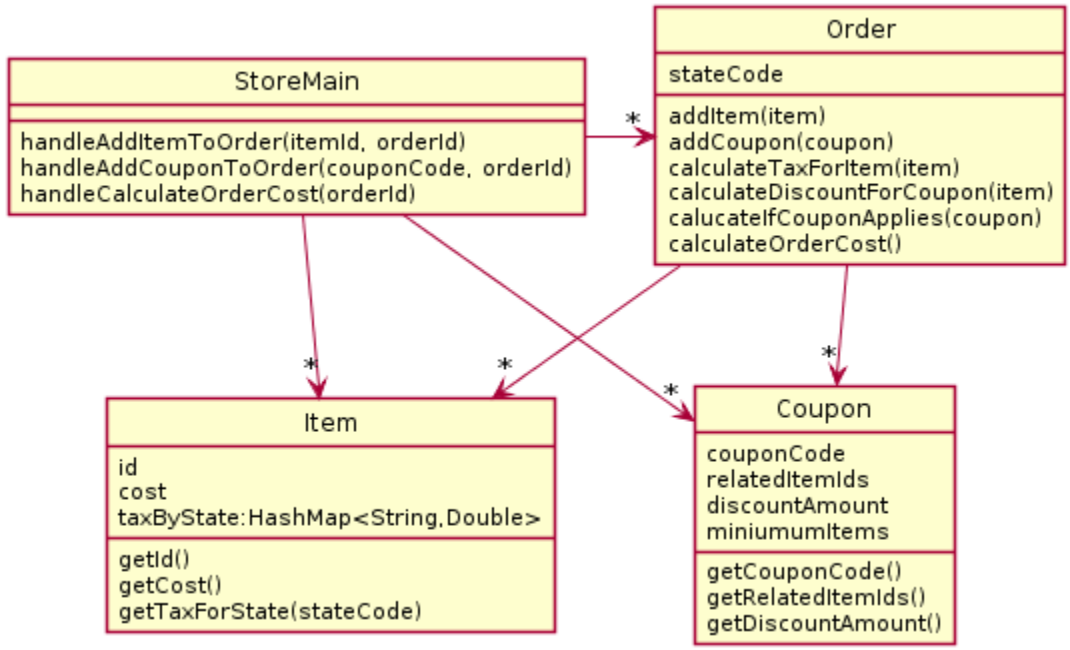
In a particular online store, customers can add both items and coupons to their order. Items are added by selecting a particular item ID. In this store, items are unique (i.e. an item cannot be added to more than one order, and each itemID is unique).

Coupons are added by entering a coupon code. A coupon looks like this "COUPON code: foobar Entitles you to $1 off PER ITEM of item75, item76, item77 if you order AT LEAST 2". The same coupon can be applied to multiple orders.

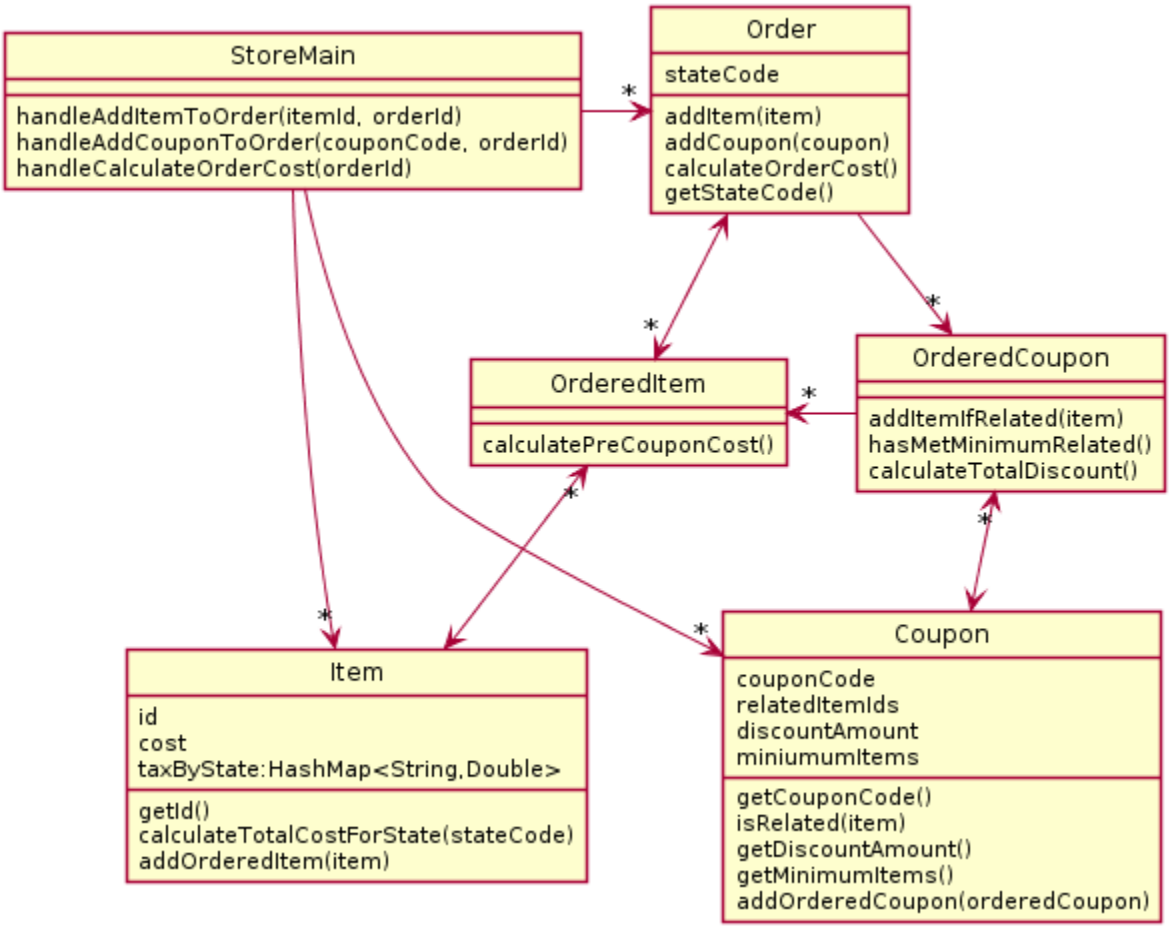
To calculate the total cost, the cost of each item must be added plus the tax (which can vary both by item and by state). Coupons must also be applied. Each coupon only applies to certain related items. Coupons require that a minimum number of related items be in the order before the coupon provides a discount. When a coupon does apply, it always provides a fixed amount off each related item and it does not affect tax.

You can assume all of these designs function correctly (that is, exclude principle 1 from your consideration).

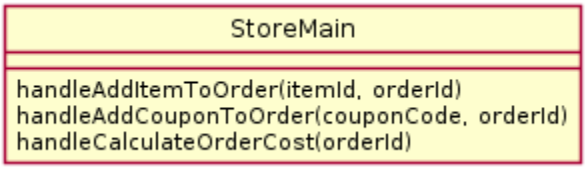
Design A



Design B



For your solution, we have provided a StoreMain to get you started. Feel free to omit any regular getter methods in your solution diagram as well.



1) Describe the problems with Design A and B

2) Design your own solution

Investment

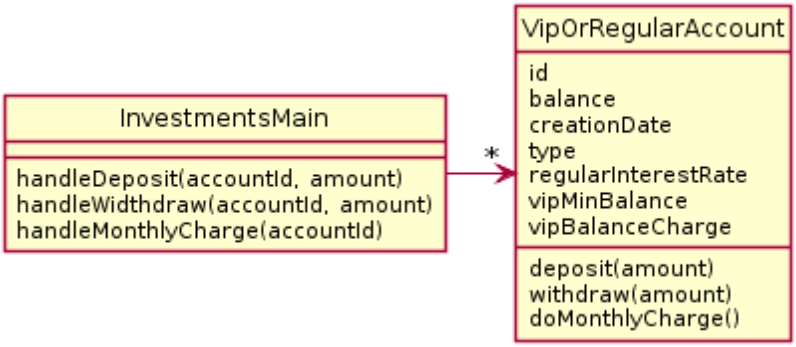
At investment company bank there are 2 kinds of accounts, regular accounts and VIP accounts. Both accounts allow deposits and withdrawals but each does something different monthly.

Every month, the regular account balance is decreased by a small percentage, stored as a negative interest rate which is set when the account is created. Every month, the VIP account is checked to see it above the minimum balance - if the balance is above minimum there is no fee, but if the account balance is below minimum a fixed balance fee is subtracted from the account. Both the minimum balance and the balance fee are set when the VIP account is created.

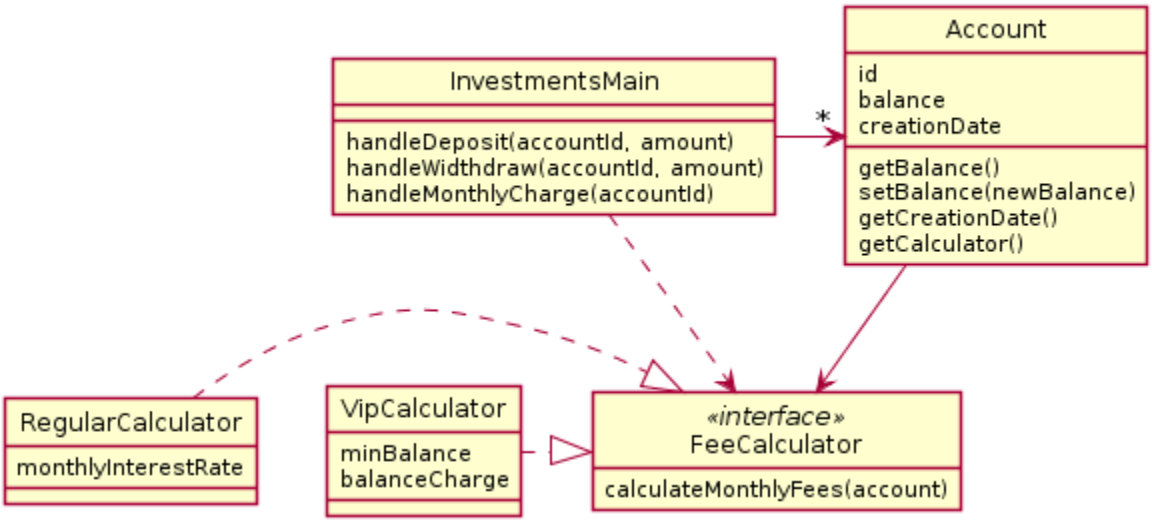
Both accounts also charge an annual fee, which is assessed as part of the fees during the month that the account was created. For that reason, the system must store the creation date for each account. Regular accounts charge $10 per year, VIP accounts charge $100/year.

You can assume that both these solutions function (i.e. exclude principle 1 from your consideration).

Design A



Design B



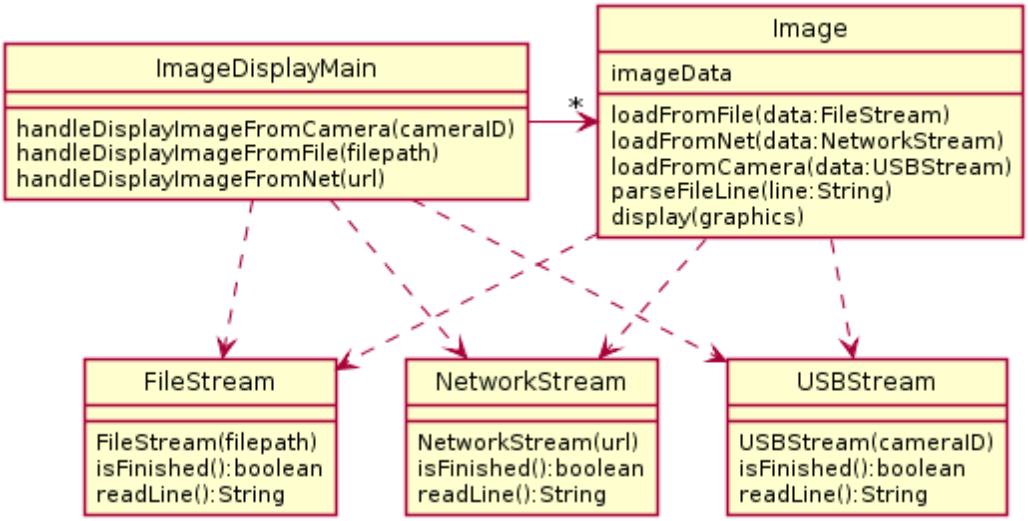
* Note this diagram does not show how RegularCalculator and VipCalculator objects get created. Don't worry about that issue.

1) Describe the problems with Design A and B

2) Design your own solution

Image Stream (in-class exercise SETechniques)

A particular Image class loads image files from a variety of sources and display them on the screen. The image sources can be a file, a network link, or a USB camera. All the various sources have similar functions, but because they are different types they each require a specialized function. Show how an improved approach using interfaces can remove the code duplication.



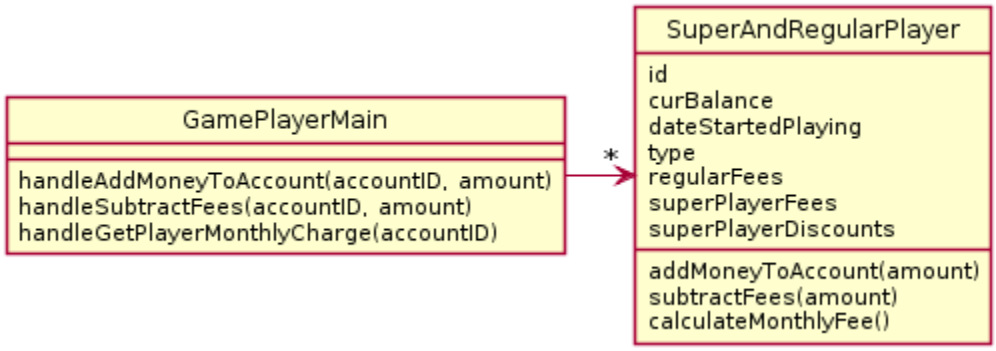
Game Player

There is an online gaming system that charges players according to their membership type. The system we want to build is just the system to calculate fees for the different players and then track their account balances. Every player has an account with a current balance that each player can add money to and we need to track the date they started playing (maybe for reports). There are two types of players: (1) Regular players that have a certain number of fees and (2) Super Players that have different fees, but there are some discounts that a Super Player can be eligible for that needs to be tracked.

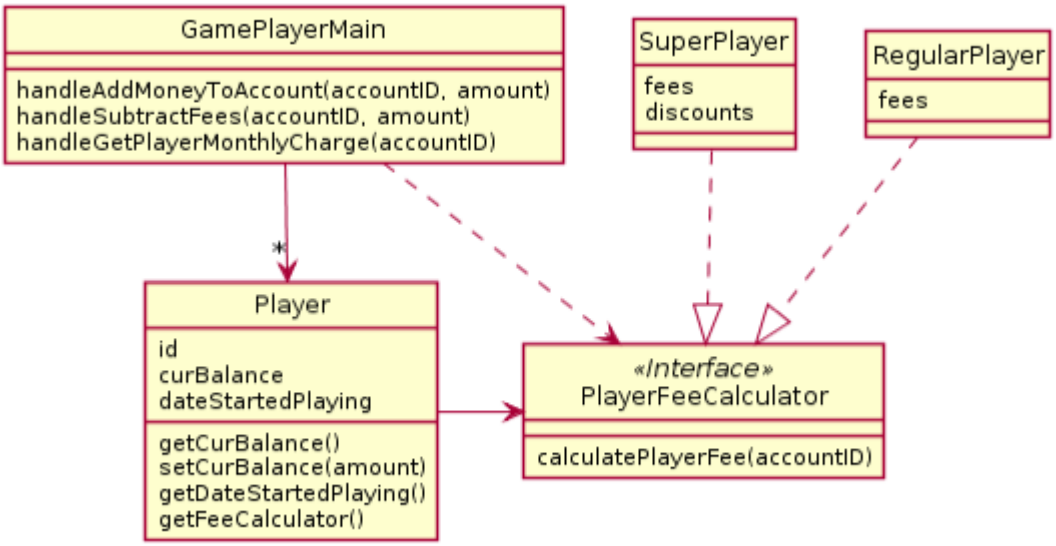
Every month, each player's account takes out the fees of the month from the current balance based on the player type and if there are any discounts for the player (Super Players only may have discounts). We want a way to calculate the total charge for the player based on the player type.

Here are 2 possible solutions. You can assume both of these designs function correctly (that is, exclude principle 1 from your consideration).

Design A



Design B

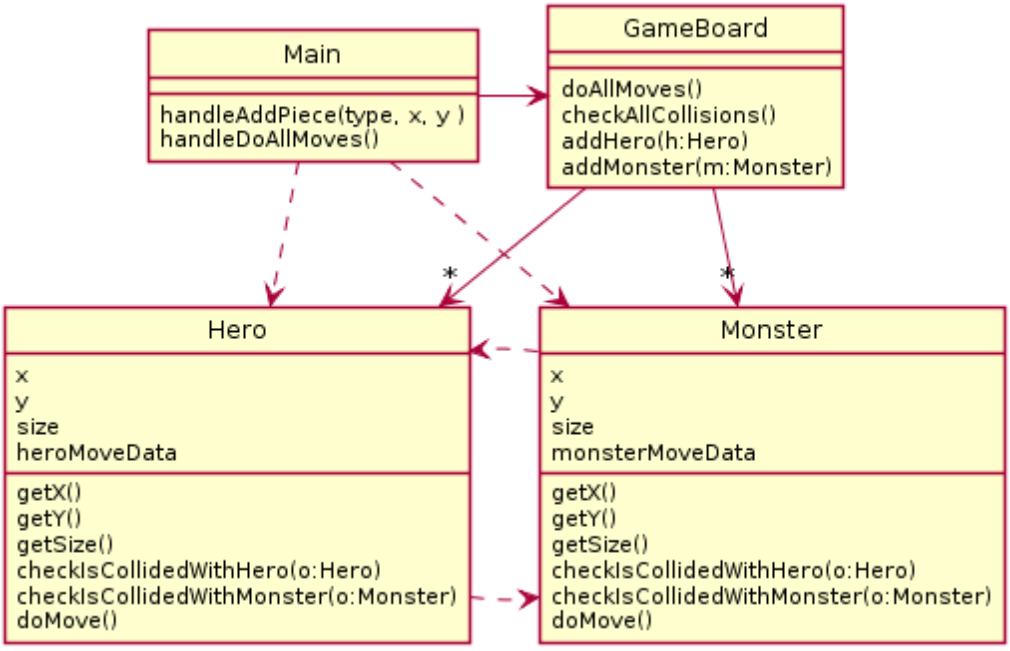


* Note this diagram does not show how RegularPlayer and SuperPlayer objects get created. Don't worry about that issue. You can assume there are parts to the system not being shown.

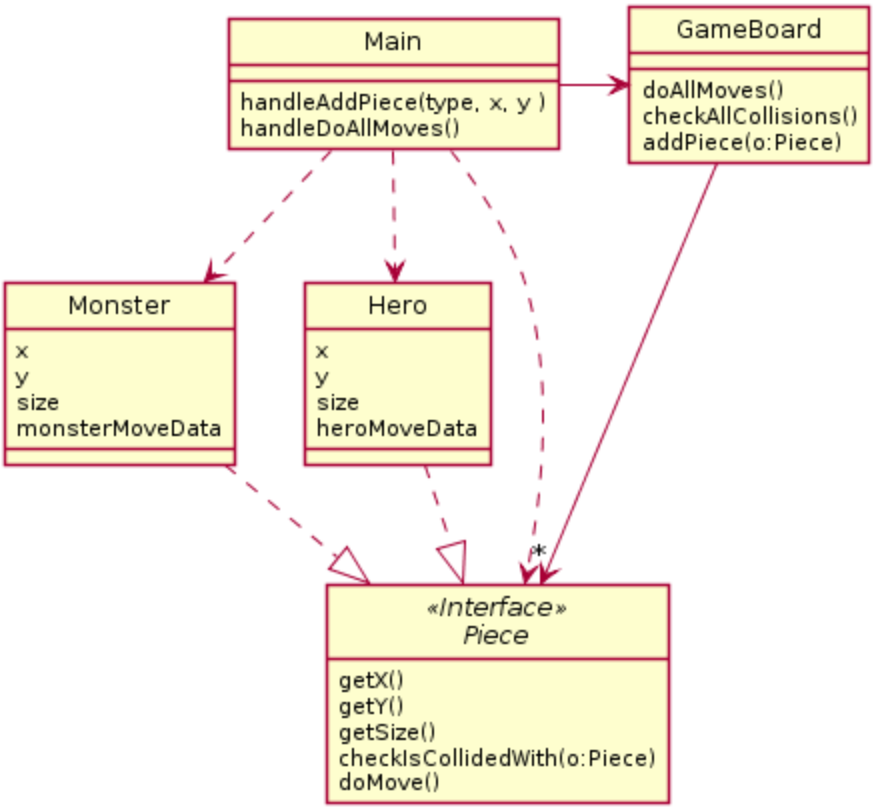
Video Game

In a particular video game, there are pieces called heros and monsters. Heros and monsters move differently from each other. The move is complex determination which requires specific data that is updated over the monster and hero's lifetime. Heros and monsters store different kind of move data. It is also necessary to detect if one piece is colliding with another piece - this functionality is pretty much the same regardless of if you are talking about heroes or monsters. The game must support: giving each piece the opportunity to move and update their position, and adding a new piece (either hero) or monster to the board.

Design A



Design B



1) Describe the problems with Design A that are fixed by Design B

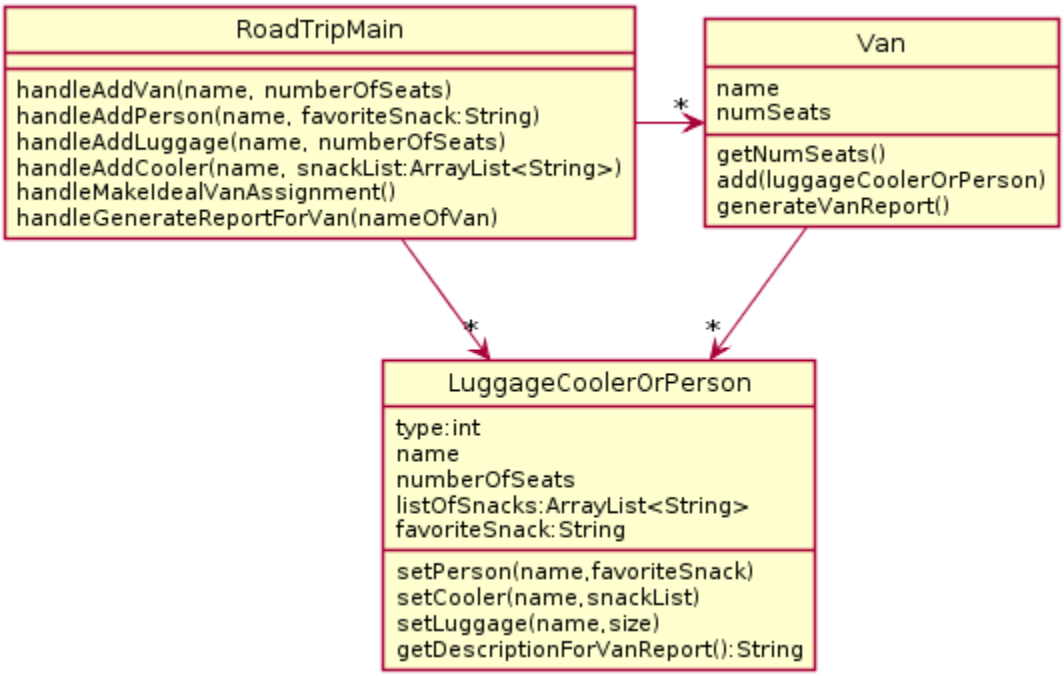
2) Describe the problems with Design B

3) Design your own solution

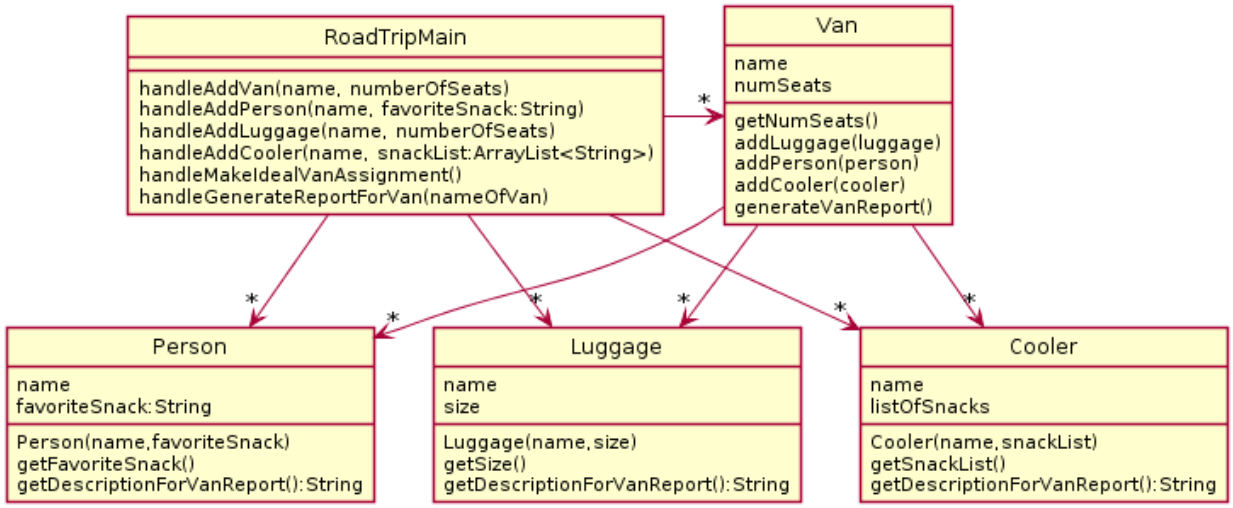
Road Trip

A particular software system is designed to fill vans to go on a road trip. Three different kinds of things can be stored in a van - luggage, coolers, and people - each has different data that guides how they can be assigned. Luggage can take up 0-2 seats, depending on its size. Coolers take up 1 seat and have a list of snacks that the cooler contains. People take up 1 seat and have a favorite snack - they must be placed in a van that has a cooler with that snack. The code in RoadTrip::handleMakeIdealVanAssignment() allocates all luggage, coolers, and people to vans (prior to that, nothing is assigned to vans). Once everything has been allocated, no new things can be added but a report can be printed for each van which displays a description of each of their contents.

Design A



Design B



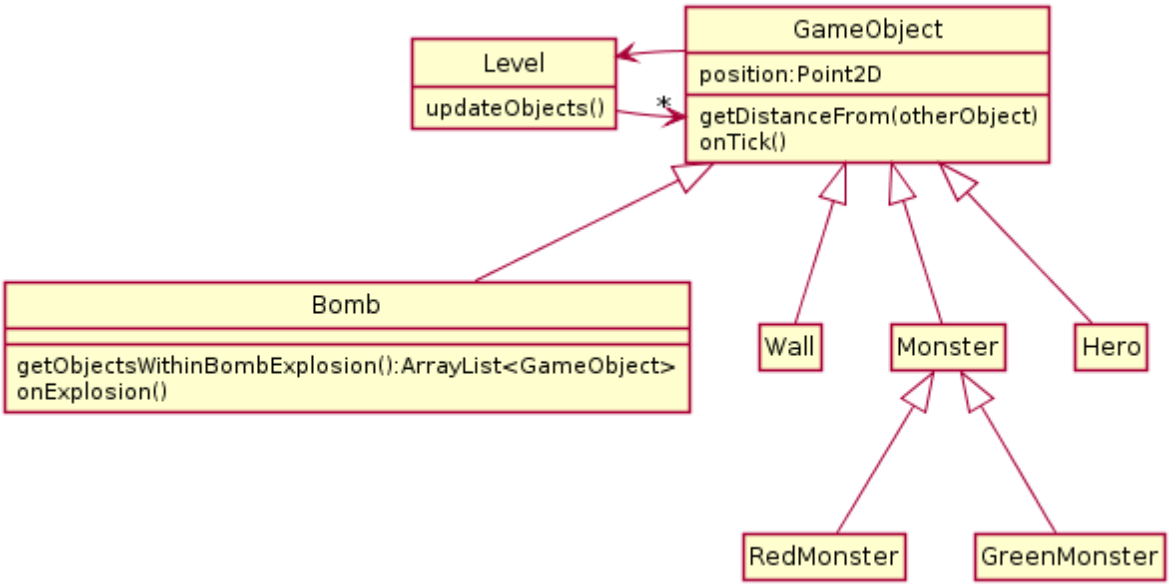
Which of these two designs has a problem with cohesion? Explain both whether the issue is high or low cohesion and how you can tell that the problem exists (1 or two sentences is all that's needed).

Which of these two designs has a problem with coupling? Explain both whether the issue is high or low coupling and how you can tell that the problem exists (1 or two sentences is all that's needed).

For the design which has a problem with coupling, you can use an interface to reduce the coupling in the system. Make a UML diagram of how this might be done. For convenience, you can omit all fields and methods from the classes your diagram EXCEPT the ones in the interface you add. Draw all lines in your diagram.

Bomberman (in-class exercise CollisionHandling)

In the Game Bomberman, everything has special behavior if caught in a bomb explosion. Heroes die and restart the level, monsters are killed and score points, walls are destroyed, and bombs explode themselves. In the design below, the Bomb class has an onExplosion method which handles this behavior and (you can assume) works correctly.



Inside of Bomb, the onTick() method makes a call to the onExplosion() when the bomb explodes. The code inside the onExplosion() method looks like this:

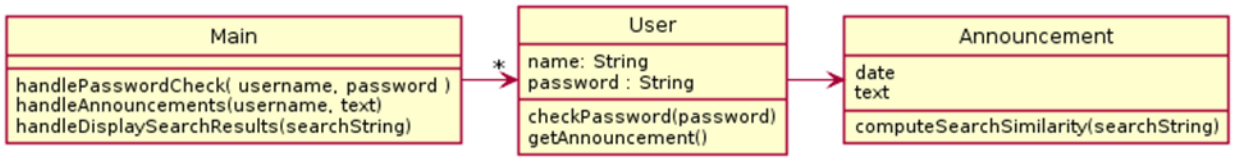
```
for(GameObject g : getObjectsWithinBombExplosion()) {

        if(g instanceof Bomb) {
        // bomb specific code
        }
        if(g instanceof Wall) {
        //wall specific code
        }
        //pattern continues...
}
```

Announcements

On a particular website, users log on with a username and password. Once logged on, they can make announcements that have their username, the current date and some text associated with them. The website also has a feature where you can search for announcements. Given a search string and announcement, an algorithm can compute a similarity rank in the range of 0-100. The results are then sorted in similarity rank order.

Solution A



Solution B