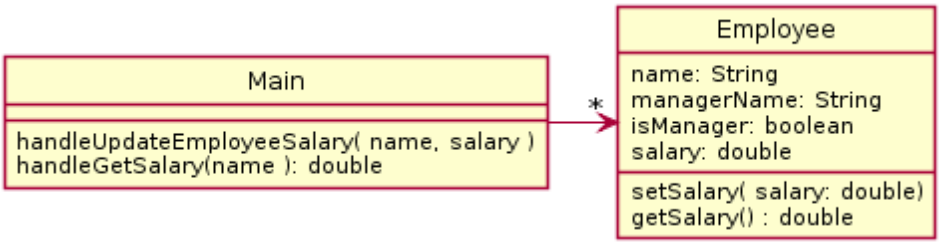
CSSE220: Design Problems 3-5 Solutions

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

All of these problems relate to Design Principle 3 – 5

Table of Contents

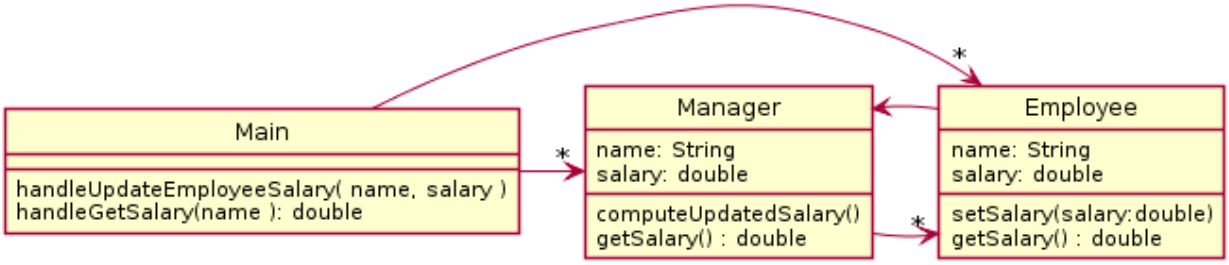
Employee Salary (in-class exercise SETechniques) - Solution



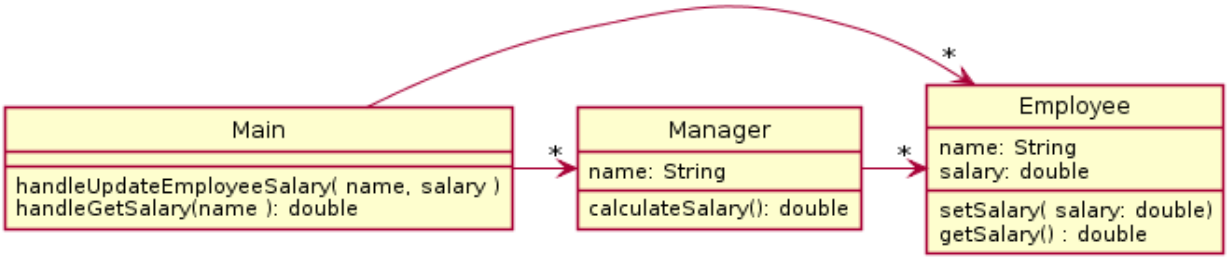
Problems With Design

3a. No class part should get too large - employee is too large

Better Design

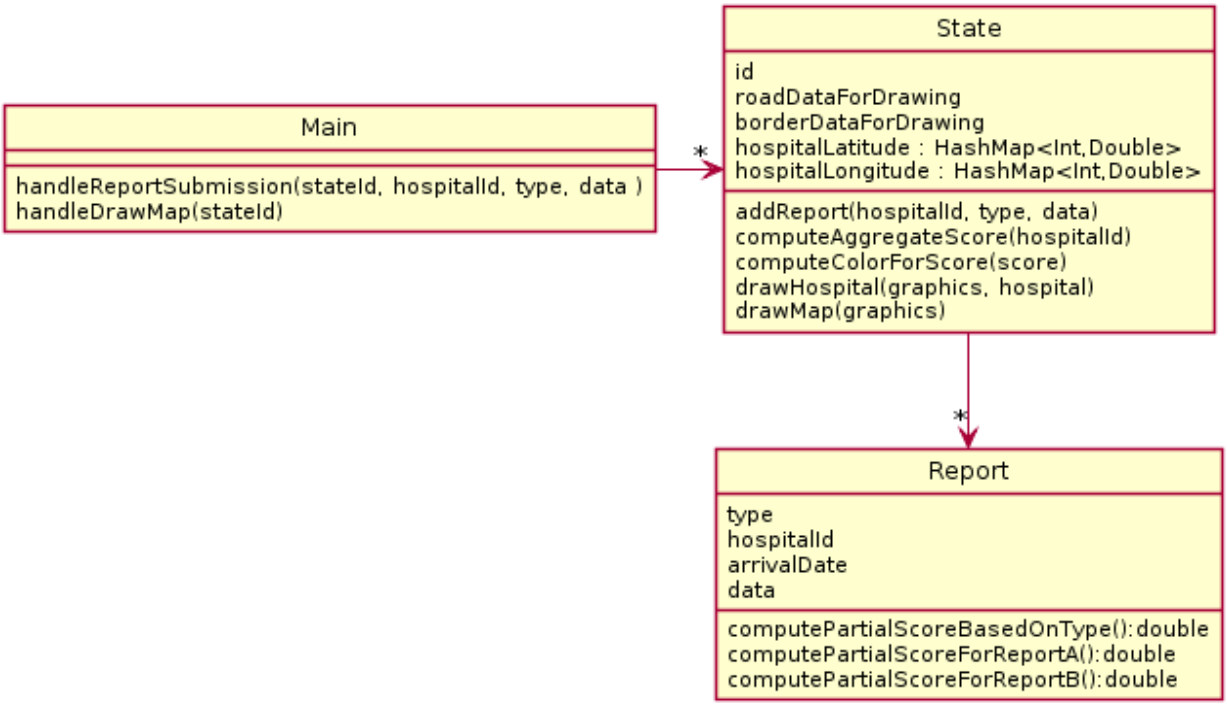


Even Better!



State Hospitals - Solution

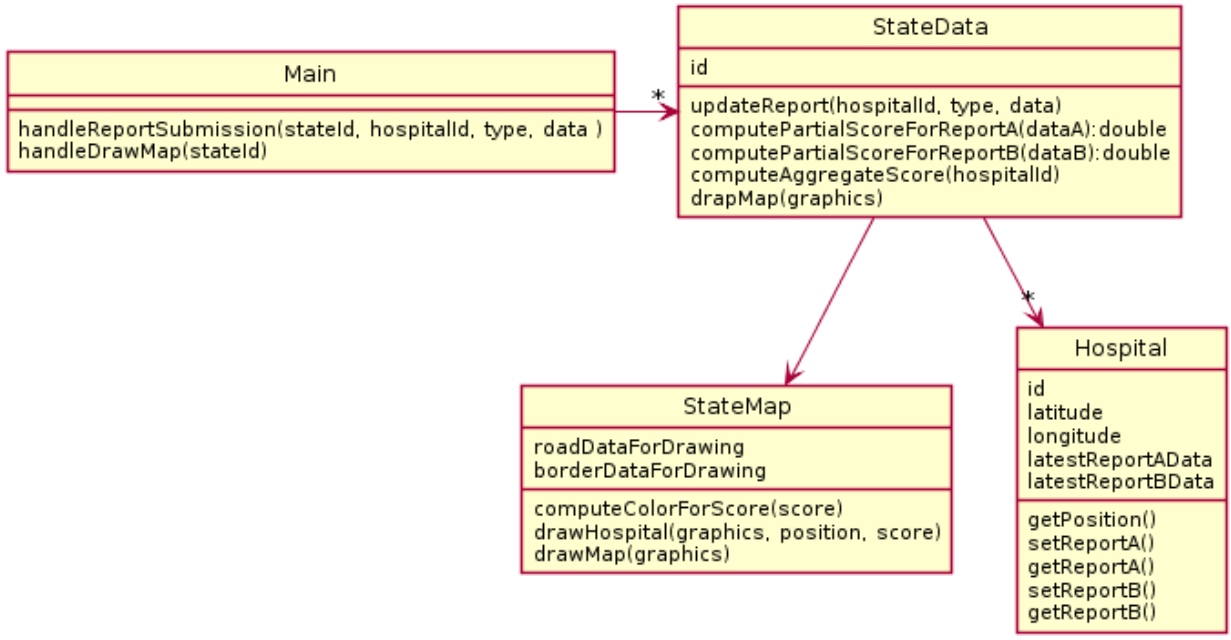
Design A



Problems With Design A

3b. Each class should have a single responsibility - State is both drawing and combining reports into score calculations

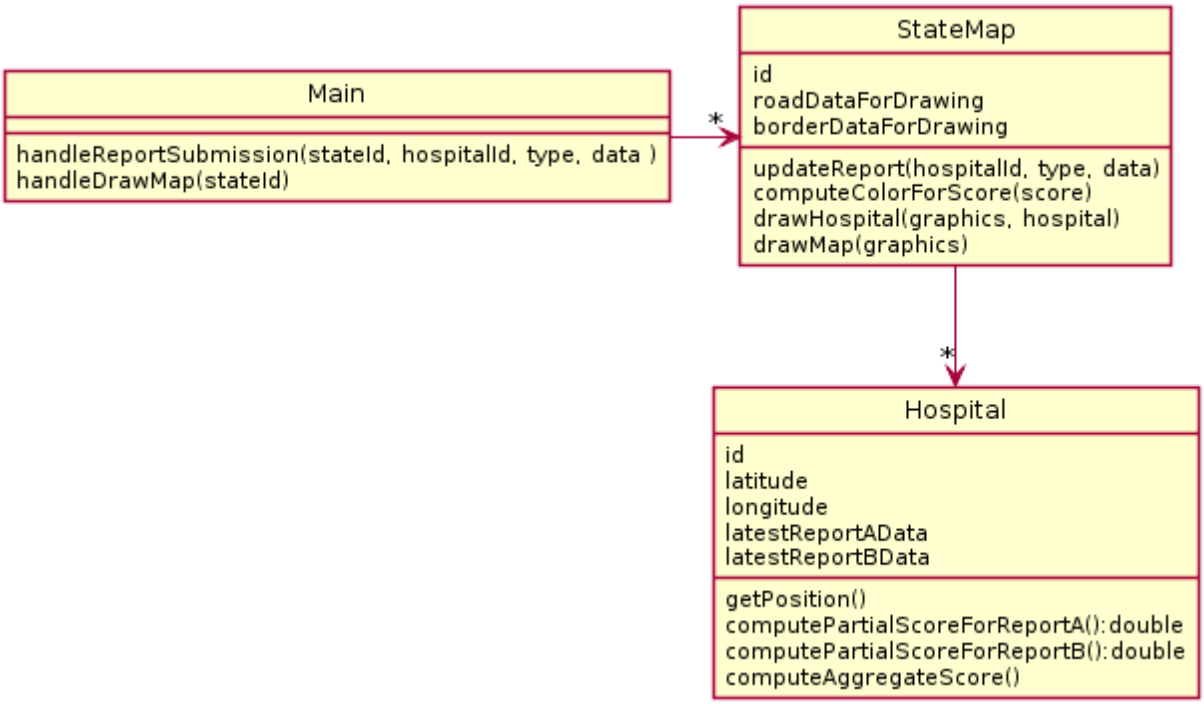
Design B



Problems With Design B

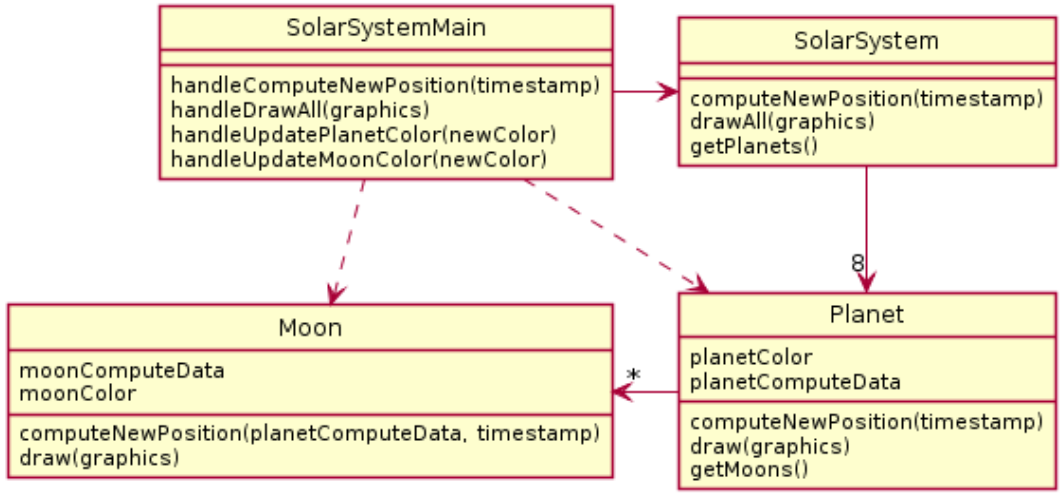
4b. Don’t have message trains - Hospital needs to be in charge of its own data

Solution



Keeping the StateData/StateMap distinction or having reports in their own classes like in A would also be acceptable answers.

Solar System (in-class exercise SETechniques) - Solution

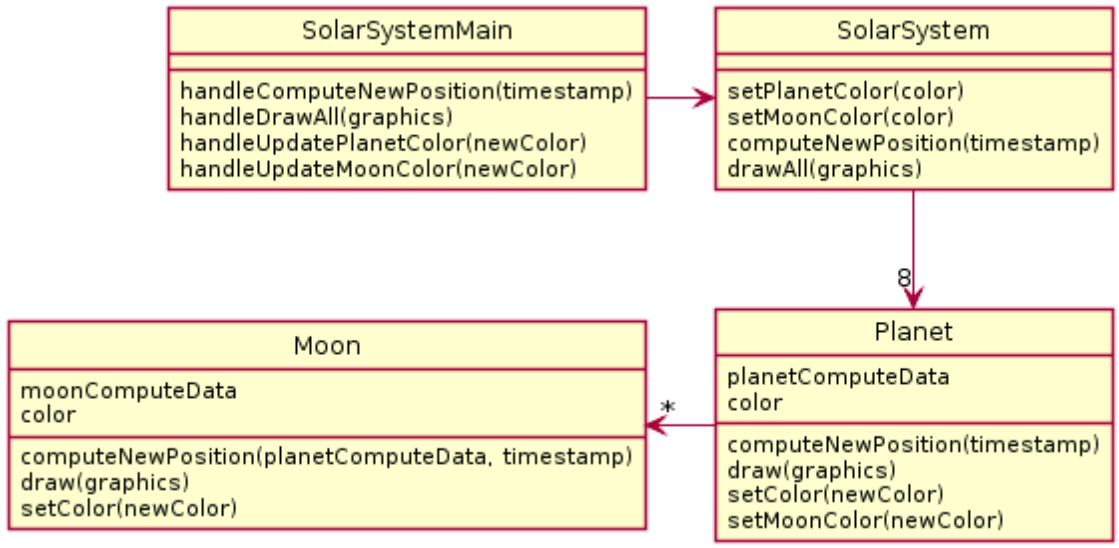


Problems With Design

4b. methodChain to update moon (ss.getPlanets().get(0).getMoons().get(0).setColor(color)

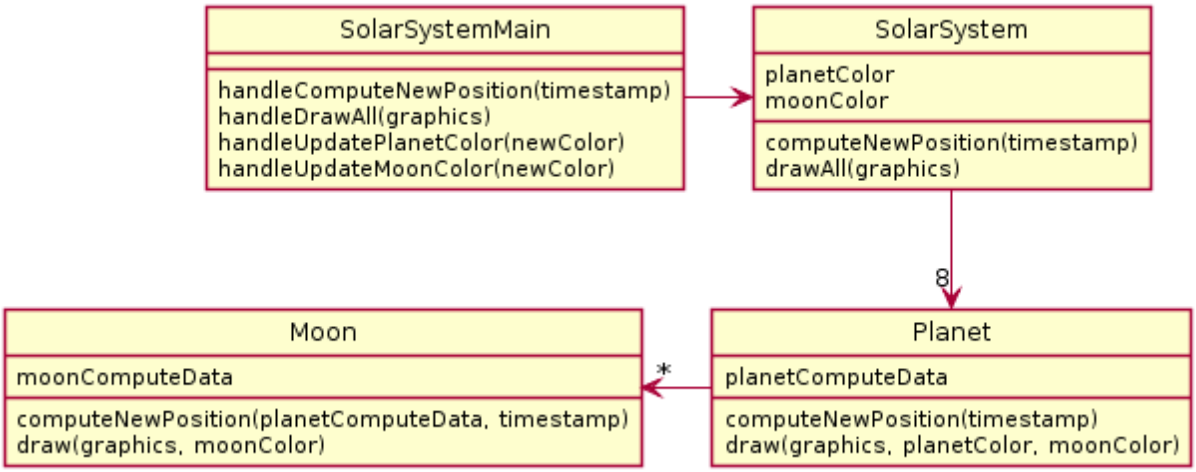
2c. planet and moon color are duplicated

Partial Solution



This is what you'll get if you mechanically follow our advice to propagate the method into the first class of the chain.

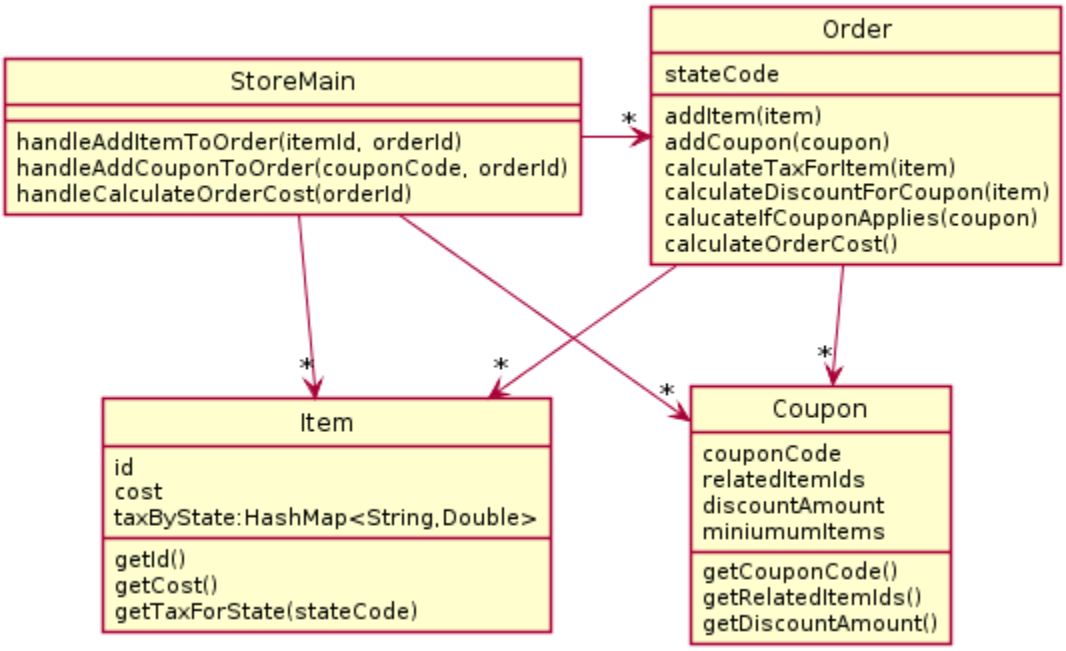
Better Solution



There are other ways to solve this problem too.

Online Store with Coupons - Solution

Design A



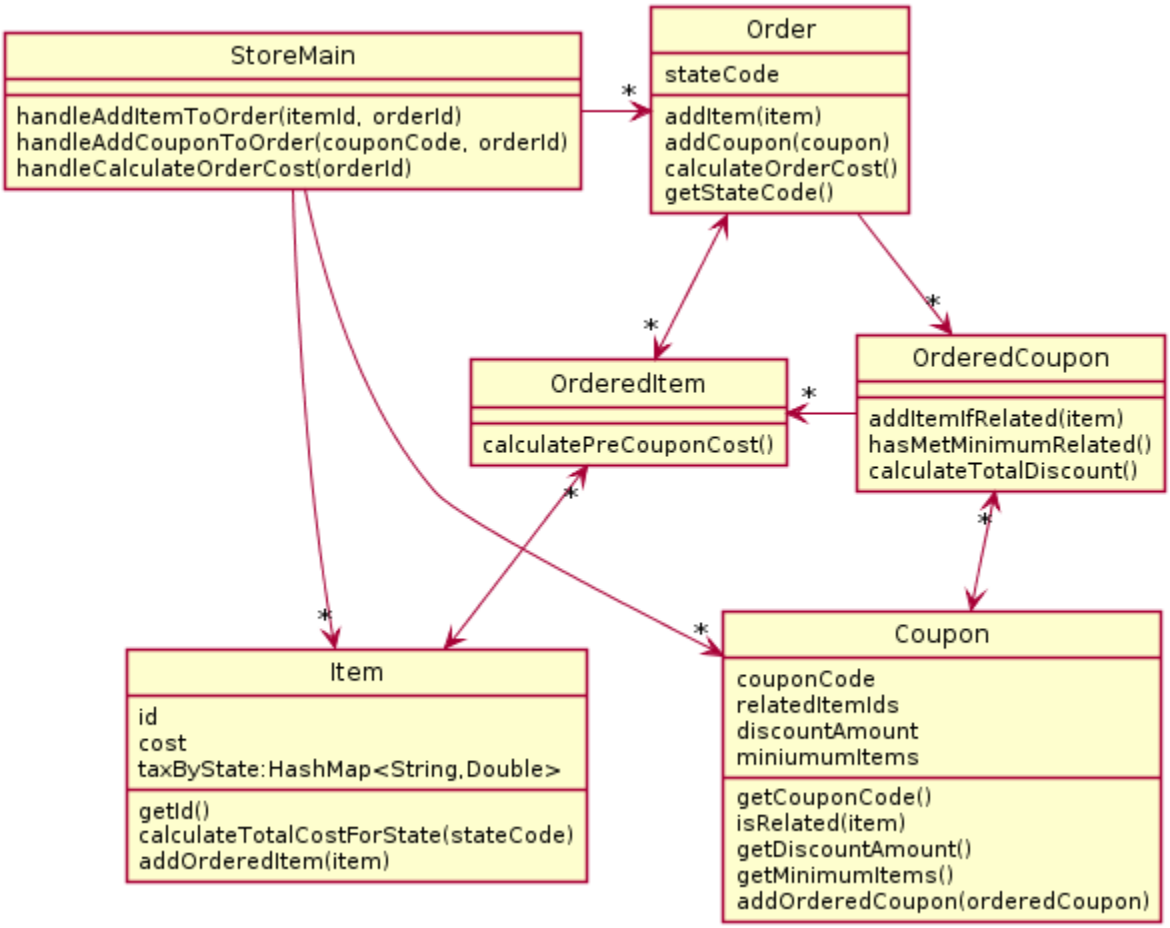
Problems With Design A

3a. No class or part should get too large - order is doing too much

4a. Tell don't ask - Item and Coupon have ask methods

3b. Each class should have intelligent behaviors - Item and Coupon are data classes

Design B

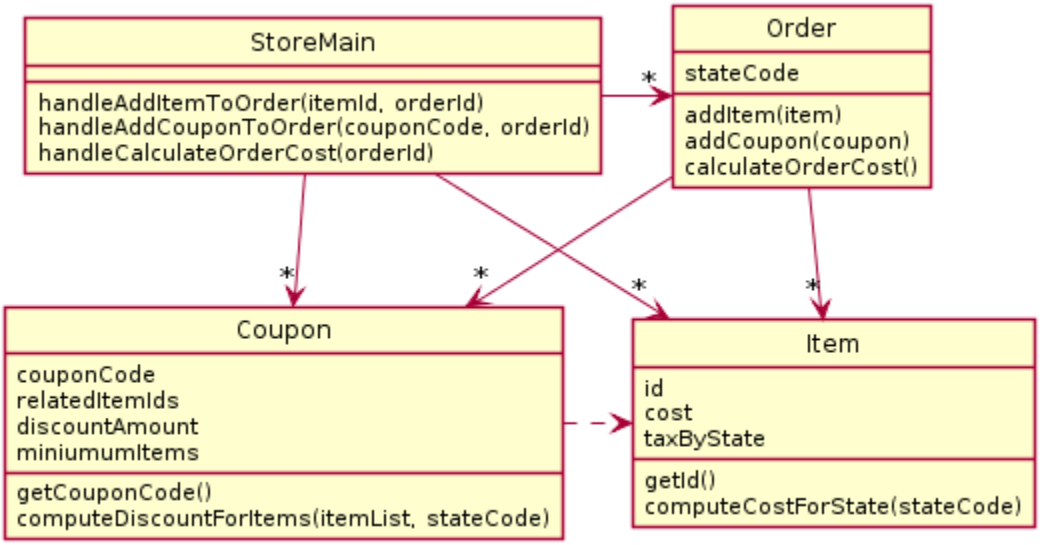


Problems with Design B

4. Minimize dependencies - too much interdependency

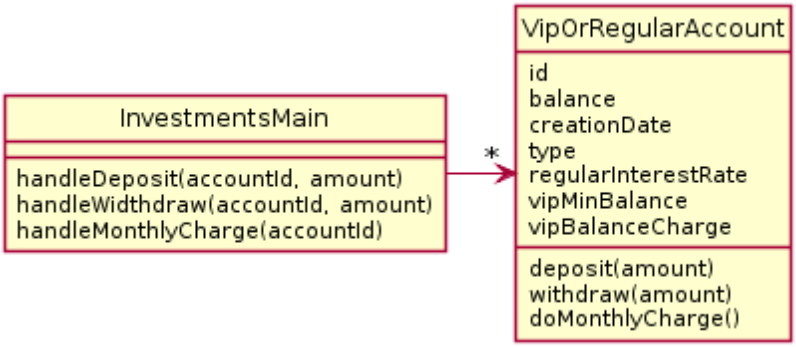
3. Functionality should be distributed efficiently - Maybe you could say that orderedItem/orderedCoupon are not cohesive

Solution


StoreMain
handleAddItemToOrder(itemId, orderId)
handleAddCouponToOrder(couponCode, orderId)
handleCalculateOrderCost(orderId)
Order
stateCode
addItem(item)
addCoupon(coupon)
calculateOrderCost()
Coupon
couponCode
relatedItemIds
discountAmount
miniumumItems
getCouponCode()
computeDiscountForItems(itemList, stateCode)
Item
id
cost
taxByState
getId()
computeCostForState(stateCode)
*

Investment - Solution

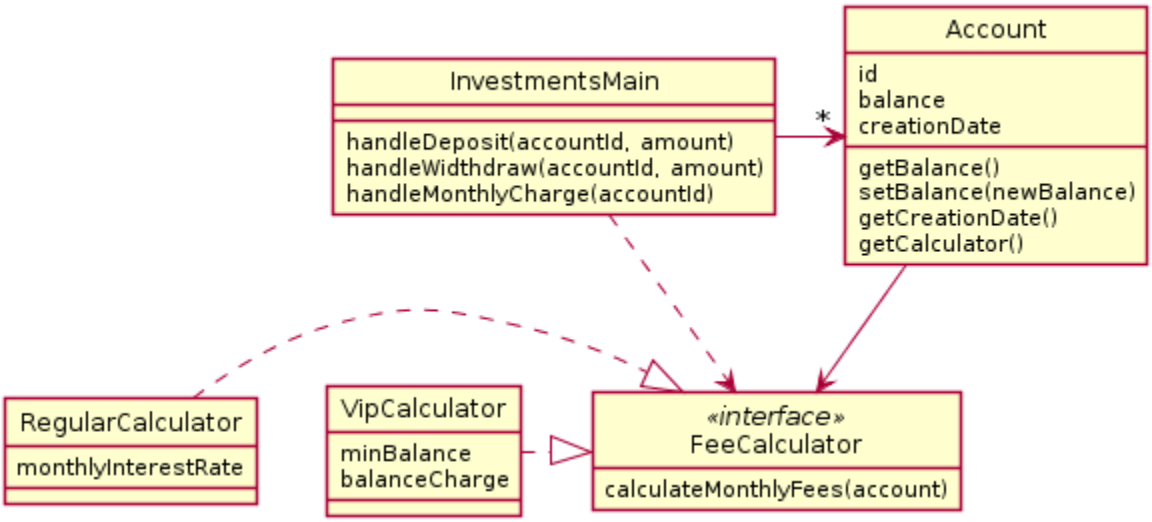
Design A



Problems with Design A

3b. Each class should have a single responsibility - VipOrRegularAccount has too many concepts it represents

Solution Design B



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

Problems with Design B

4a. Tell don't ask - Account has too many ask methods

2b. Classes should have intelligent behaviors - Account is a data class

Solution

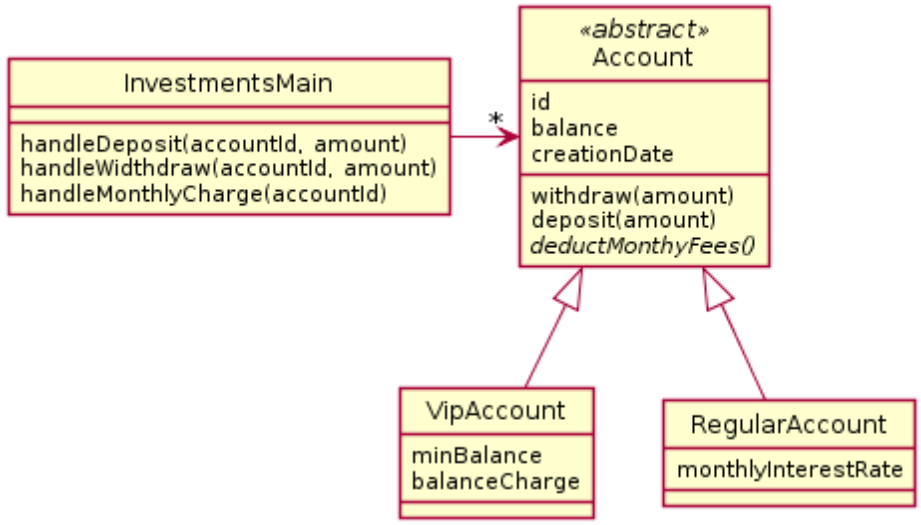
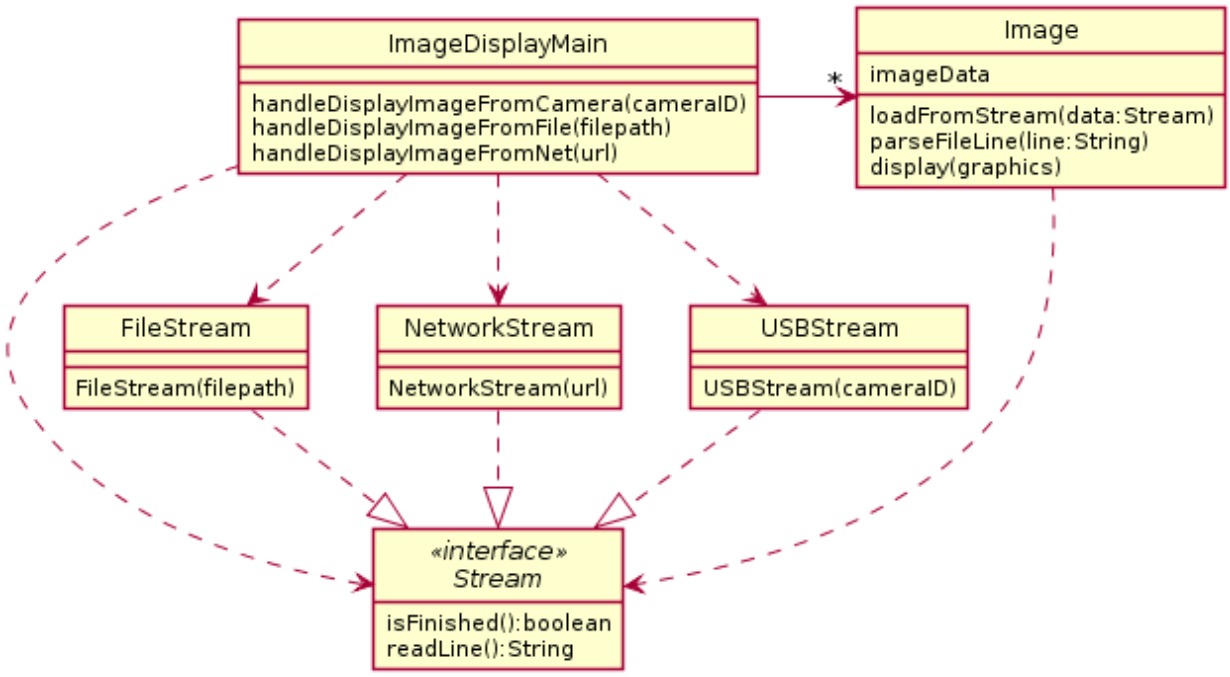

InvestmentsMain
handleDeposit(accountId, amount)
handleWidthdraw(accountId, amount)
handleMonthlyCharge(accountId)
*
«abstract»
Account
id
balance
creationDate
withdraw(amount)
deposit(amount)
deductMonthyFees()
VipAccount
minBalance
balanceCharge
RegularAccount
monthlyInterestRate

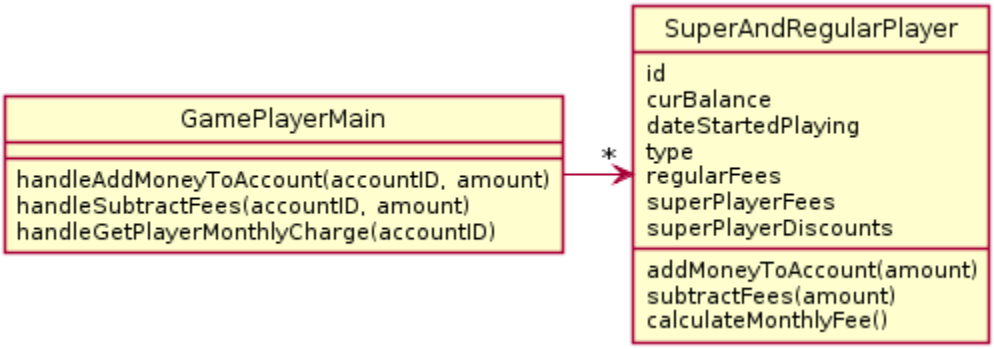
Image Stream (in-class exercise SETechniques) - Solution



Note that as is tradition, we omitted methods in the {File,Network,USB}Stream classes that are implied by the interface - although those methods are of course still there.

Game Player - Solution

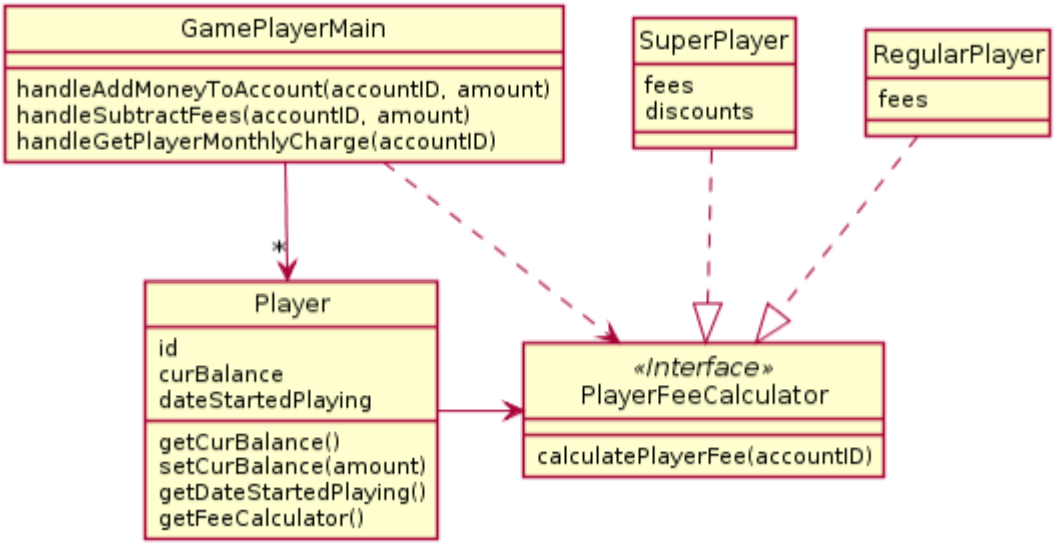
Design A



Problems with Design A

3b. Each class should have a single responsibility - SuperAndRegularPlayer is trying to do too much and does not represent a single concept.

Design B

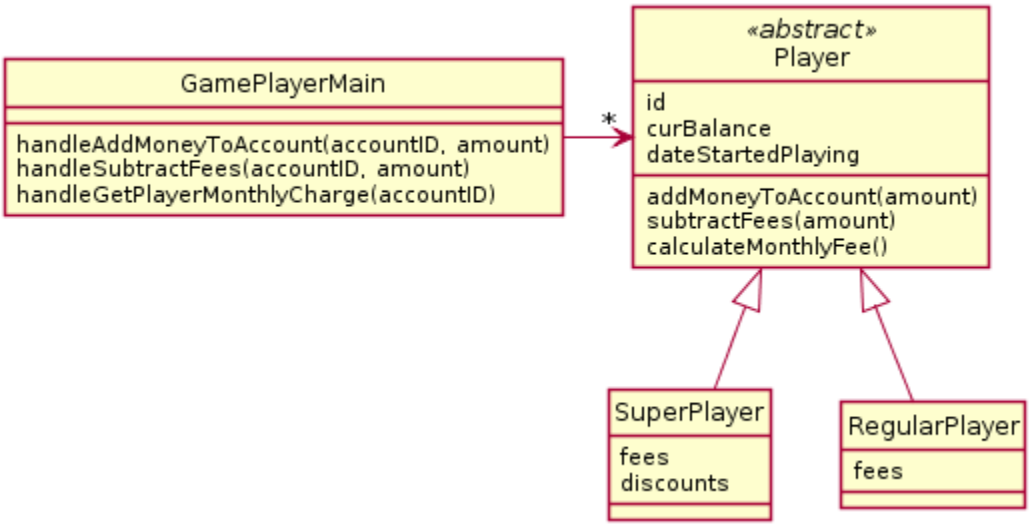


Problems with Design B

4a. Tell don't ask - Player has too many ask methods

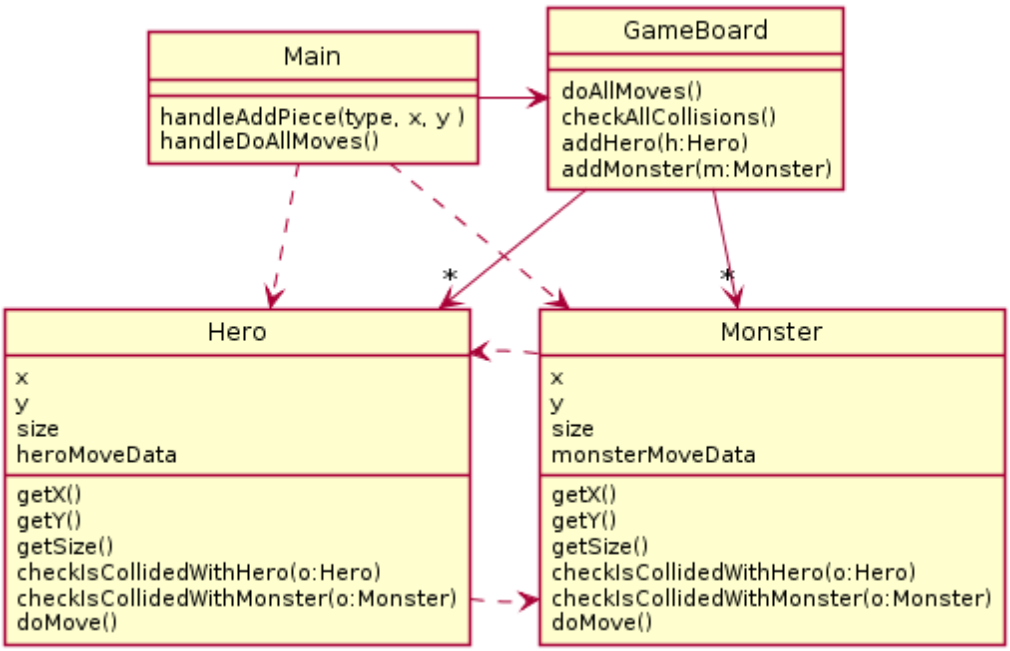
2b. Classes should have intelligent behaviors - Player is a data class and could do more in terms of functionality

Solution


GamePlayerMain
handleAddMoneyToAccount(accountID, amount)
handleSubtractFees(accountID, amount)
handleGetPlayerMonthlyCharge(accountID)
*
«abstract»
Player
id
curBalance
dateStartedPlaying
addMoneyToAccount(amount)
subtractFees(amount)
calculateMonthlyFee()
SuperPlayer
fees
discounts
RegularPlayer
fees

Video Game - Solution

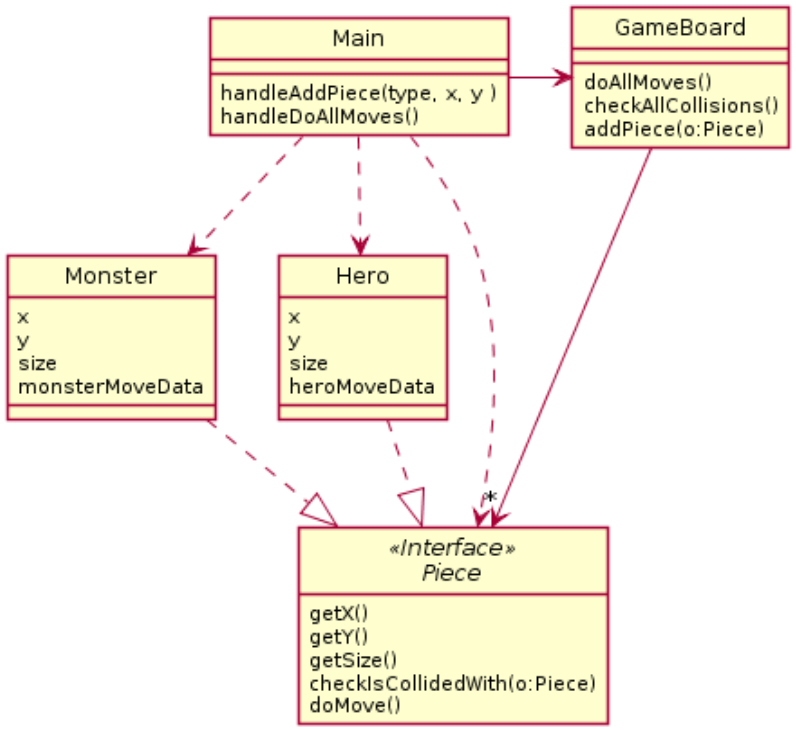
Design A



Problems With Design A

5b. Classes with similar features should be given common interfaces

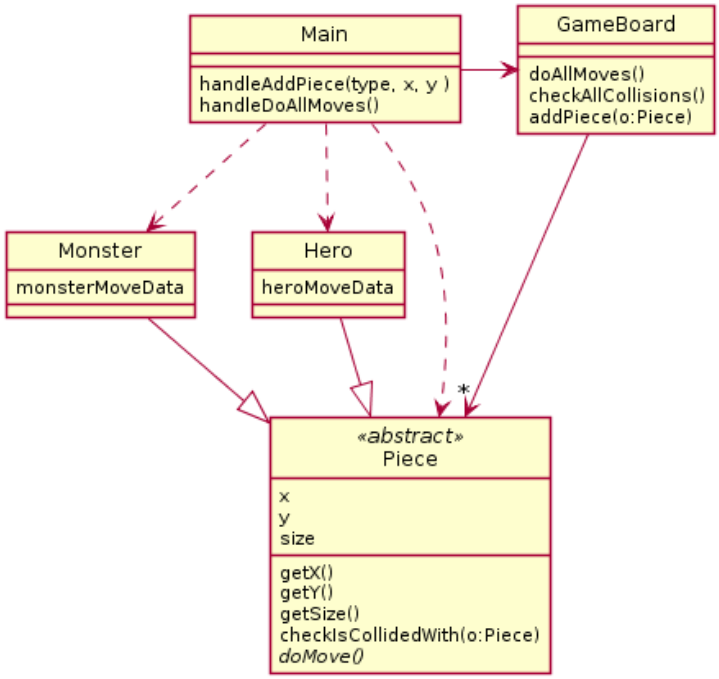
Design B



Problems With Design B

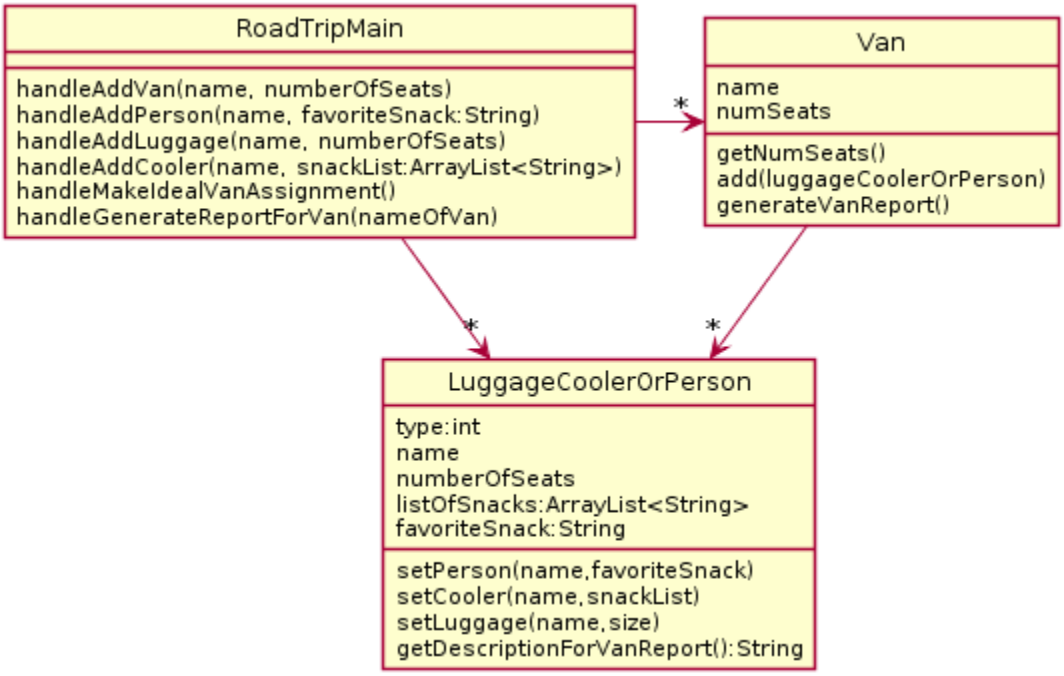
1c. Data should not be duplicated

Solution



Road Trip - Solution

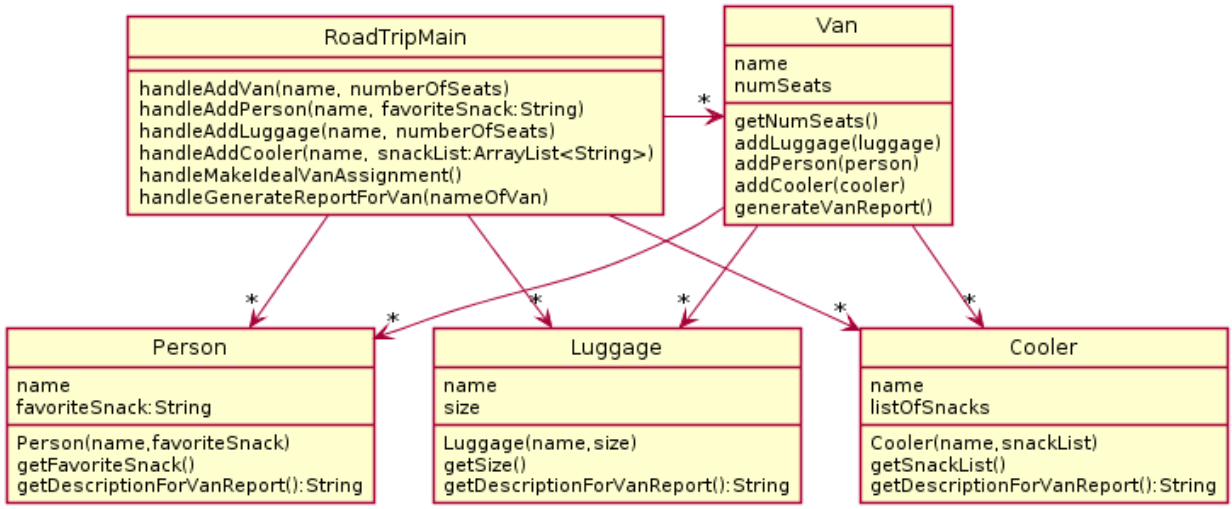
Design A



Problems With Design A

There is low cohesion in this solution - LuggageCoolerOrPerson is responsible for doing many loosely related things that could be broken into smaller tasks and different classes.

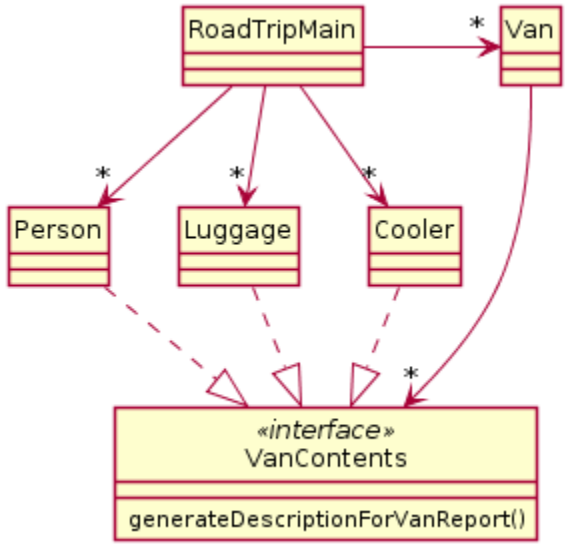
Design B



Problems With Design A

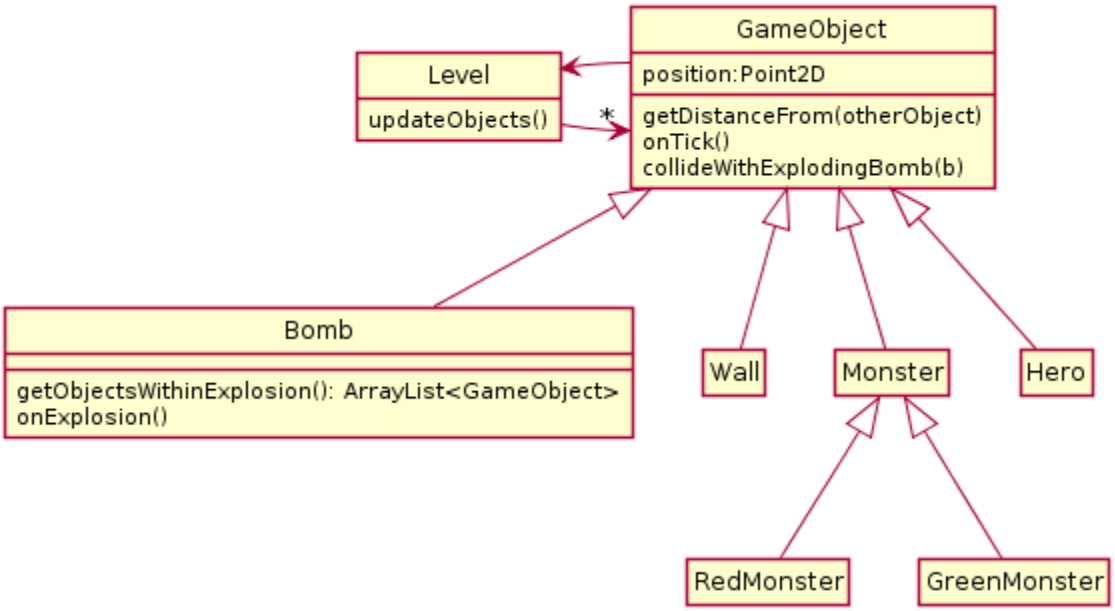
There is high coupling in this solution, each of the classes has dependency with each of the others.

Solution


RoadTripMain
*
Van
*
*
*
Person
Luggage
Cooler
*
«interface»
VanContents
generateDescriptionForVanReport()

Bomberman (in-class exercise CollisionHandling) - Solution

One solution



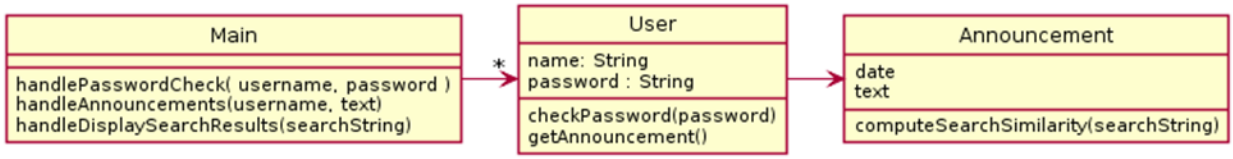
Each subclass will process the effects of a collision inside of the collideWithExplodingBomb method. No instanceof is required. Additionally, the code to handle collisions can be done inside onExplosion() as before:

```
for(GameObject g : getObjectsWithinBombExplosion()) {
  g.collideWithExplodingBomb(this);
}
```

Each subclass will deal with handling the class specific effects of being hit by a bomb

Announcements - Solution

Solution A



Problems With A

Looking at Announcement, it seems odd that such an important object is this system is just seemingly a sub-member of User. What problems does that cause?

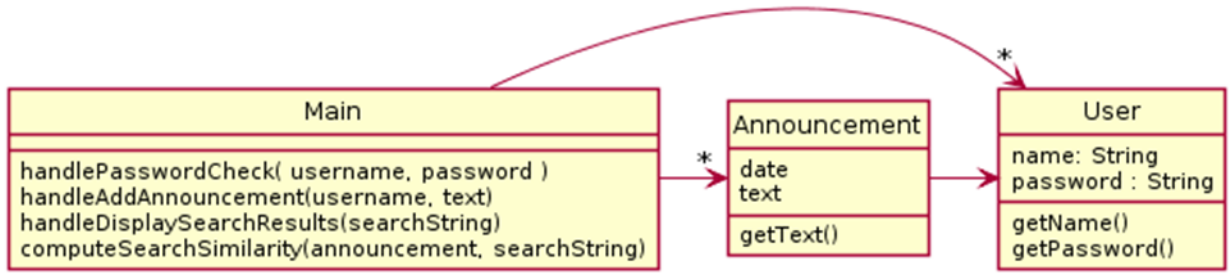
The Problems

1a. Can't have multiple Announcements per user

1b. No obvious way for Main to call Announcement's similarity computation

4b. (minor) message chain to compute similarity, assuming there was a way

Solution B



Problems With B

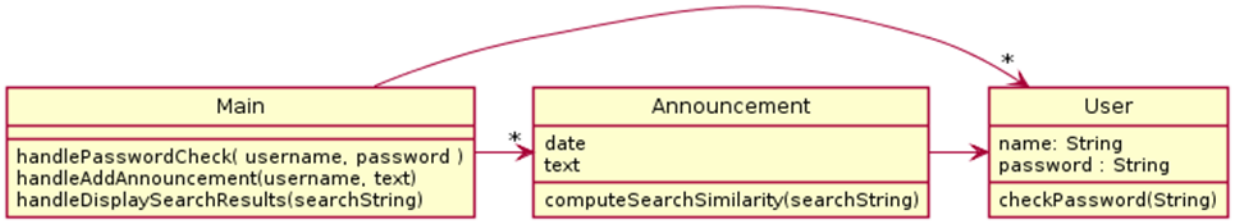
As is often the case, I start by noticing that both Announcement and User are really just data classes. (Equivalently - who is calling all those ask methods in Announcement and User?) Isn't there some functionality that could be in them to make them work better? Oh wait...why is main doing similarity computation?

The Problems

3a. Main is doing too much - it should not handle similarity computation OR Announcement is not doing enough - is a data class

3a. (minor) user should handle password check

Final Solution



This is a good example of the fact that a good solution doesn't need to be complex. Just put the functionality where it belongs and you'll go far.