CSSE 220: Object Design

Part 1 of Many

Also Class Diagrams

Import *FirstOODesignPractice* from Git clone

Designing Classes

- Programs typically begin as abstract ideas – I want Twitter, for dogs
- These ideas form a set of requirements (i.e. what the user wants)
  - This is a difficult process – see CSSE 371 for learning basics for how to do this
  - In this class, instructors hand you the requirements – you build the software system from these
- We must take these requirements, and figure out an approach for our coding
- Usually the approach is not obvious
- So we propose designs, then iteratively refine them into something that might work (continued…)

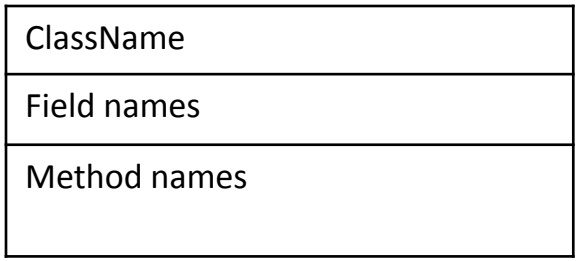
Designing with Iterative Refinement

- So we propose designs, then iteratively refine them into something that might work
  - Many bad ideas in the process – as we iteratively define the design, we'll end up tossing them out
  - We don't want to go through the effort of implementing bad ideas in code – it's too time and resource costly to implement an idea in order to determine if it's an incomplete/inconsistent solution
  - So, we need a way to communicate/think concretely about these half-baked program approaches
- We need a diagram language!
  - With these diagrams, which can be put together with reasonable cost, we can test out ideas for solutions, and thus help us eliminate the incomplete/inconsistent approaches early on in the refinement

Tools of the Trade - Diagramming

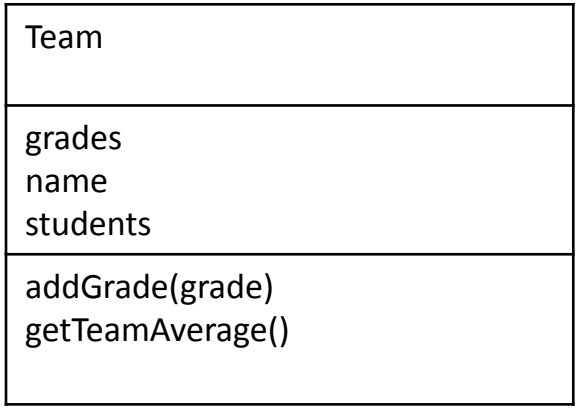
- Class Diagrams (UML)
- UML – Unified Modeling Language
  - Language unspecific (or language agnostic)
  - Has a lot of different diagrams it provides specifications for – but the class diagram language is the most widely used

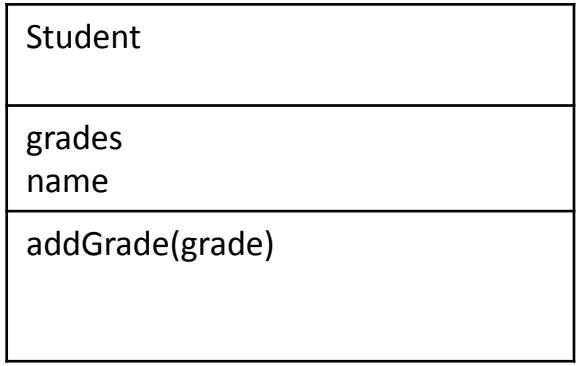
A little class diagram will get you a long way



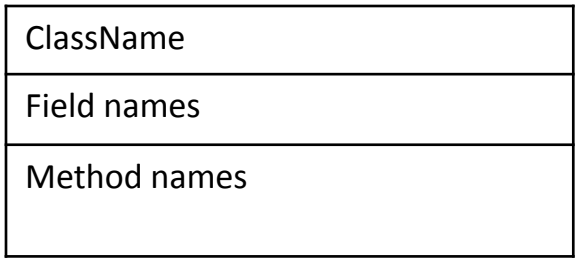
- Classes represented by a diagram with 3 sections
- Not the final version of UML we will teach, but covers the main points

Example – "Team" class from TeamGradebook



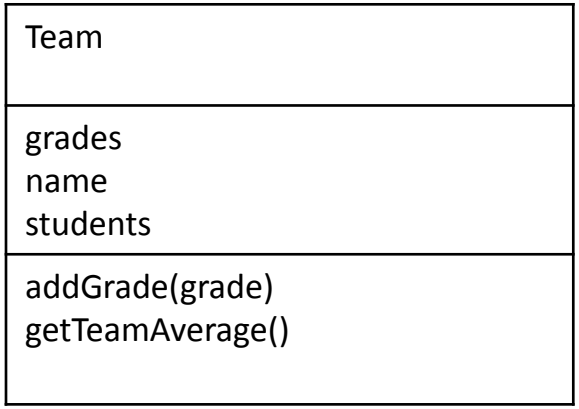


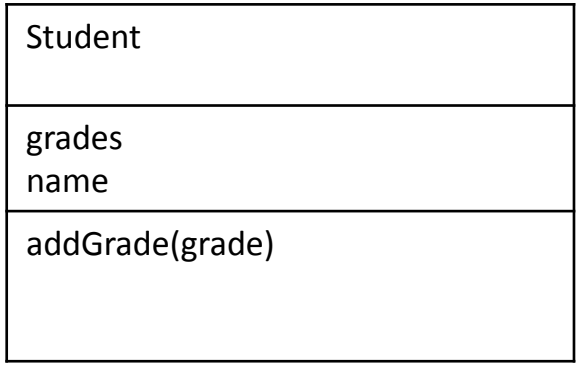
A little class diagram will get you a long way



- Classes represented by a diagram with 3 sections
- Not the final version of UML we will teach, but covers the main points

Example – "Team" class from TeamGradebook

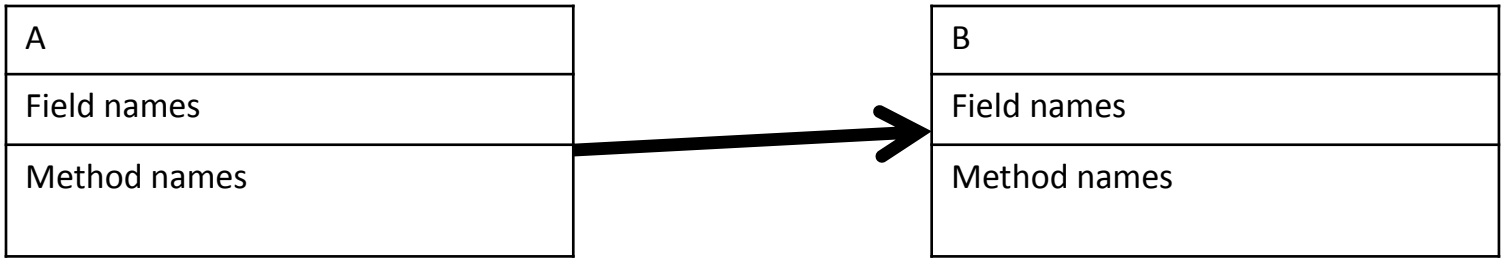




- At this level we're leaving out:
  - types declarations for parameters
  - type declarations for field names
  - Return type declaration for a function method

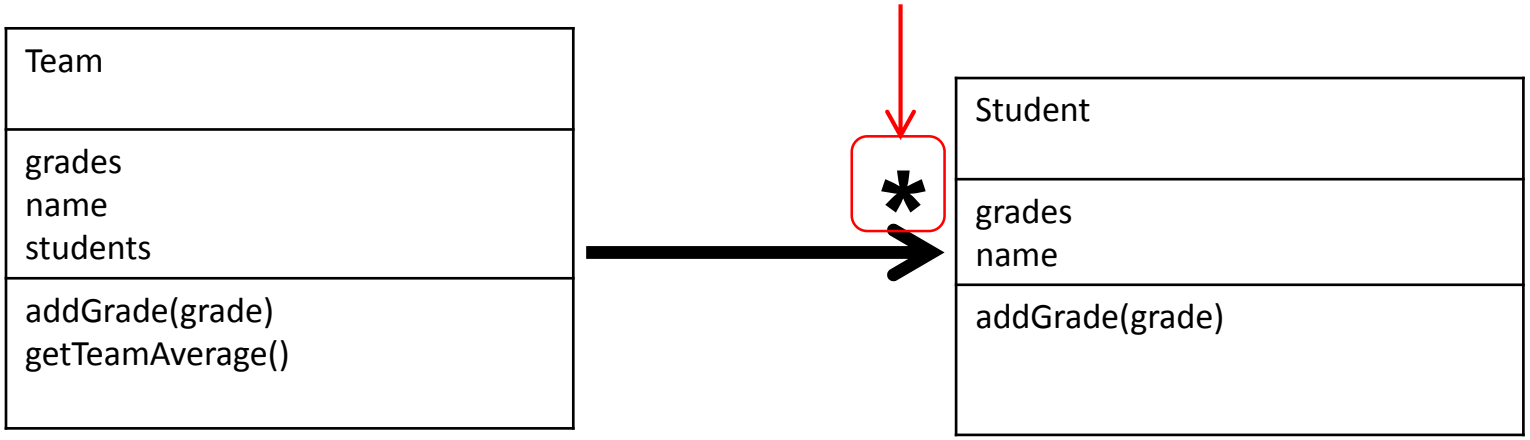
Arrows – to illustrate relationships

A has a B (field)



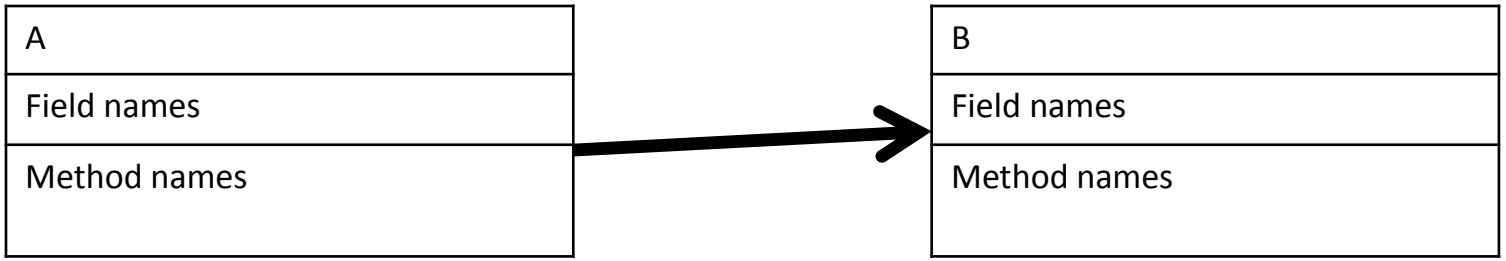
Example

Note: the star means several, zero to many.
Often stored in a collection, e.g., a list



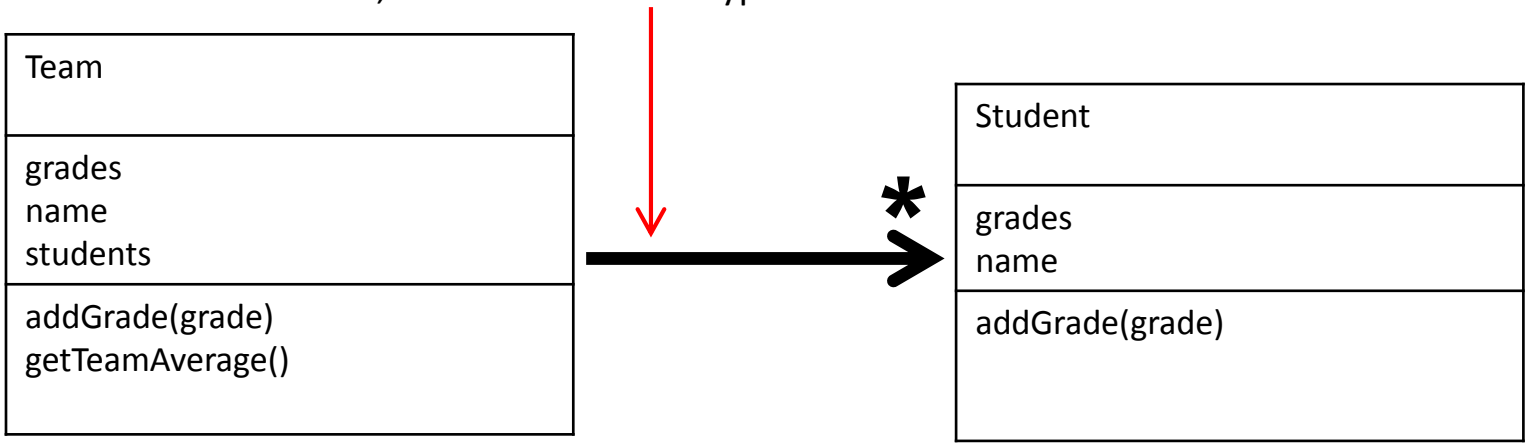
Arrows – to illustrate relationships

A has a B (field)



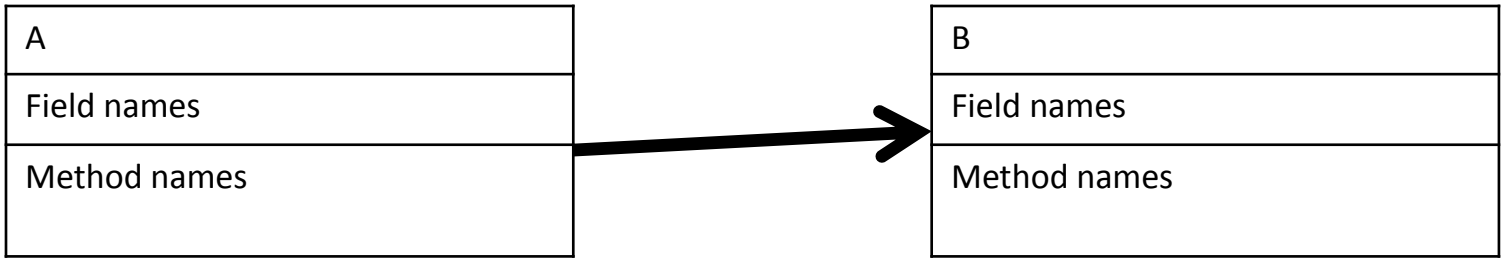
Example

This arrow means, Team has a field of type Student

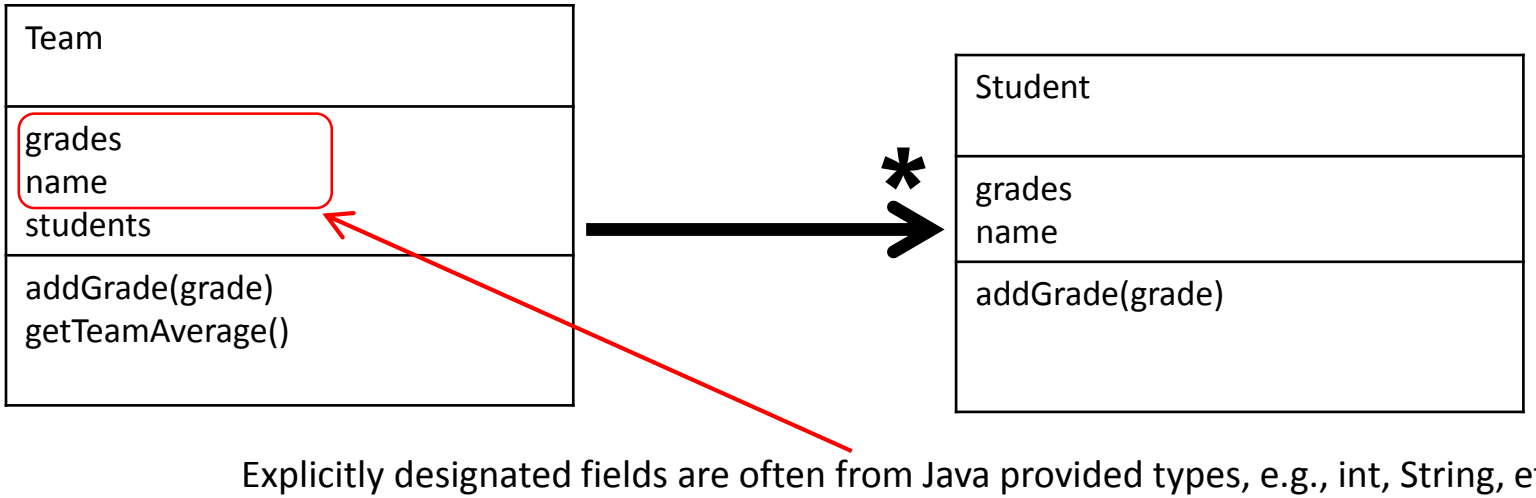


Arrows – to illustrate relationships

A has a B (field)



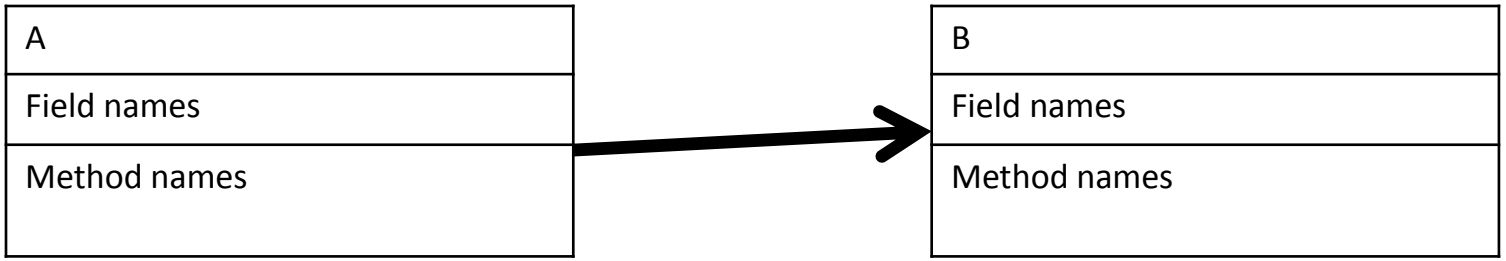
Example



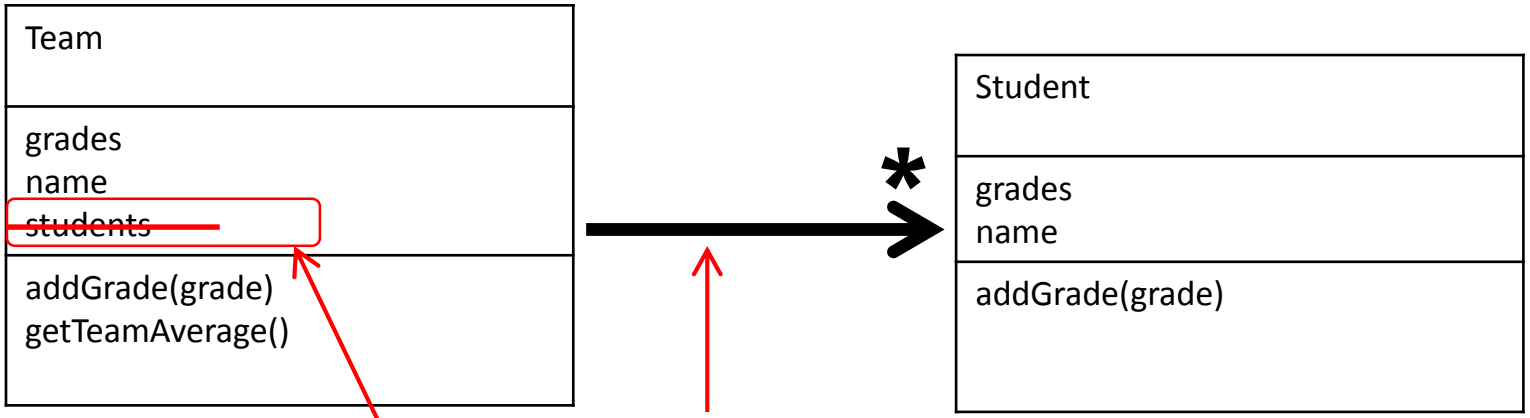
Explicitly designated fields are often from Java provided types, e.g., int, String, etc.

Arrows – to illustrate relationships

A has a B (field)



Example



When there's an arrow to another class,

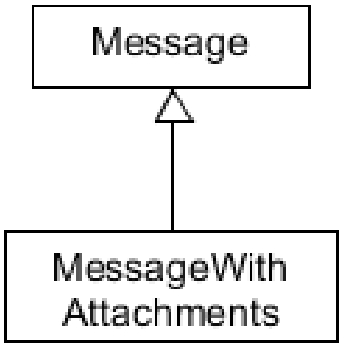
then we often do NOT explicitly define the field at the tail of the arrow

Practice exercises

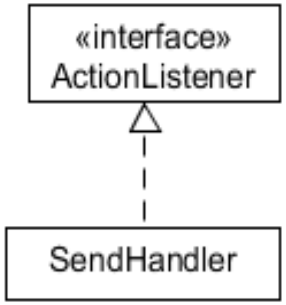
- From the today's in-class quiz do questions #1 and #2
- About 10 minutes

Summary of UML Class Diagram Arrows

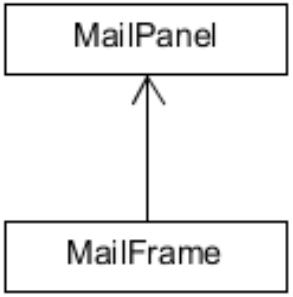
Inheritance (is-a)



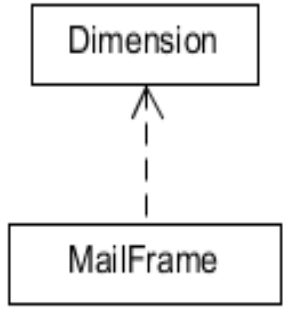
Interface Implementation (is-a)

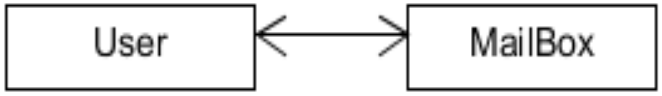


Association (has-a-field)

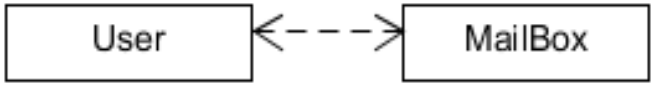


Dependency (depends-on)

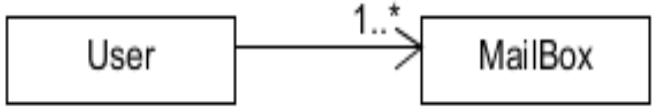




Two-way Association



Two-Way Dependency

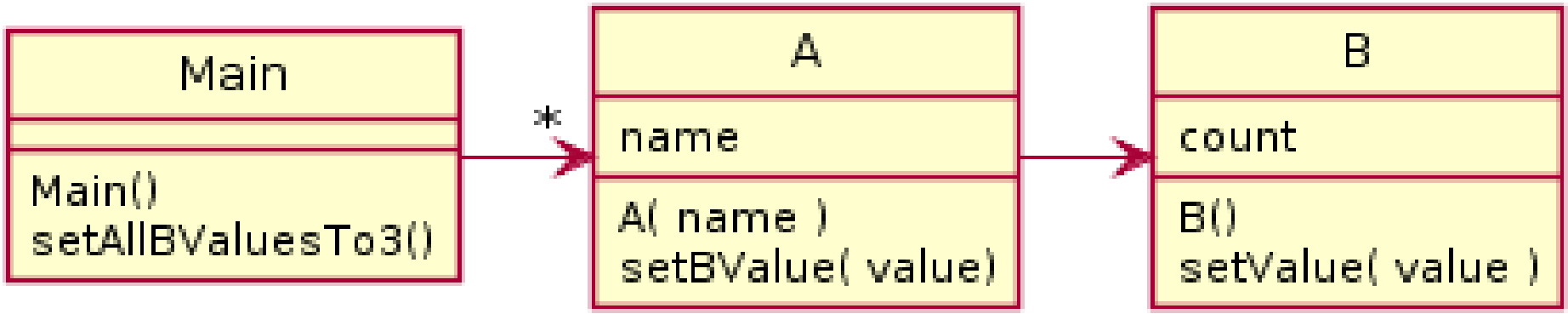


Cardinality
(one-to-one, one-to-many)
One-to-many is shown on left

Let's try to code a simple UML diagram!

Open up Eclipse and turn this diagram into Java classes/code

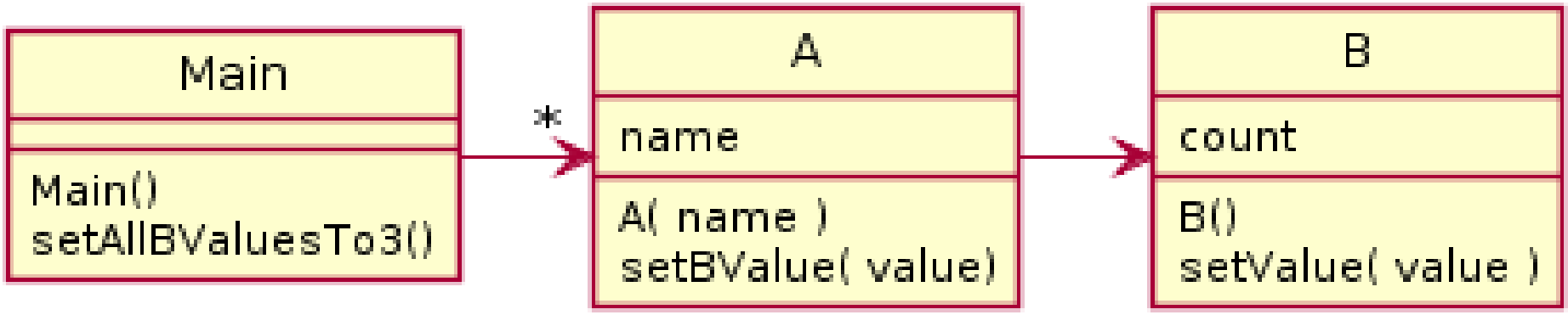
1. First add the class name and its fields
2. For methods, create empty methods and leave for step 3
3. Finally, implement the methods as the **last** thing you do.



Let's try to code a simple UML diagram!

Open up Eclipse and turn this diagram into Java classes/code

1. First add the class name and its fields
2. For methods, create empty methods and leave for step 3
3. Finally, implement the methods as the **last** thing you do.



Note: A diagram can generate quite a bit of Java

Overview: Principles of Design (for CSSE220)

- Make sure your design **allows proper functionality**
  - Must be able to **store required information** (one/many to one/many relationships)
  - Must be able to **access the required information** to accomplish tasks
  - Data should **not be duplicated** (id/identifiers are OK!)
- Structure design **around the data** to be stored
  - **Nouns should become classes**
  - **Classes should have intelligent behaviors** (methods) **that may operate on their data**
- Functionality should be **distributed efficiently**
  - **No class/part should get too large**
  - **Each class should have a single responsibility** it accomplishes
- **Minimize dependencies** between objects when it does not disrupt usability or extendability
  - Tell don't ask
  - Don't have message chains
- **Don't duplicate** code
  - Similar "chunks" of code should be **unified into functions**
  - Classes with similar features should be given **common interfaces**
  - Classes with similar internals should be simplified using **inheritance**

Structure design **around the data** to be stored

- **Nouns should become classes**
- **Classes should have intelligent behaviors** (methods) **that may operate on their data**

A good object oriented design is structured around the data

- Look for the nouns in your problem, consider making each of them a Class
  - …if work related to that noun is complex enough
- Store the data as fields in the Class
- Add operations to the Class to accomplish what you need, i.e., manipulate the internal class fields
- Avoid Plural Nouns – i.e., Class name should be singular. Know that client can always make multiple object instances of this class

Make sure your design **allows proper functionality**

- Must be able to **store required information** (one/many to one/many relationships)
- Must be able to **access the required information** to accomplish tasks
- Data should **not be duplicated** (id/identifiers are OK!)

An object oriented design must work!

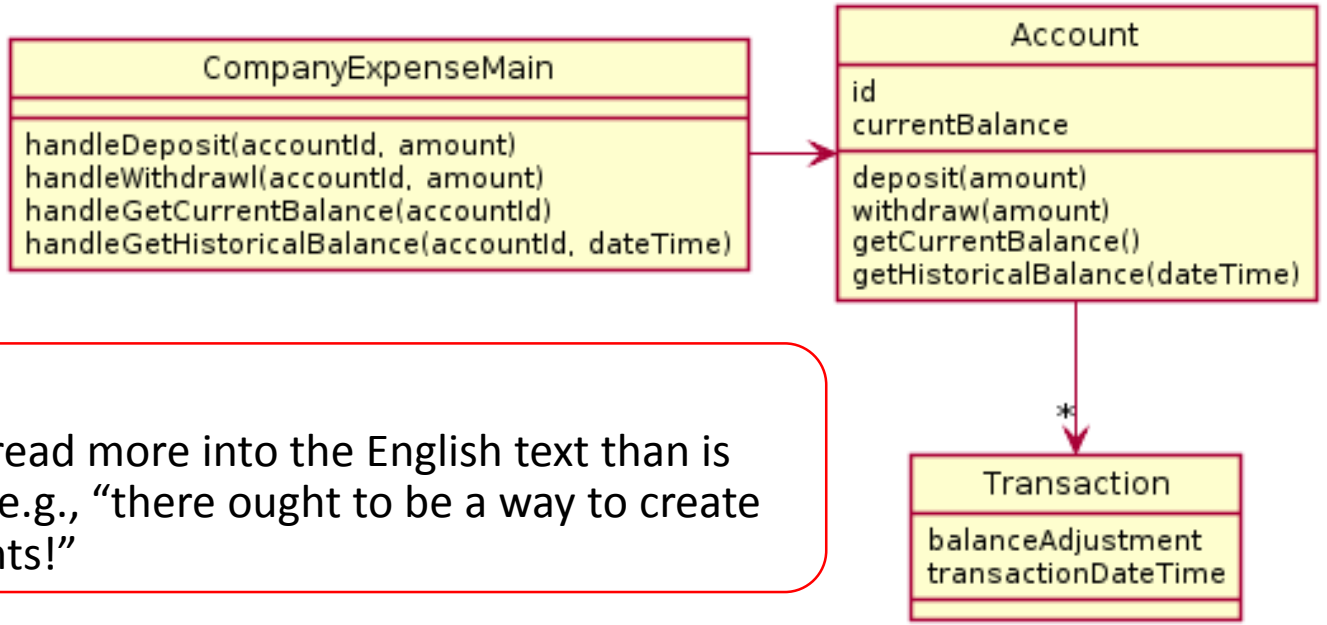
Make sure all the data that you need is stored somewhere

- Think as a client programmer: Can you access all the necessary data from the new Class?
  If not, then the design of the new class “doesn’t work”
- The solution is not to keep 2 copies of the same data, one in the client and one in the new Class, change the new Class

What is wrong with this design?
5 to 10 min, in pairs if you like

A particular company keeps a variety of different accounts for its projects. Each account has an account number and a balance. When a deposit or withdrawal occurs, the transaction occurs immediately and the current balance should be updated. The system should support getting the current balance.

The system should also support getting the balance as it existed at any date/time in the past. Note the input historical date/time may not correspond to a particular transaction time - e.g. if the system had a balance of $1 at 1 pm and then was changed to $2 at 3 pm, a request for the balance at 2 pm should return $1.



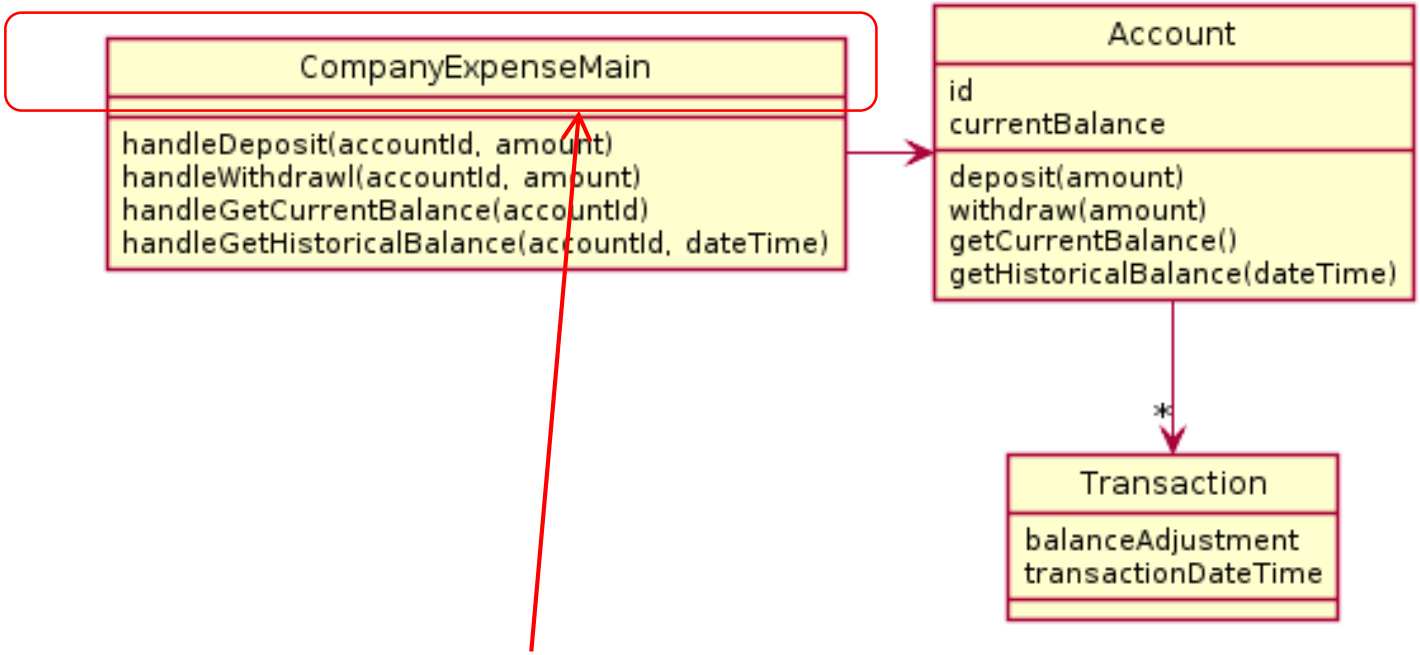
Note:

Don't read more into the English text than is there, e.g., "there ought to be a way to create accounts!"

Quiz question #3

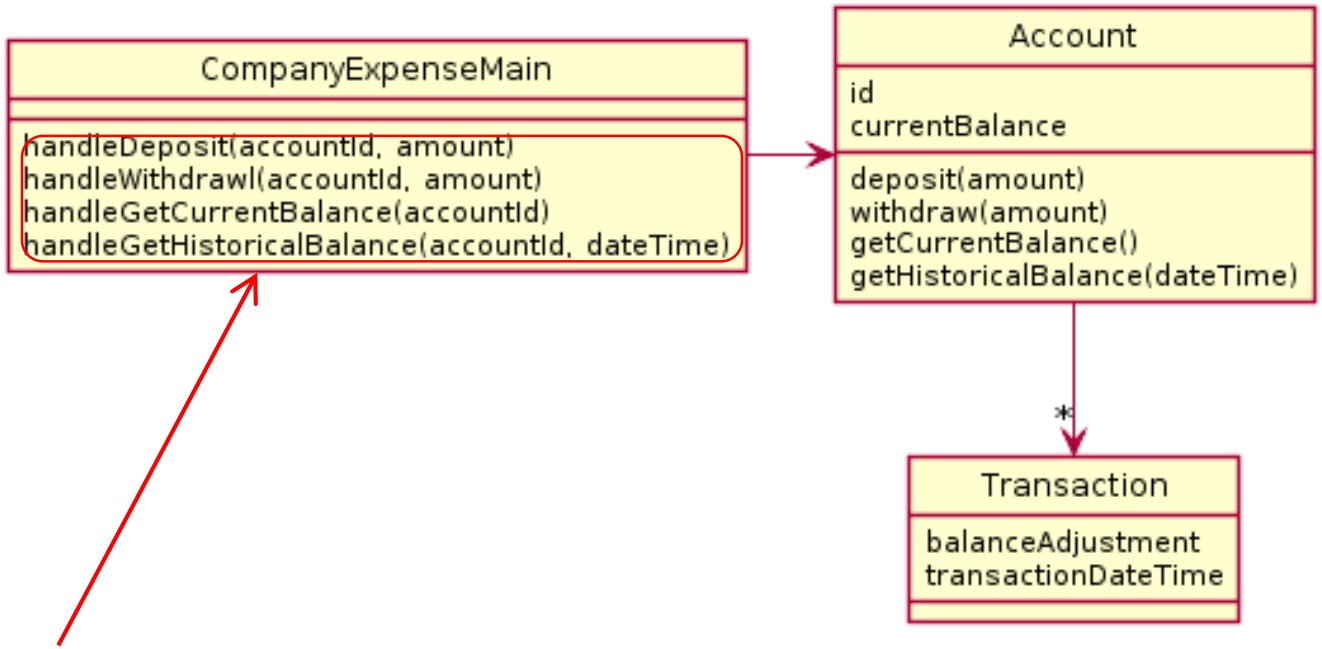
3

Good parts of the design - Main class

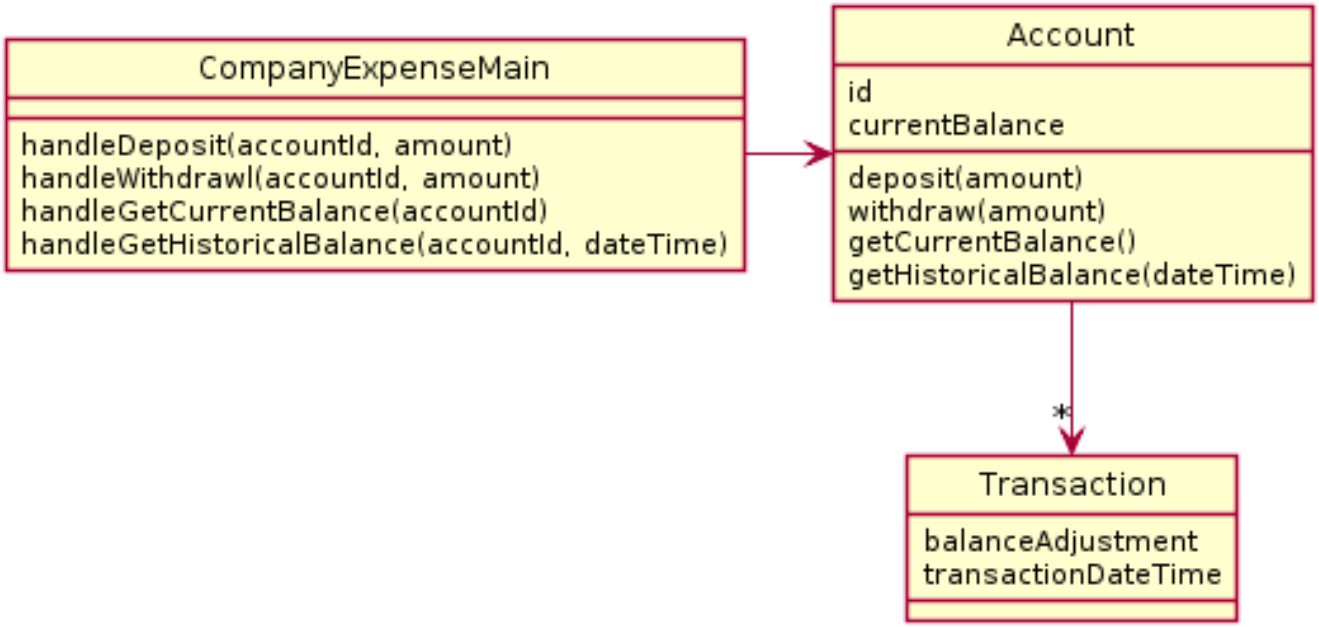


- Every program starts somewhere, and any design should make clear where the starting point is. In our class, we will name the starting point class SomethingMain

Good parts of the design – “handle” methods



- In our very simple designs, this class also deals with user input
- "handle" methods will have special meaning for us, as they will represent places where user commands enter the system.
- By looking at parameters to the handle methods, you can sometimes get more info on how the various commands in the description should work.



This design does not function correctly. Why?

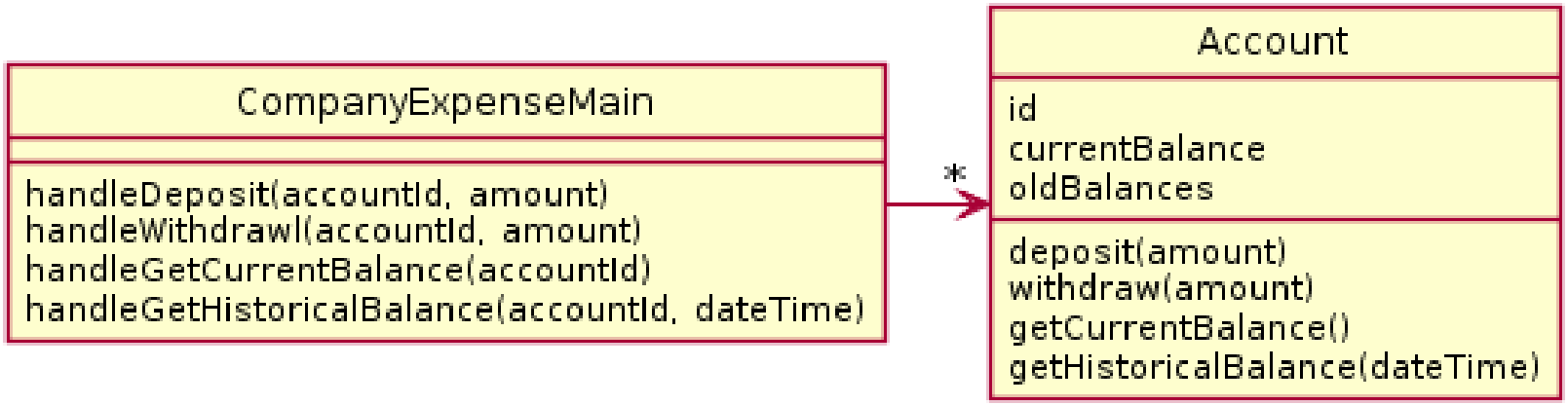
1. Main has only one account, but the system needs to support many. How do we know that from this diagram?

2. Also, computing the data for historic balances is moderately hard.

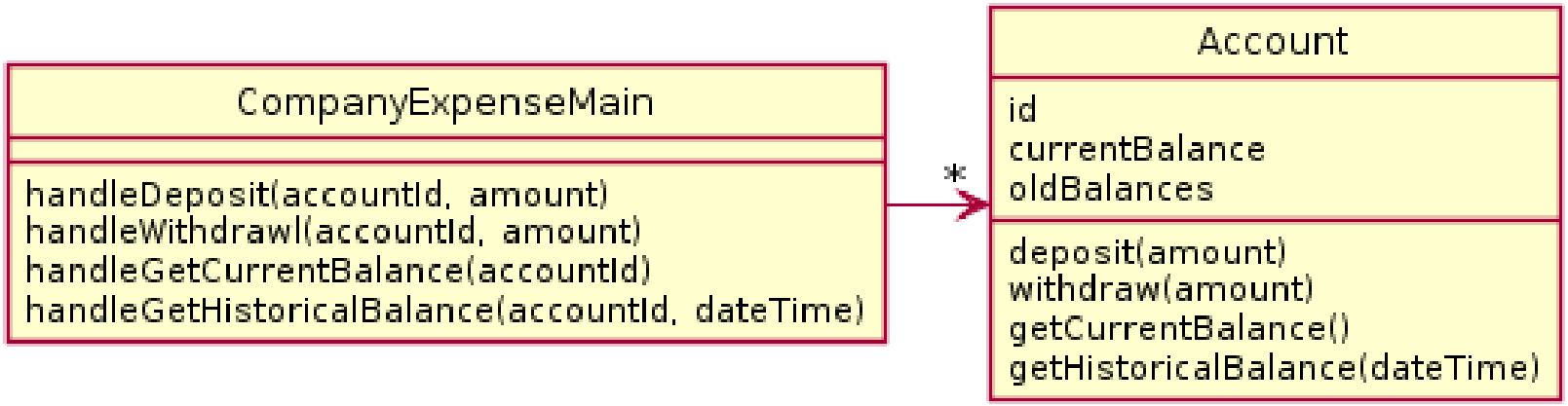
A particular company keeps a variety of different accounts for its projects. Each account has an account number and a balance. When a deposit or withdrawal occurs, the transaction occurs immediately and the current balance should be updated. The system should support getting the current balance.

The system should also support getting the balance as it existed at any date/time in the past. Note the input historical date/time may not correspond to a particular transaction time - e.g. if the system had a balance of $1 at 1pm and then was changed to $2 at 3pm., a request for the balance at 2pm should return $1.

Find the problem with this design and then propose your own.



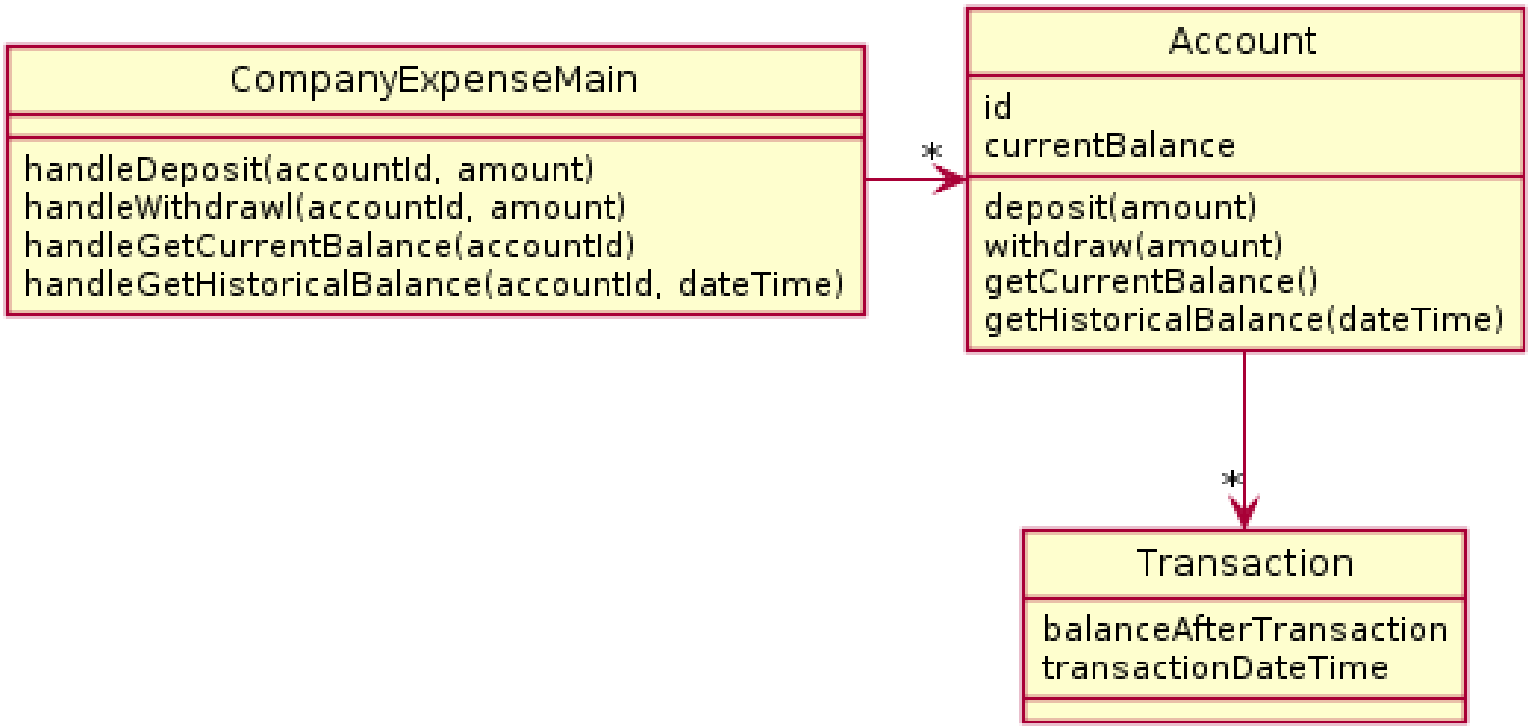
Questions #4 & #5 on today's quiz



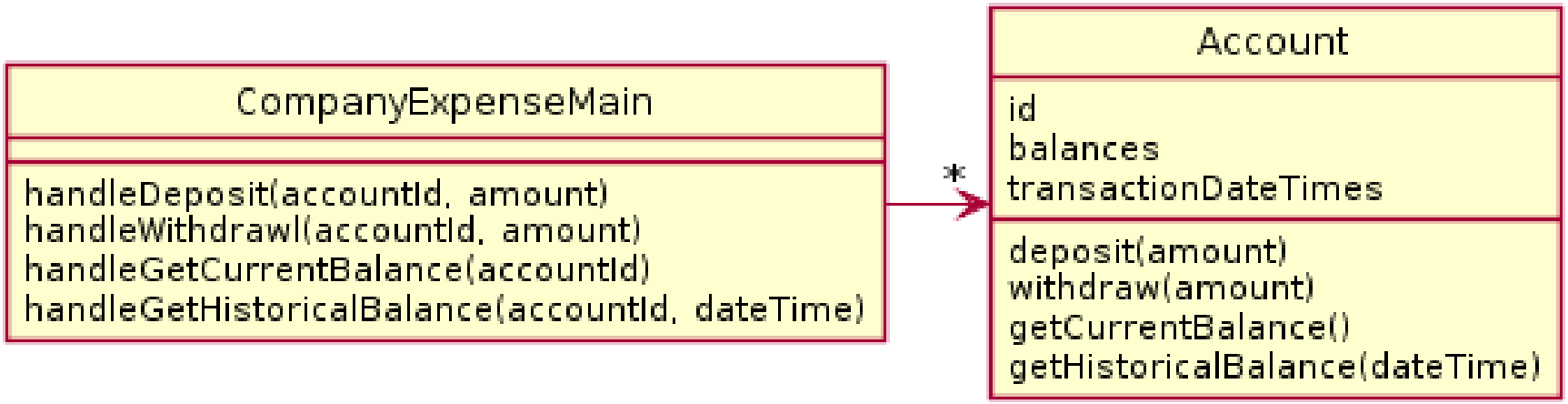
This design does not function correctly.

Account does not have enough data to make getHistoricalBalance work. oldBalances stores perhaps a list of balances? But the date of the transactions is not stored, so we can't look up the balance on a particular day/time.

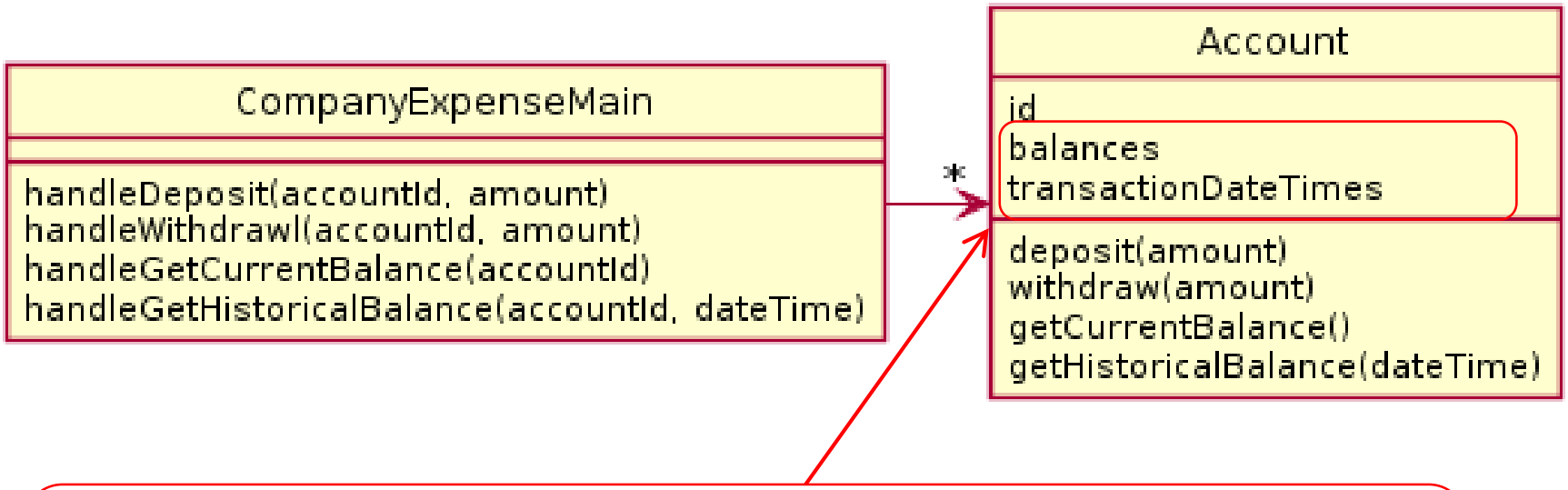
My solution 1



My Solution 2



My Solution 2

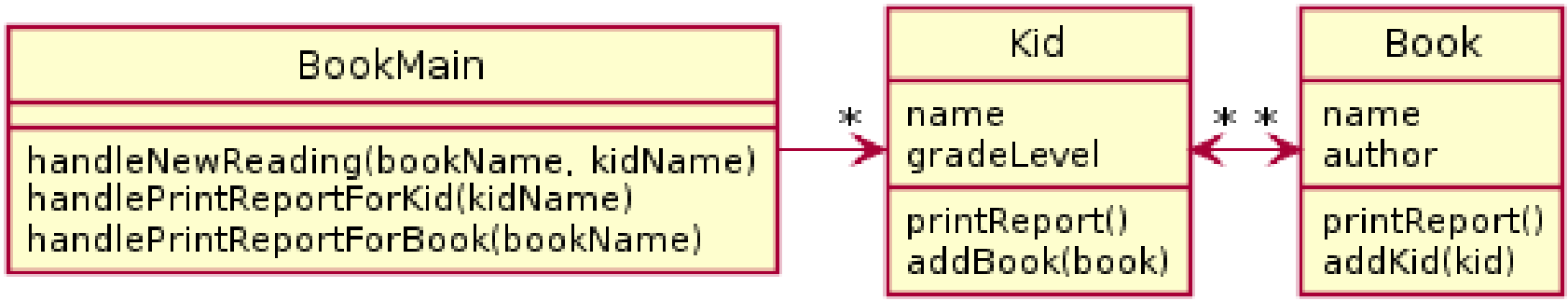


Note:

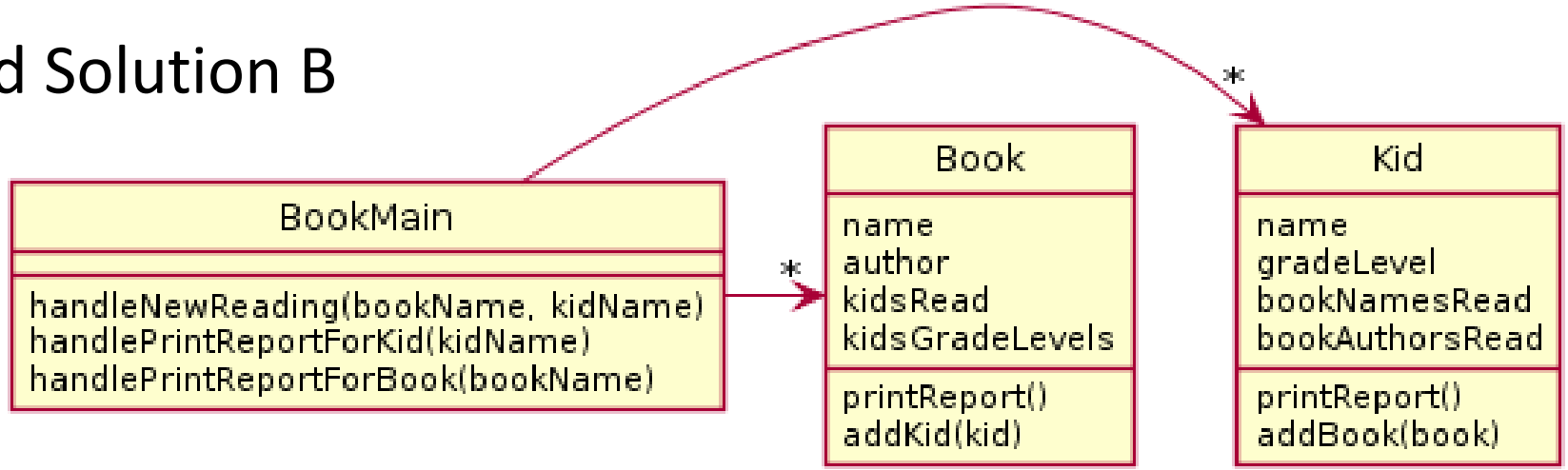
- *balances* & *transactionDateTimes* – parallel ArrayLists, for example
- You might change this design to create a Class to capture this functionality

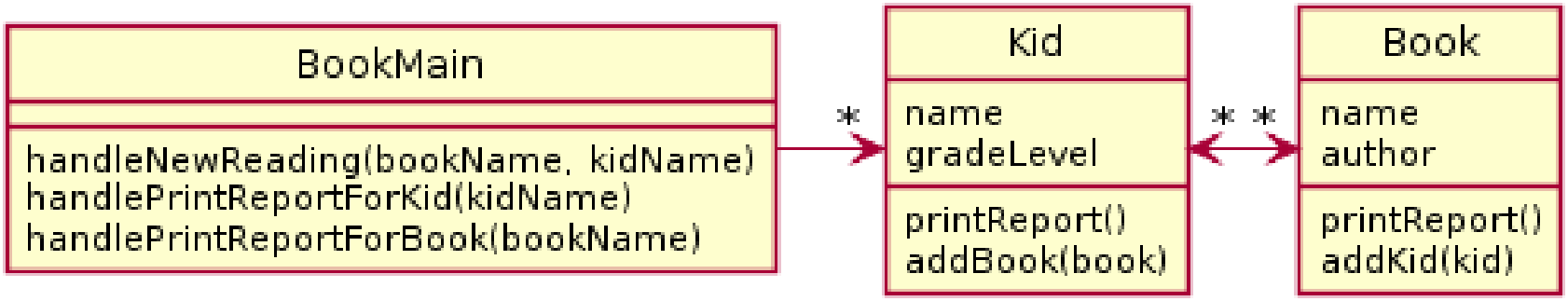
A website tracks books and the kids that read them. For each book the system stores the name and author. For each kid the system stores name and grade level. The teacher enters when a kid reads a particular book. It should be possible to print a report on a book that includes all kids who have read a particular book (with their grade level). It should be possible to print a report on a kid that includes the books (with authors) a particular kid has read.

Bad Solution A



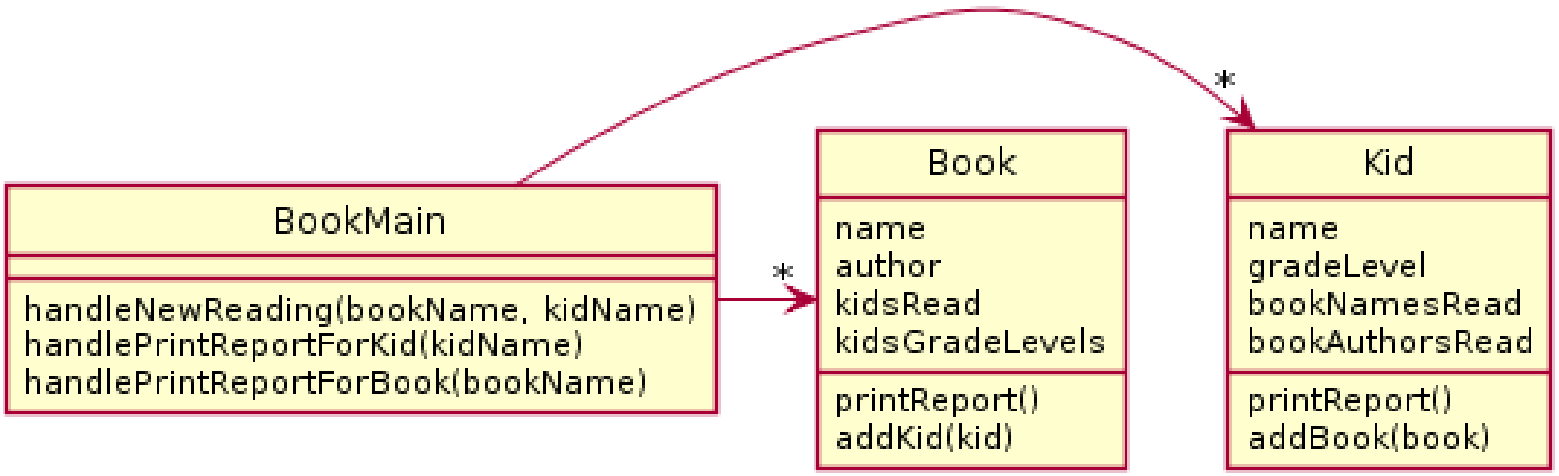
Bad Solution B





This design does not function.

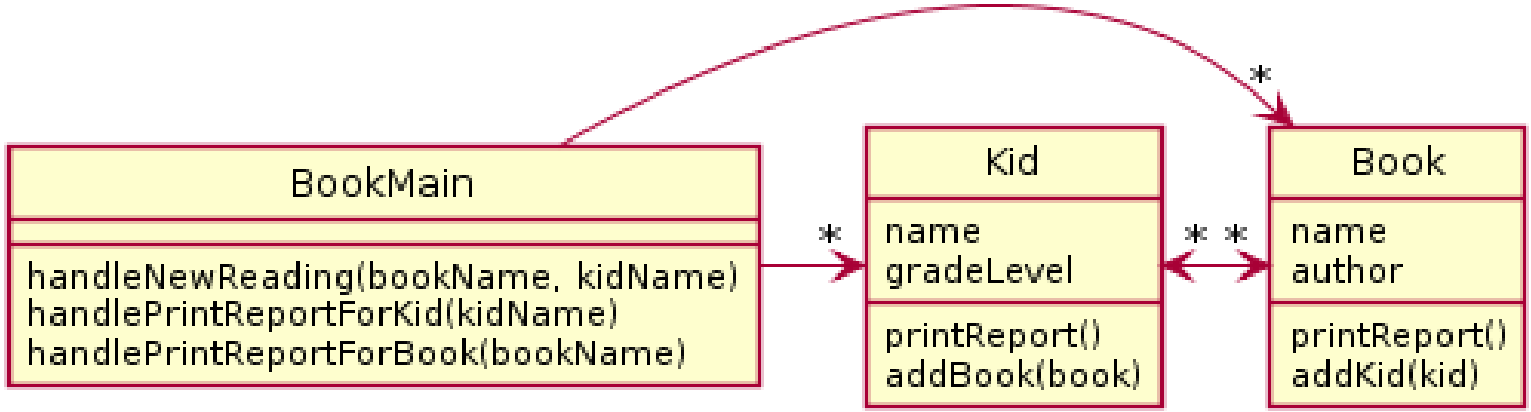
There is no (sane) way to look up a book for printing a report or for associating with a Kid.



This design functions but there is a very large amount of duplication – which in general we want to avoid.

In particular, the author/title information in the kid is duplicated and the name/grade level information in the book is duplicated.

My Solution


BookMain
handleNewReading(bookName, kidName)
handlePrintReportForKid(kidName)
handlePrintReportForBook(bookName)
*
Kid
name
gradeLevel
printReport()
addBook(book)
* *
Book
name
author
printReport()
addKid(kid)
*

In most cases non-workable design is caused by…

- Not reading the problem carefully or not mapping it to design carefully (e.g. not noticing that each kid reads several books, not just one)
- Not thinking about how specific required features might be implemented (e.g. how can we print a book report if we don’t have access to the book objects?)
- Duplicating data (e.g. what does it matter if we store a copy of the author and title for every kid that reads the book)

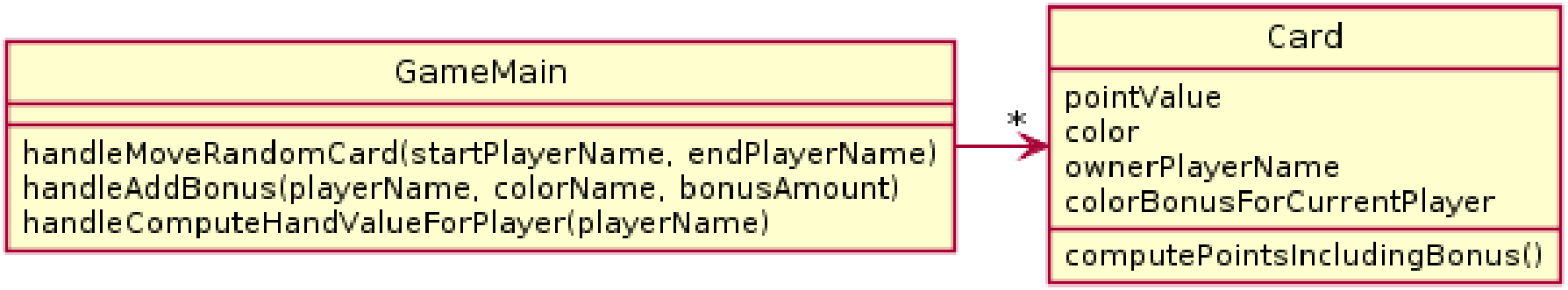
For Next Class

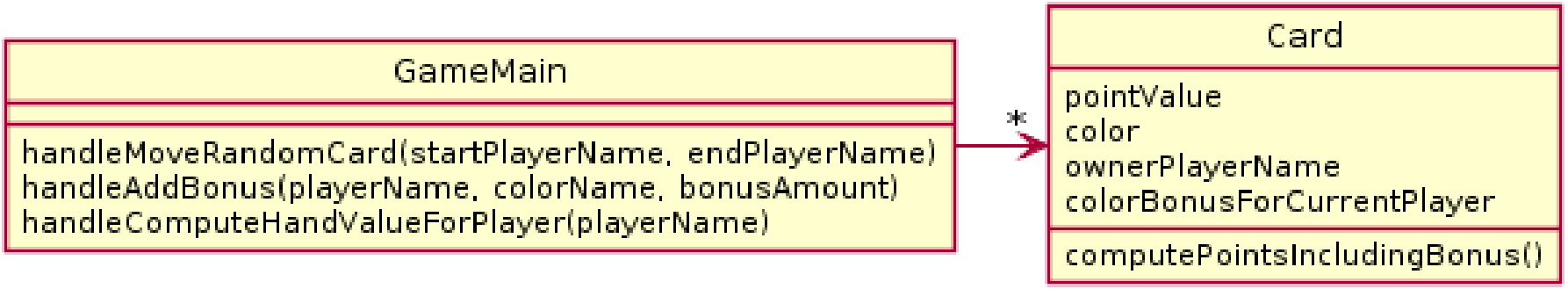
- Solve the 2 Design Problems in the handout
  - Bring your solution to be collected **at the start of** next class
  - We will go over the solution at the beginning of next class
  - Anything turned in late will be worth zero points
  - No extensions will be given

- We'll discuss more design principles after Exam 1

In a particular card game, players have hands of cards. Each card is worth some points and also has a color (red, blue, green). During play, players accrue bonuses that mean cards of a particular color are worth bonus points. During play, sometimes a random card is selected from one player's hand and moved to another player's hand. At the end of game, it is necessary to compute the total points for each player's hand.

What is wrong with this design? (Hint: look at and refer to your design principles by number). I see at least 2 separate categories violated.





My answer (in order of importance)

1a. The design does not function correctly

The player's color bonus cannot be preserved if he/she loses all their cards of a particular color

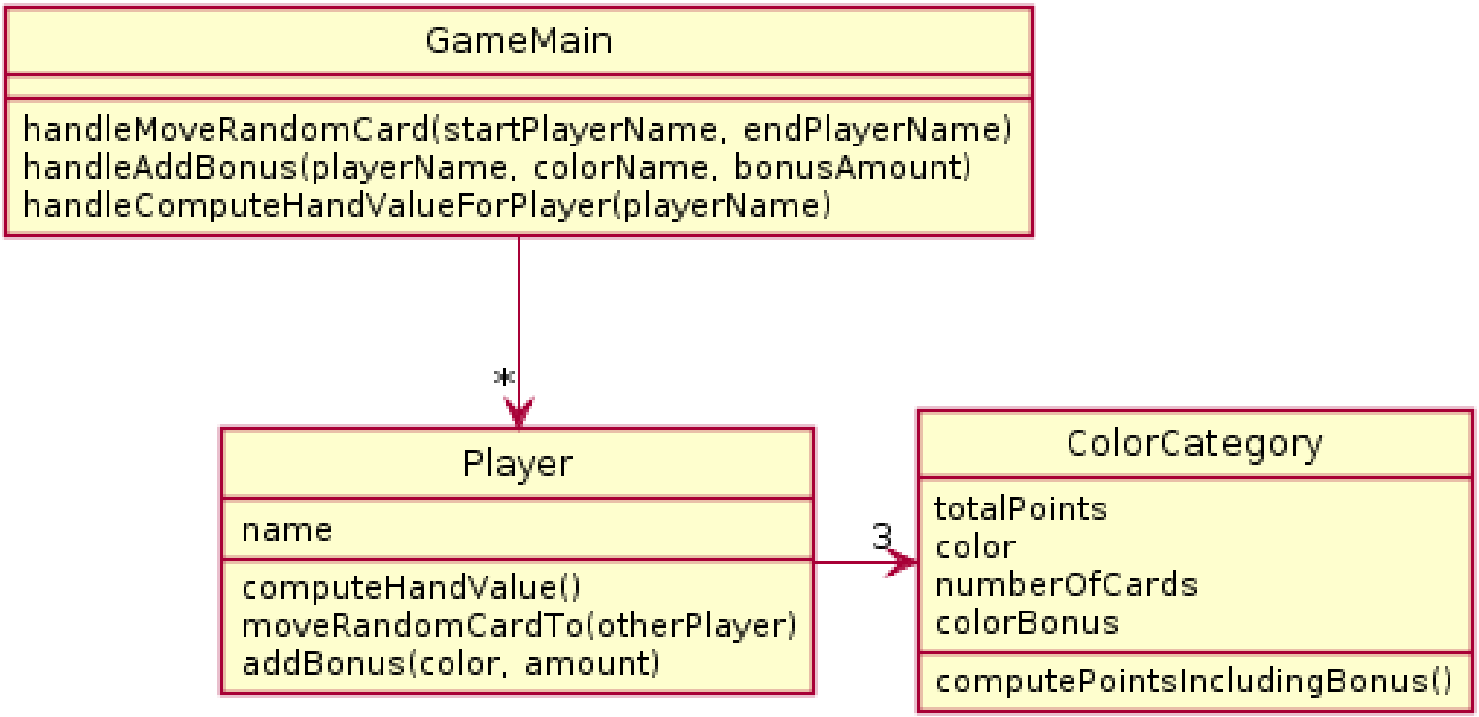
It requires iterating over all objects to get the full set of cards in the players hands to move cards or compute final total

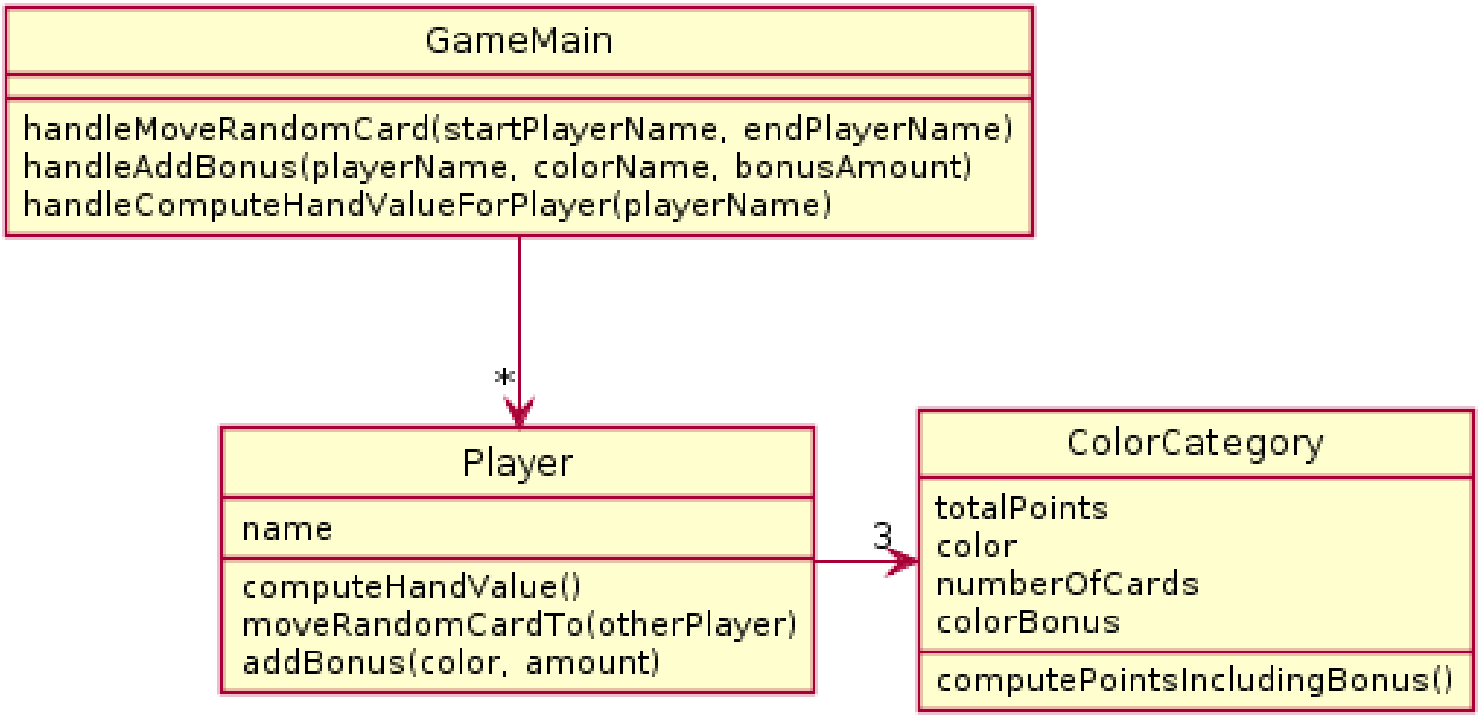
1c. Playername & player color bonus are duplicated across cards

2a. Player (common noun from problem) not represented

In a particular card game, players have hands of cards. Each card is worth some points and also has a color (red, blue, green). During play, players accrue bonuses that mean cards of a particular color are worth bonus points. During play, sometimes a random card is selected from one player's hand and moved to another player's hand. At the end of game, it is necessary to compute the total points for each player's hand.

What is wrong with this design? (Hint: look at and refer to your design guidelines). I see at least 2 separate categories violated.





My answer (in order of importance)

1a. The design does not function correctly

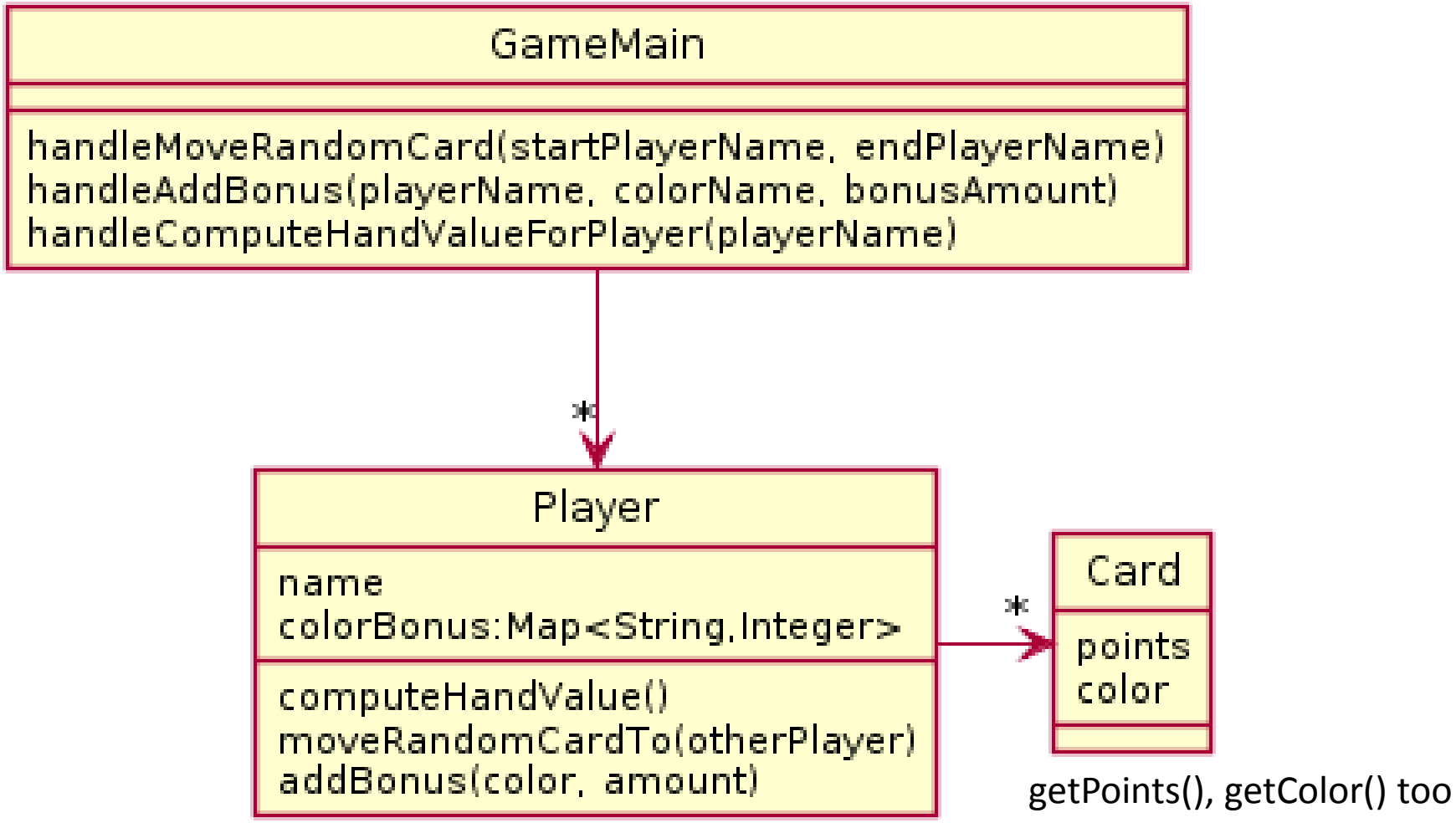
Once a card is added to a players hand, its specific point value is lost so the card cannot be randomly moved to another players hand

2a. Card (common noun from problem) not represented

In a particular card game, players have hands of cards. Each card is worth some points and also has a color (red, blue, green). During play, players accrue bonuses that mean cards of a particular color are worth bonus points. During play, sometimes a random card is selected from one player's hand and moved to another player's hand. At the end of game, it is necessary to compute the total points for each player's hand.

Now design your solution that solves all problems.

My Solution


GameMain
handleMoveRandomCard(startPlayerName, endPlayerName)
handleAddBonus(playerName, colorName, bonusAmount)
handleComputeHandValueForPlayer(playerName)
*
Player
name
colorBonus:Map<String,Integer>
computeHandValue()
moveRandomCardTo(otherPlayer)
addBonus(color, amount)
*
Card
points
color
getPoints(), getColor() too