CSSE220: Example Design Problem Solutions

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

This document includes example problems, plus my solutions and commentary. 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

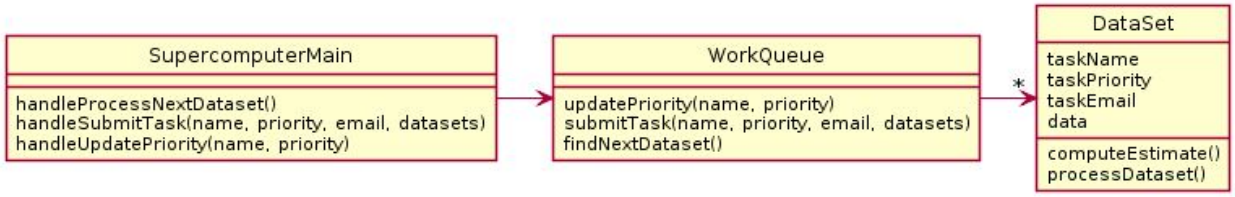
List of Design Problems:

1. Supercomputer
2. Announcements
3. State Hospitals
4. VideoGame

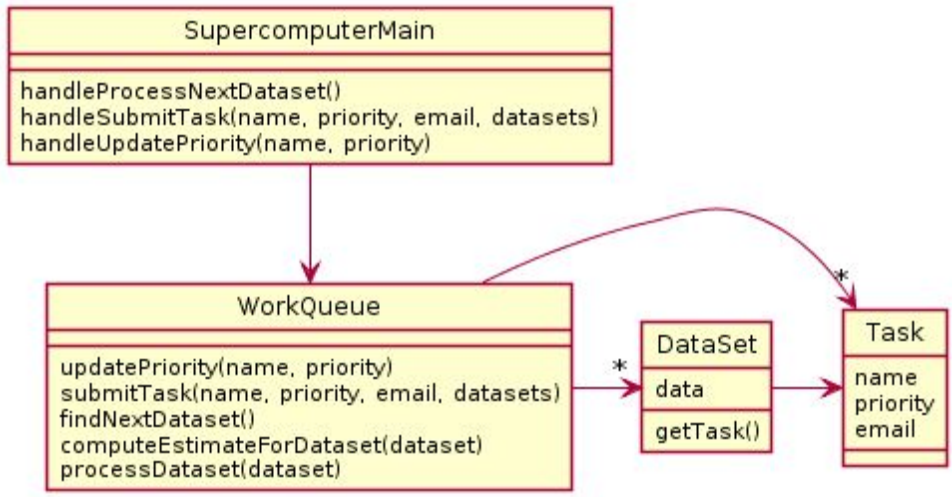
Supercomputer

The astronomy department maintains a supercomputer that everyone wants to use. Astronomers submit tasks to the supercomputer that consist of a series of datasets that must be processed independently. Each task includes a name, priority, and an email address where the results should be sent. Each dataset just has data. Given a dataset, it is possible to compute an estimate of how long it will take to run. The department agrees that the supercomputer should process datasets in priority order, and when priority is equal the supercomputer should select the dataset with the smallest estimate runtime. However, it must be possible to change a task's priority after it has been submitted.

Solution A



Solution B



Solution

Problems With A

Commentary
The first thing I notice about this problem is that task is not represented. That's not a huge problem in it's own right, but it makes me look into what happened to the data in Task. Oh hey, it seems to be Dataset - would that cause a bunch of duplication? Yes it does.

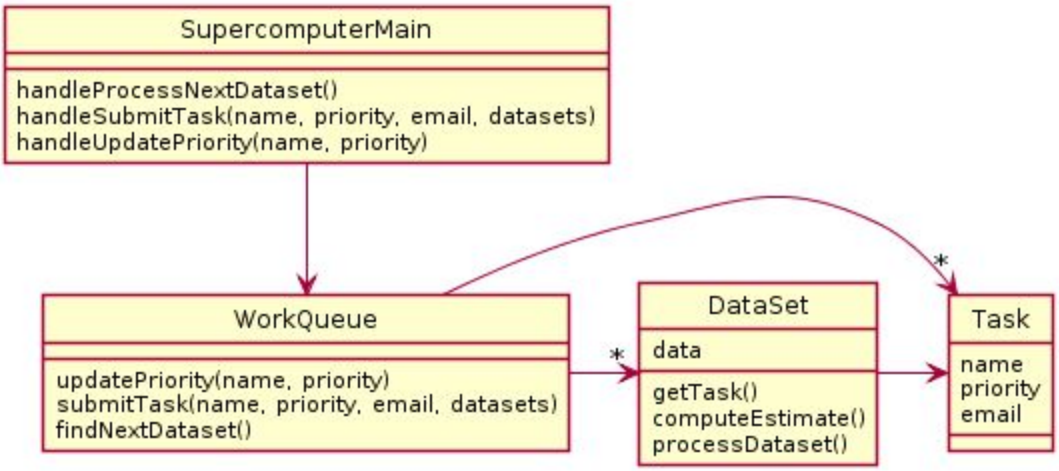
The Problems
1c. This design has duplication of the taskName, priority, and email. As well as being bad in it's own right, this causes updating priority to require searching all datasets.
2a. No task

Problems With B

Commentary
The first thing I notice is how small Task and dataset look - they are data classes. Is there any functionality in the system that could logically migrate into them?
Hmmm...computeEstimateForDataset or processDataset could, and they are both in a class that is already large.

The Problems
3a/b. WorkQueue is doing too much - DataSet should logically be responsible for computing estimates and processing.

Final Solution

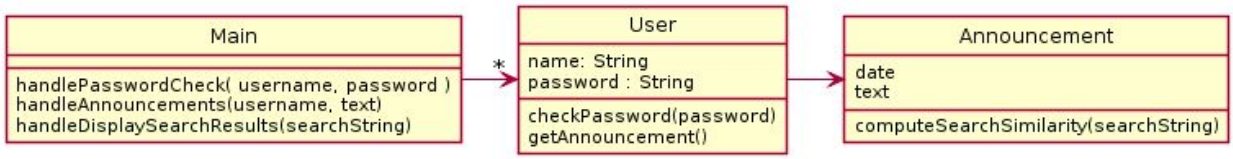


I could live with Tasks containing datasets or even DataSet not existing at all and just being handled through task. Both have their pros and cons.

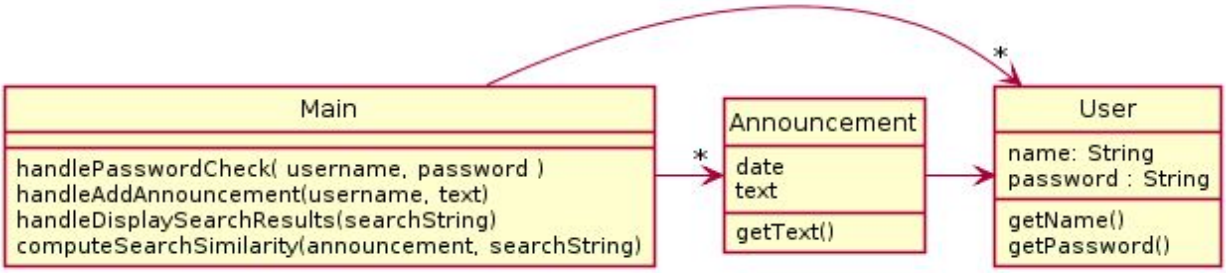
Announcements

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



Solution

Problems With A

Commentary
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?

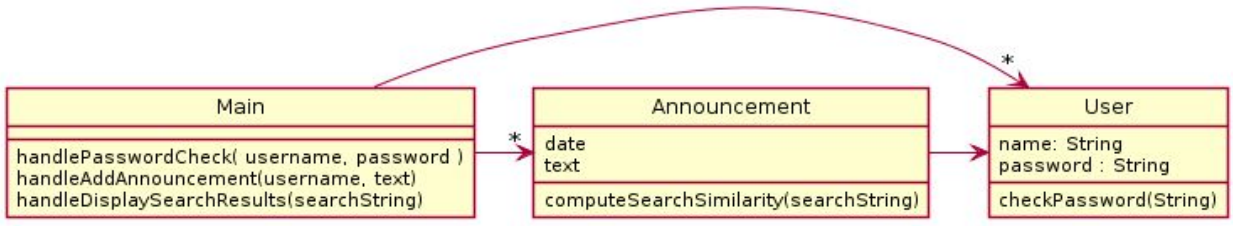
The Problems
1a. Can't have multiple Announcements per user
1b. No obvious way for Main to call Announcement's similarity computation
4b. (minor) message chain to compute similarity, assuming there was a way

Problems With B

Commentary
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

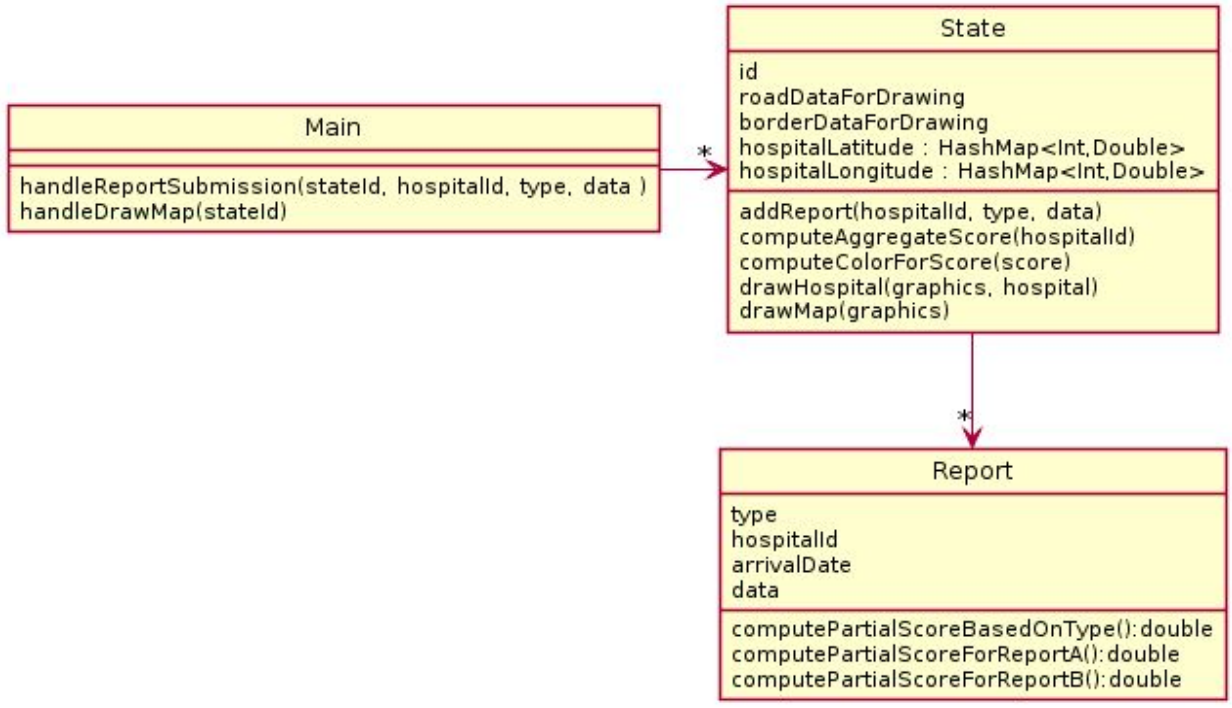


This is a good example of the fact that a good solution doesn't need to be complex. Just put the functionality where it belongs and you'll go far.

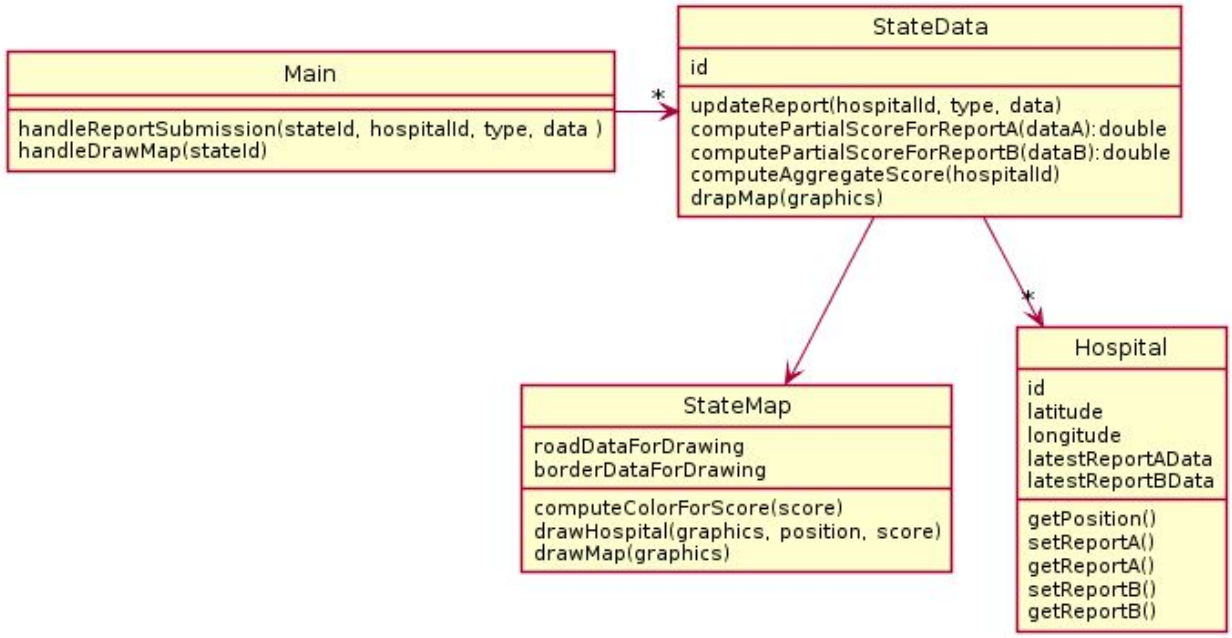
State Hospitals

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 two 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.

Solution A



Solution B


Main
handleReportSubmission(stateId, hospitalId, type, data)
handleDrawMap(stateId)
*
StateData
id
updateReport(hospitalId, type, data)
computePartialScoreForReportA(dataA):double
computePartialScoreForReportB(dataB):double
computeAggregateScore(hospitalId)
drapMap(graphics)
*
StateMap
roadDataForDrawing
borderDataForDrawing
computeColorForScore(score)
drawHospital(graphics, position, score)
drawMap(graphics)
Hospital
id
latitude
longitude
latestReportAData
latestReportBData
getPosition()
setReportA()
getReportA()
setReportB()
getReportB()

Solution

Problems with A

Commentary
First I notice that Hospital is not represented, so I look for where it's data and functionality got stored. Part of it went to report - OK, maybe a bit odd but seems fine. Part of it went to State, which also seems to have all the drawing functionality. It's weird - State is 50% drawing, 50% processing reports? That's not cohesive.

The Problems
3b. State is doing 2 different things - drawing and dealing with combining reports into the aggregate score calculations. A less clear but ok explanation would be - State is doing too much.
2a. Hospital is not represented. (minor)

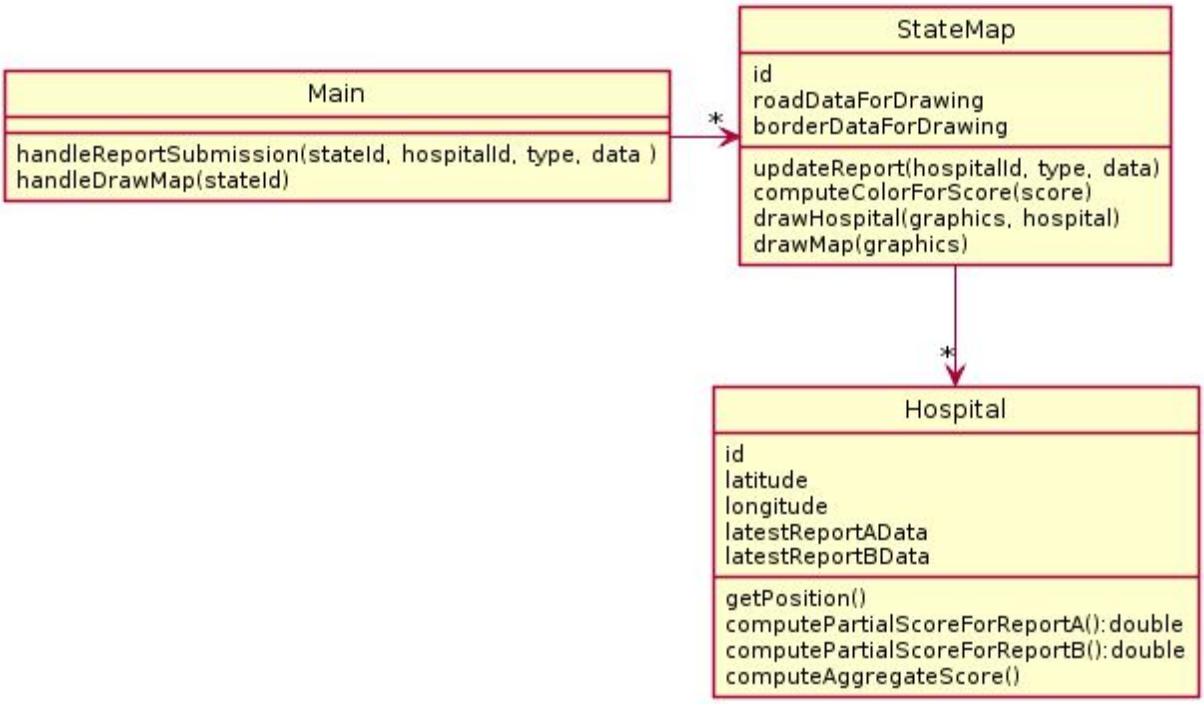
Problems with B

Commentary
How I found the problem: the red flag was the big list of ask methods in hospital. What is doing the asking (StateData)? Is that functionality that makes more sense in hospital so it can be encapsulated? Yes.

BTW: what is NOT a problem is that StateMap doesn't have a direct reference to Hospital. If you look at the methods carefully, you can see that StateData can extract that data from the hospital object and call the drawHospital method on StateMap. The fact that Hospital/StateMap are decoupled from each other is a good thing about this design, not a bad thing.

The Problems
4a. Ask methods on hospital meaning it is not in control of its own data

Final solution



Many solutions kept StateData and StateMap from the previous solution. That's fine, but I thought StateData didn't really have a responsibility anymore now that hospital can do the computation itself. So my solution has only 3 classes.

A few folks kept BOTH the StateData/StateMap distinction AND made a class for Reports. Although I didn't take off for it, that was really a lot of classes for the amount of functionality in this problem.

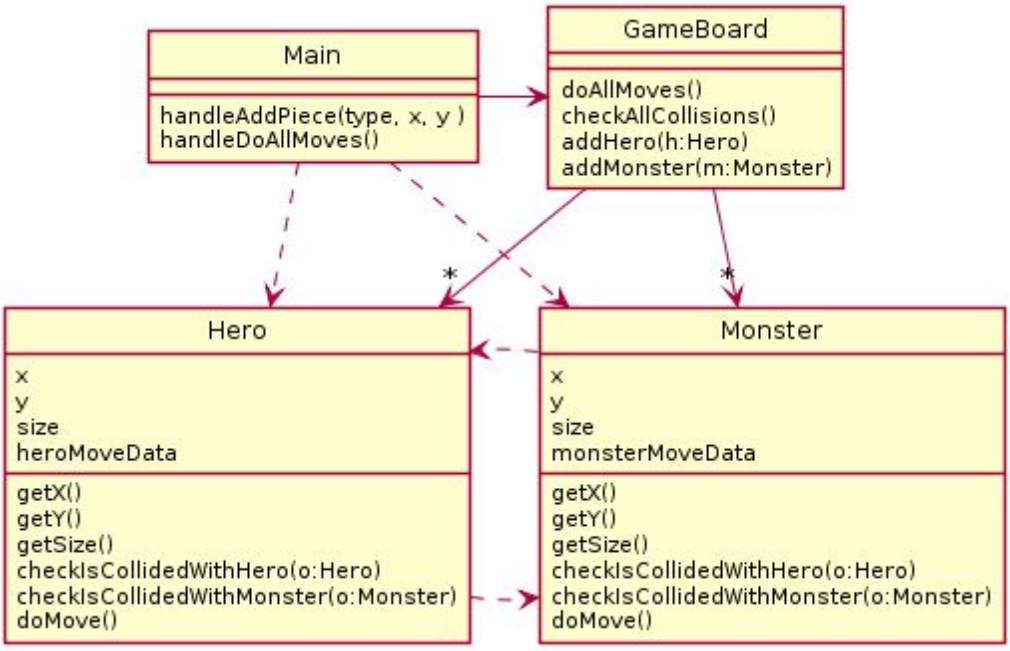
A few folks who had the StateData/StateMap distinctions made BOTH StateData and StateMap have a list of all the hospitals. This in my opinion was too much coupling (though maybe not egregious enough for me to take off). Remember just because a class needs some data sometimes doesn't mean it necessary has to have a field - in this case the data could be passed to StateMap when it was time to draw.

VideoGame

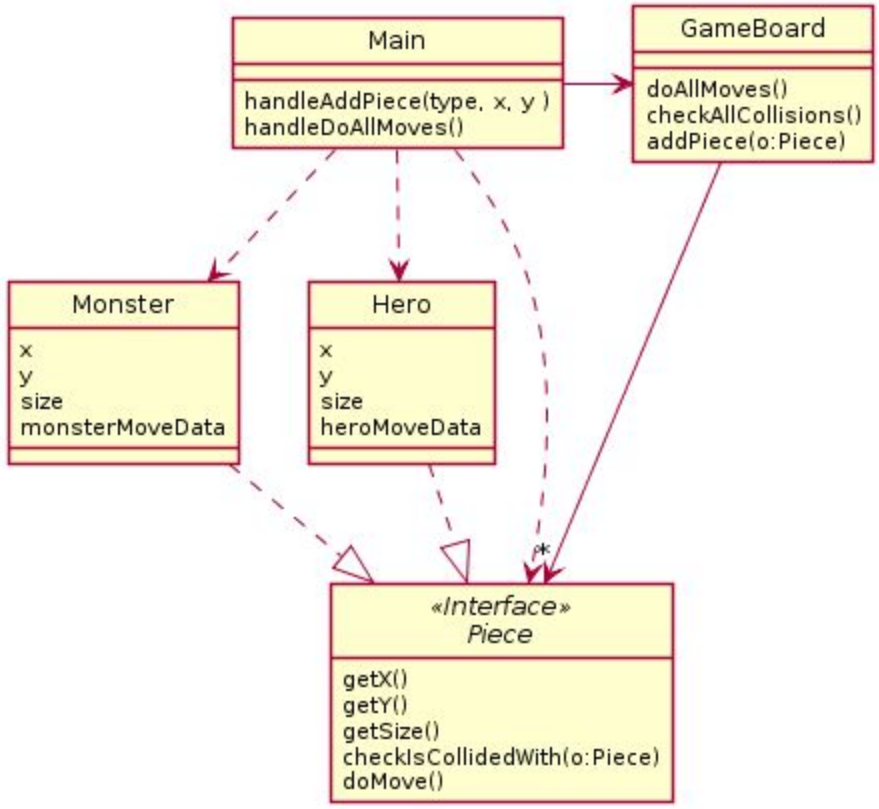
This problem uses interfaces & inheritance, so don't bother looking at if you haven't gotten there in the class yet.

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.

Solution A
For this one, I only want you to identify problems in A *that are fixed by Solution B.*



Solution B


Main
handleAddPiece(type, x, y)
handleDoAllMoves()
GameBoard
doAllMoves()
checkAllCollisions()
addPiece(o:Piece)
Monster
x
y
size
monsterMoveData
Hero
x
y
size
heroMoveData
*
«Interface»
Piece
getX()
getY()
getSize()
checkIsCollidedWith(o:Piece)
doMove()

Solution

Problems with A

...that are Fixed by B

4. Too many interdepencies between classes.
5. Duplicated code for addHero/addMonster in GameBoard, similar fields in GameBoard. Reduced complexity in checkAllCollisions and doAllMoves.
5. 2 copies of checkIsCollidedWith in both Hero and Monster.

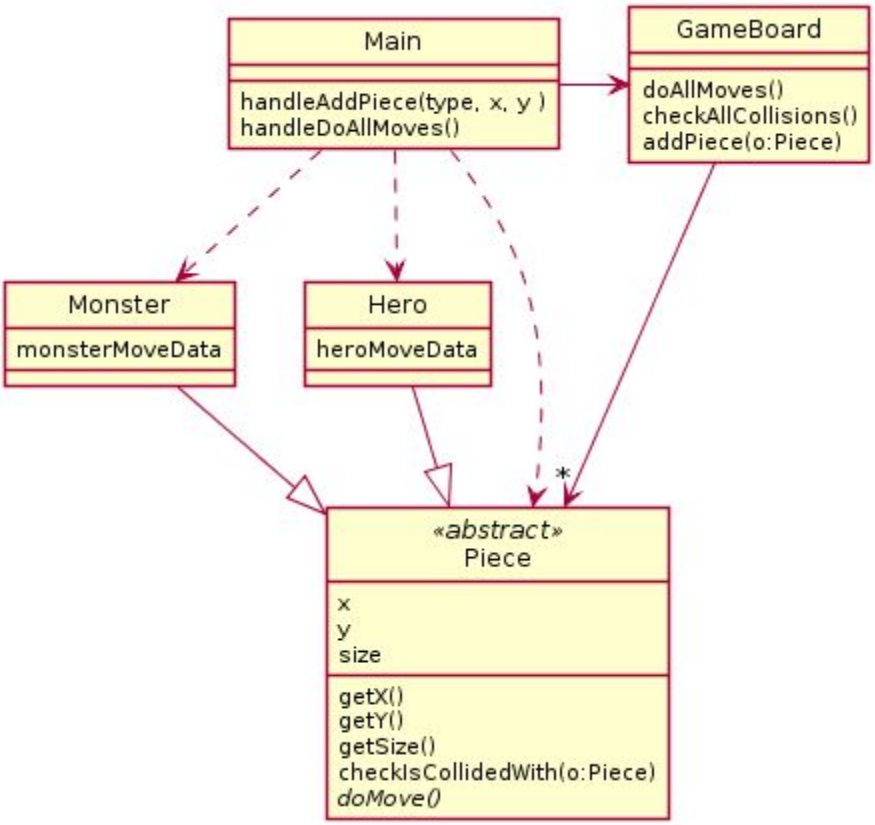
Key point: What has NOT improved is the duplication between Hero and Monster. It may seem that way because Piece seems to have "taken" the duplicated methods. However, Piece is an *INTERFACE* so there is still a getX() getY() getSize() checkIsCollidedWith() in both classes. Similarly, there are still duplicated fields.

This is usual: interfaces deal with duplication in the clients of classes with similar interfaces. So GameBoard (the client of Hero and Monster) is improved. Hero and Monster (for the most part) remain duplicated.

Problems with B

Duplication of code and fields across Hero and Monster.

Final Solution



Now rather than use interfaces we've used an abstract class. The duplicated methods can be put in Piece and be inherited by Hero and Monster.