CSSE 220: Object Design

Part 1 of Many

Also Class Diagrams

Designing Classes

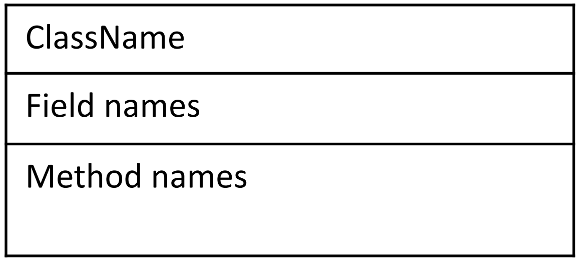
- Programs typically begin as abstract ideas
- These ideas form a set of requirements (i.e. what the user wants)
- We must take these requirements, and figure out an approach for our coding
- Usually the approach is not obvious
- So we propose designs, then iteratively refine them into something that might work (continued…)

- So we propose designs, then iteratively refine them into something that might work
  - Many bad ideas in the process
  - We don’t want to go through the effort of implementing bad ideas in code
  - But we need a way to communicate/think concretely about these half-baked program approaches
- We need a diagram language!

Tools of the Trade

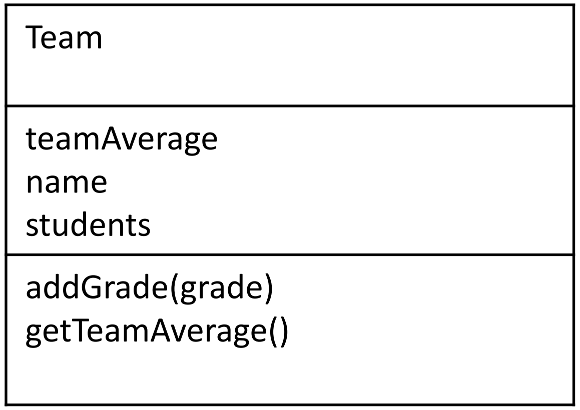
- Class Diagrams (UML)
- UML – Unified Modeling Language
  - Language unspecific
  - Has a lot of different diagrams it provides specifications for – but the class diagram language is the most widely used

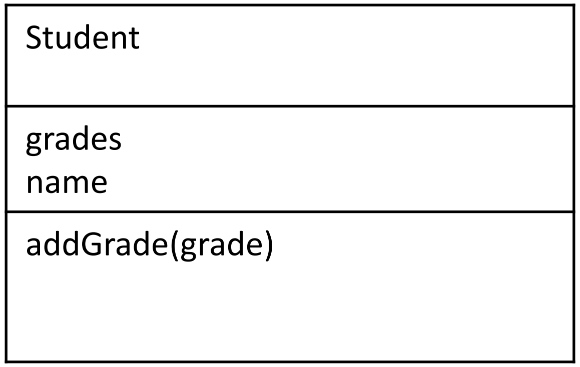
A little class diagram will get you a long way



- 3 sections
- Not the final version of UML we will teach, but covers the main points

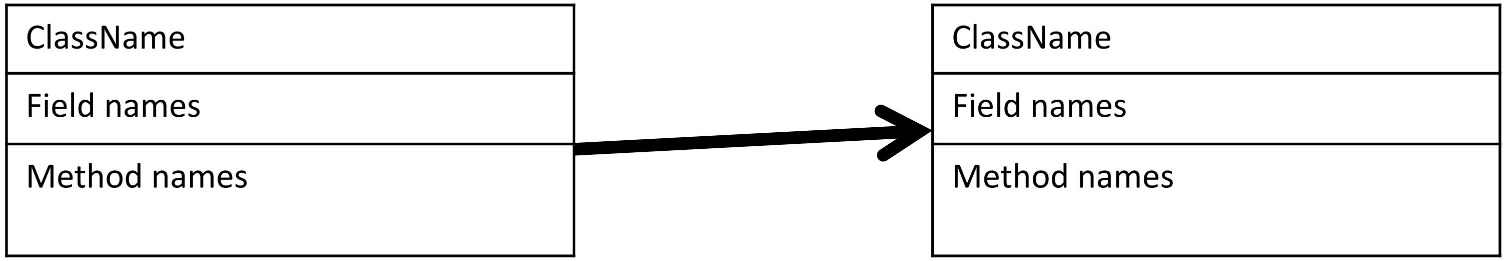
Example





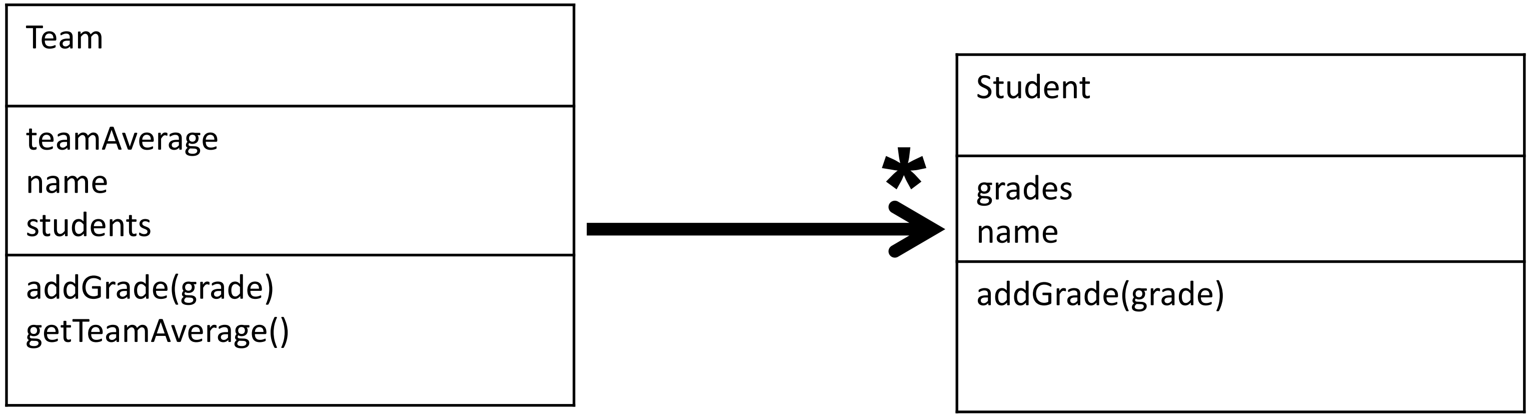
Lines

A has a B (field)



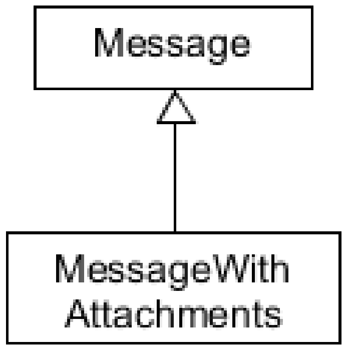
Example

Note the star means several… usually a list or collection.

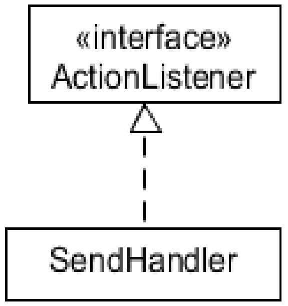


Summary of UML Class Diagram Arrows

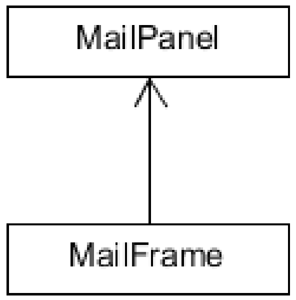
Inheritance (is-a)



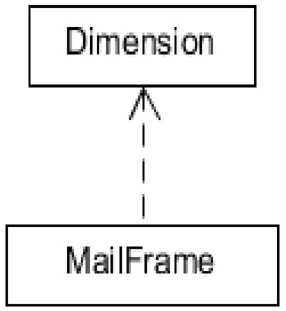
Interface Implementation (is-a)

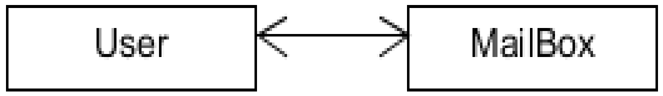


Association (has-a-field)

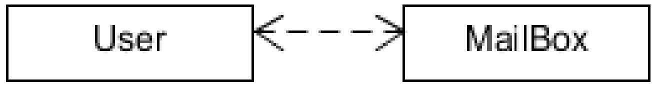


Dependency (depends-on)

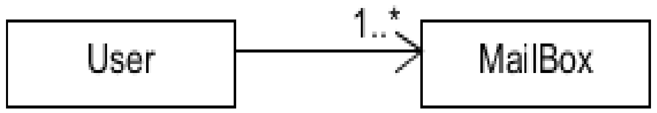




Two-way Association



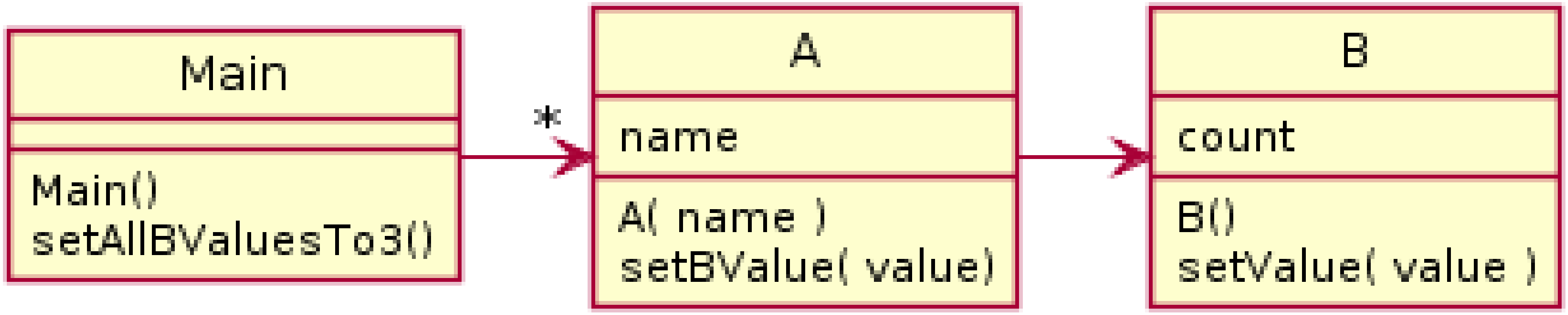
Two-Way Dependency



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

Let’s try to code a simple UML diagram!

- Try to make the classes and fields first!
- You can create empty methods and leave TODOs
- Implement the methods as the **last** thing you do.



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**

The principles go from most important to least important. Today's focus:

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**

An object oriented design must work!

Make sure all the data that you need is stored somewhere

- And that you can access it from the classes that need it
- The solution is not to keep 2 copies of the same data.

A good object oriented design is structured around the data

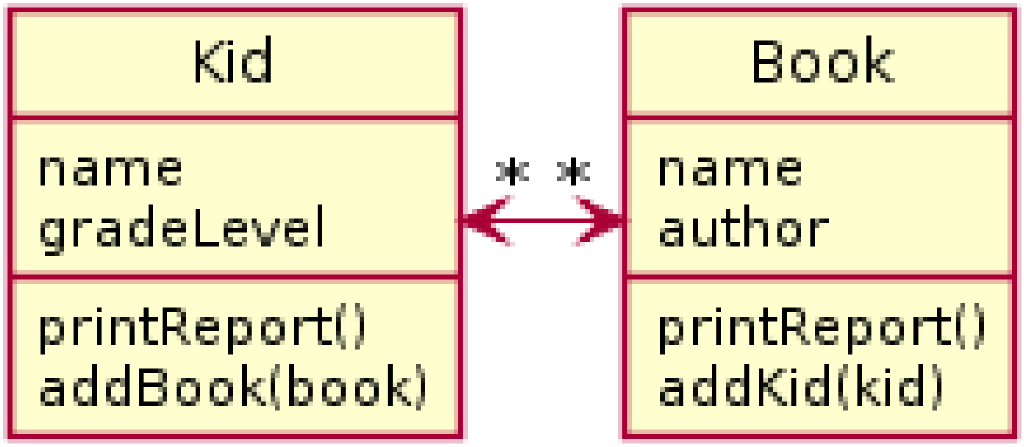
- Look for the nouns in your problem, consider making them classes
  - …if they are complex enough
- Put the data you need to store as fields in your classes
- Add operations to the classes to accomplish what your need
- Avoid Plural Nouns

An Example Problem

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. It should be possible to print a report on a kid that includes the books a particular kid has read.

Make a UML diagram of your proposed design for this system.

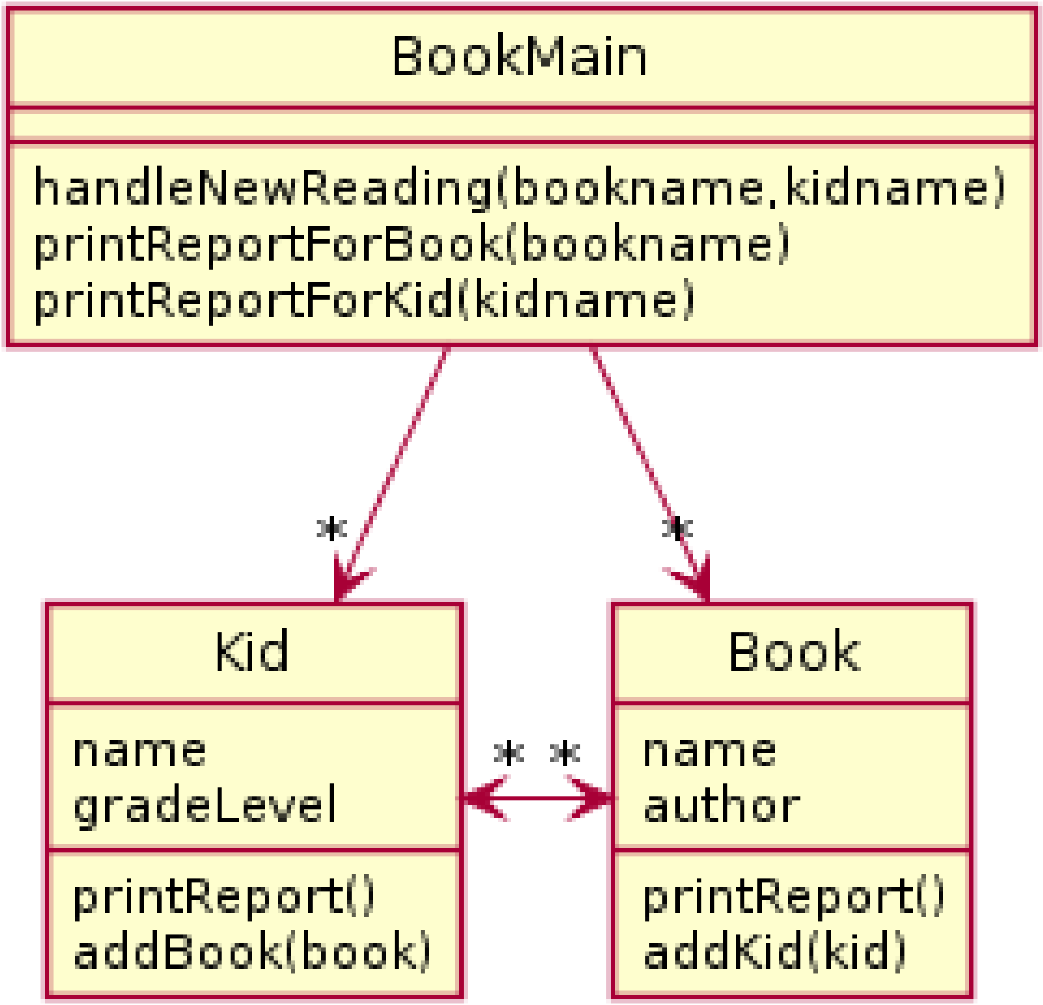
Basic solution



Note that List<Book> isn't listed by name as an instance variable of Kid, but the line from Kid to Book with the * implies that. Ditto for List<Kid> in book, since the arrow is double-ended with * on each end

Main class

- In really small programs, you could just have them as local variables in a static main
- But for larger programs, it's more usual for the class with main to be a real class with fields (also aids testing)
- In our very simple designs, this class also deals with user input
- Also be sure your design shows where things start and how use commands are handled

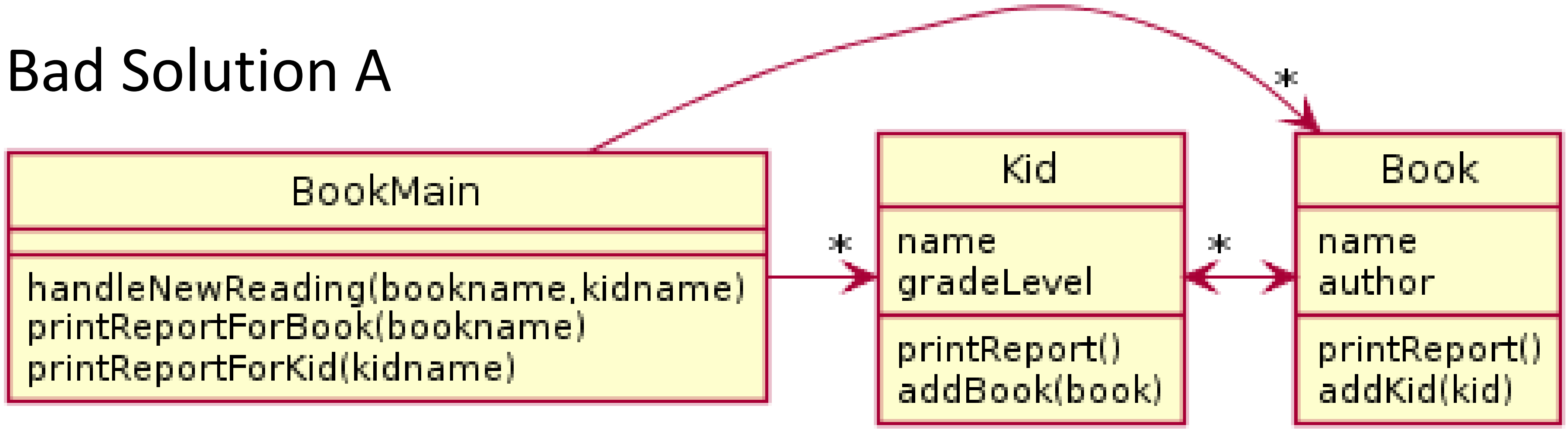


Today’s Focus

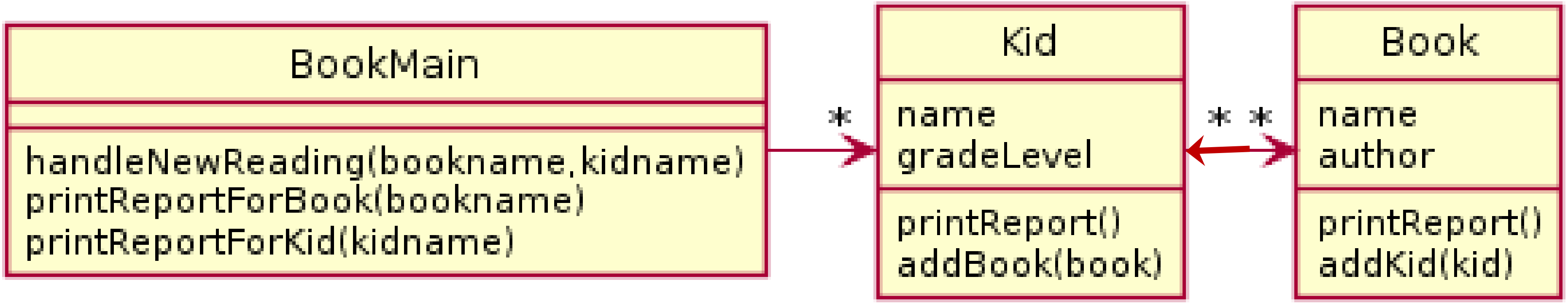
1. Structure your program around the data that needs storing
   - Nouns become your classes, operations become their methods
2. Your structure needs to function correctly
   - Every class must have access (directly or indirectly) to the data it needs to complete its operations
   - Usually this means the problem must be modeled correctly
   - Data should also not be duplicated

Consider

Bad Solution A



Bad Solution B

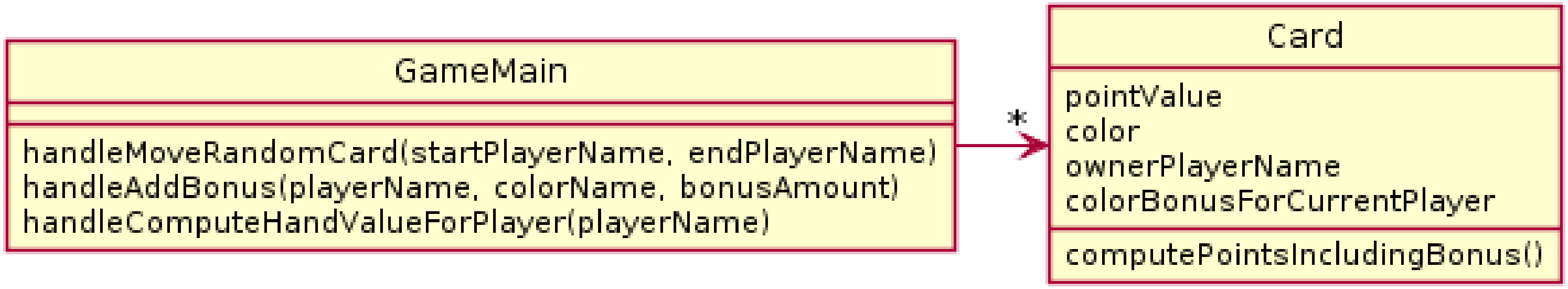


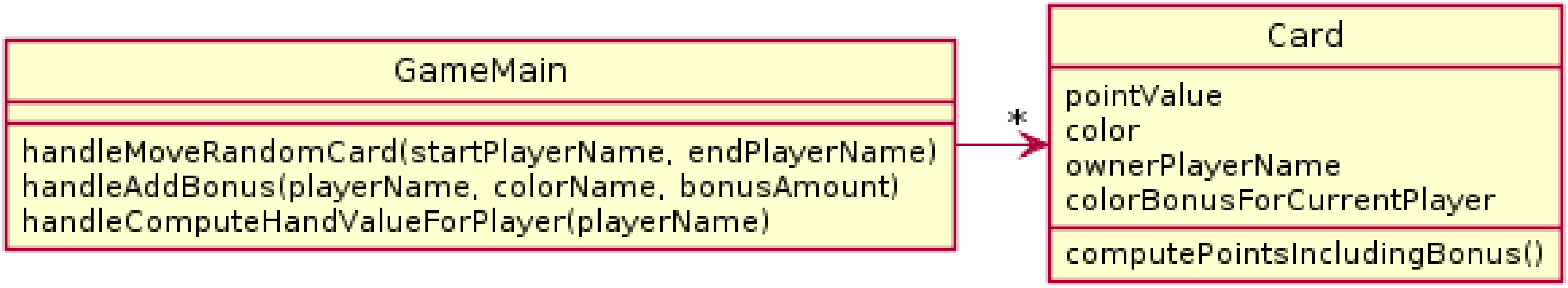
In most cases non-workable design is caused by…

- 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)

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.


GameMain
handleMoveRandomCard(startPlayerName, endPlayerName)
handleAddBonus(playerName, colorName, bonusAmount)
handleComputeHandValueForPlayer(playerName)
*
Card
pointValue
color
ownerPlayerName
colorBonusForCurrentPlayer
computePointsIncludingBonus()


GameMain
handleMoveRandomCard(startPlayerName, endPlayerName)
handleAddBonus(playerName, colorName, bonusAmount)
handleComputeHandValueForPlayer(playerName)
*
Card
pointValue
color
ownerPlayerName
colorBonusForCurrentPlayer
computePointsIncludingBonus()

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

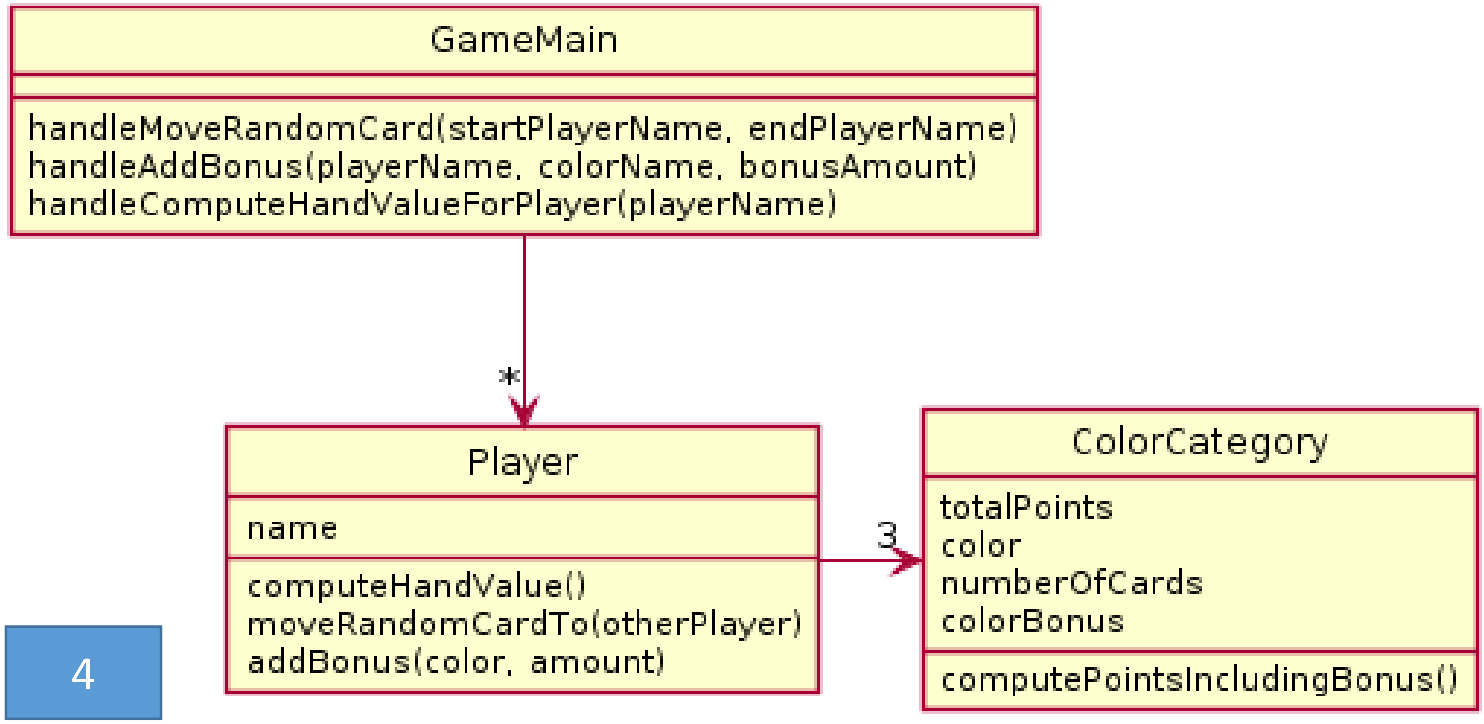
It requires iterating over all objects to get the full set of cards in the players hands to move cards or compute final total

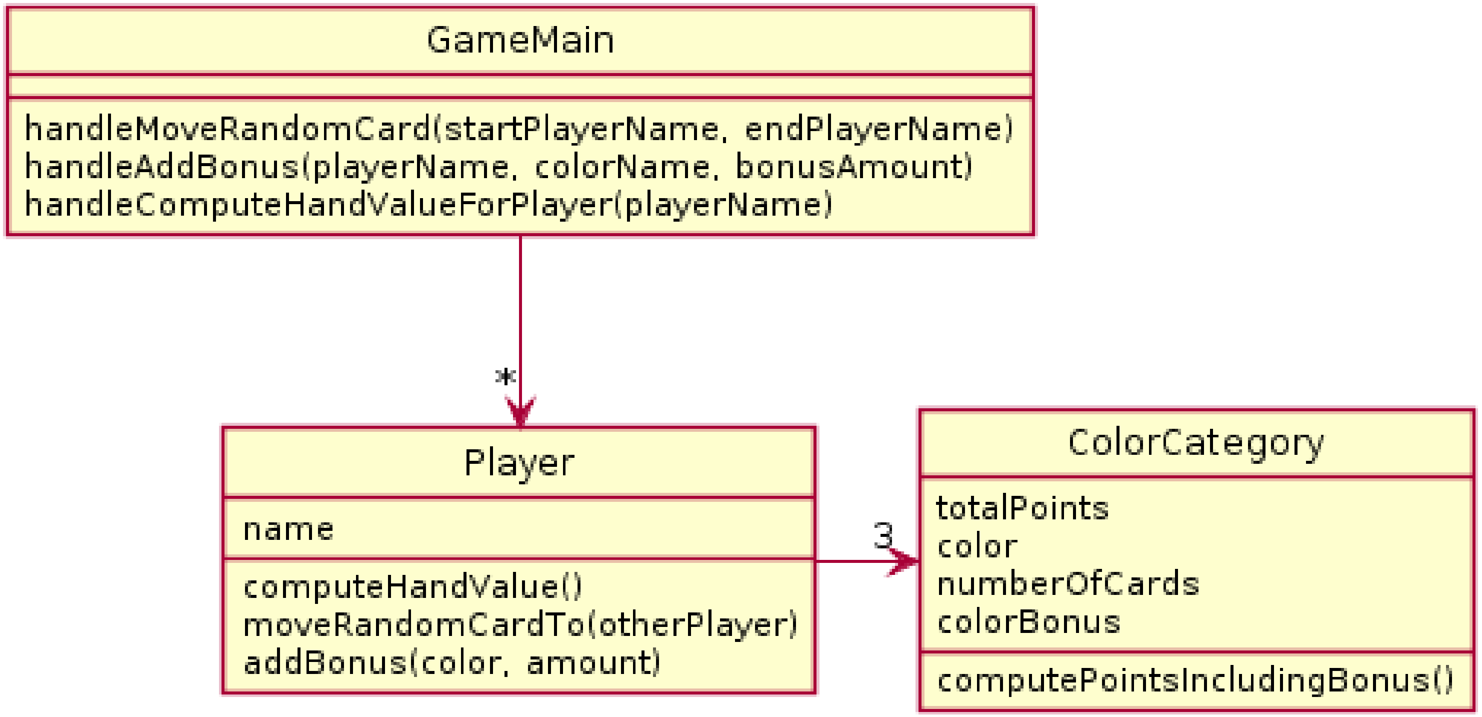
1c. Playername & player color bonus are duplicated across cards

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.


GameMain
handleMoveRandomCard(startPlayerName, endPlayerName)
handleAddBonus(playerName, colorName, bonusAmount)
handleComputeHandValueForPlayer(playerName)
*
Player
name
computeHandValue()
moveRandomCardTo(otherPlayer)
addBonus(color, amount)
3
ColorCategory
totalPoints
color
numberOfCards
colorBonus
computePointsIncludingBonus()


GameMain
handleMoveRandomCard(startPlayerName, endPlayerName)
handleAddBonus(playerName, colorName, bonusAmount)
handleComputeHandValueForPlayer(playerName)
*
Player
name
computeHandValue()
moveRandomCardTo(otherPlayer)
addBonus(color, amount)
3
ColorCategory
totalPoints
color
numberOfCards
colorBonus
computePointsIncludingBonus()

My answer (in order of importance)

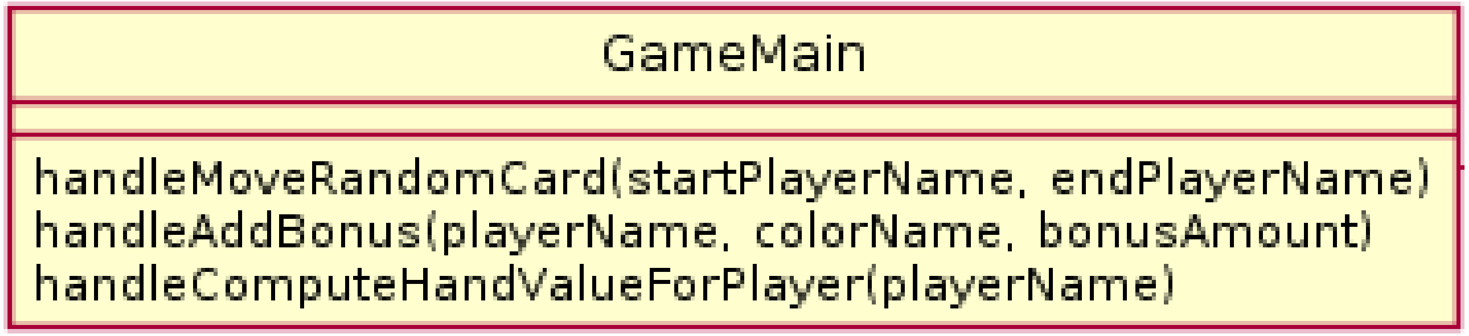
1a. The design does not function correctly

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

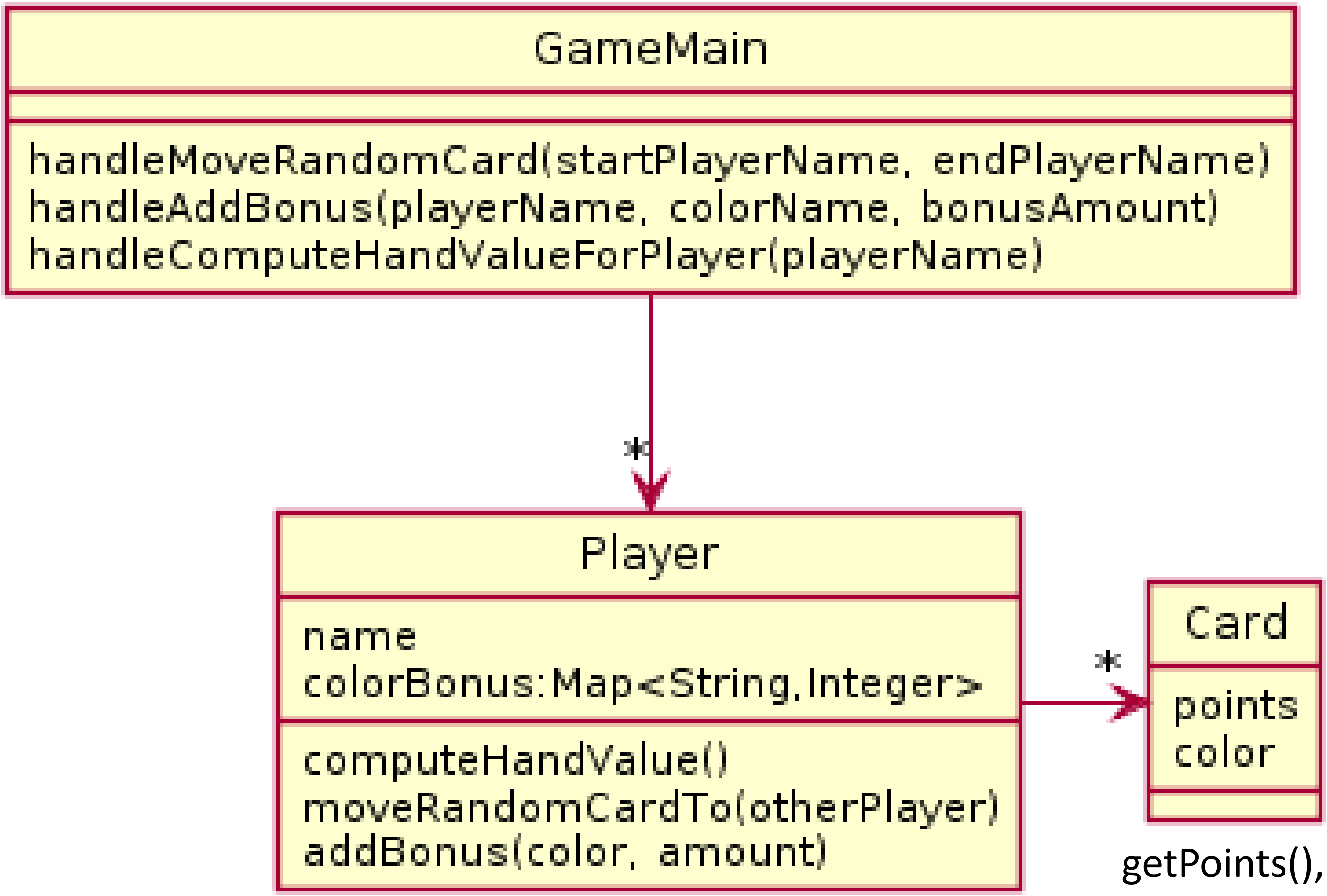
2a. Card (common noun from problem) not represented

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.


GameMain
handleMoveRandomCard(startPlayerName, endPlayerName)
handleAddBonus(playerName, colorName, bonusAmount)
handleComputeHandValueForPlayer(playerName)

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

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
- Problem 1 is also associated with ImplementingDesign1 homework
  - Due **Thursday night**. (We will go over a solution Wednesday)
  - You will have the option to implement your own design or out solution
  - ImplementingDesign1 description
- We'll discuss more design principles next class

A problem (if we have time)

A factory sells a small number of unique products. Each product has an id code, a description, price and quantity (the amount currently available at the factory). When a customer places an order, they buy a specific number of each product. The order needs to be stored in the system for future reference, with the customer's name and address.

At some point, the order should ship to the customer, and that date should also be recorded.

The main operation of the system is to add a new order and mark an order as shipped.

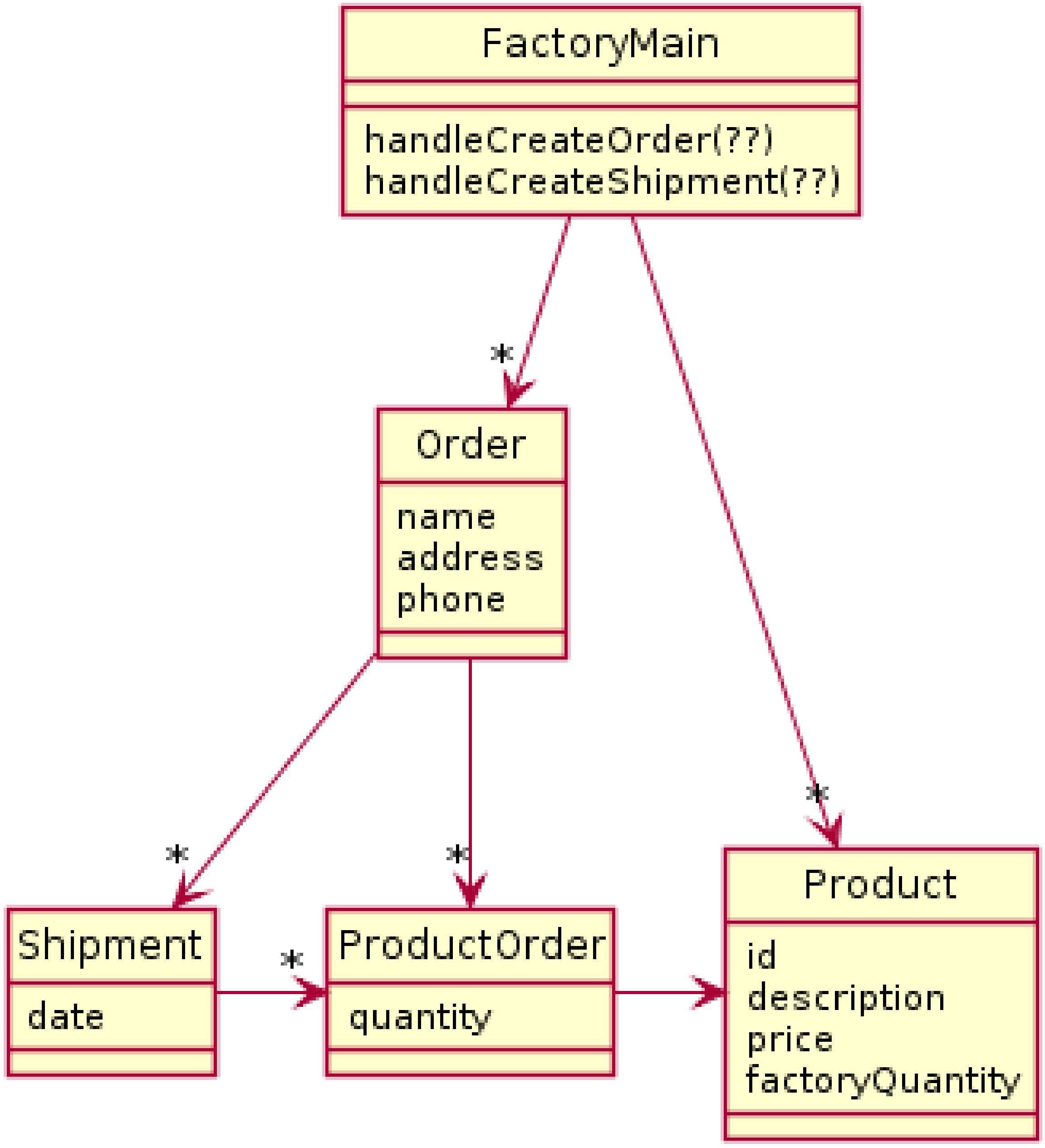
In a group of 2-3, make with an object design for this system and document it in UML (on paper).

A problem –revised

Now orders can be partially shipped – i.e. a single order might take several shipments to complete.

The main operation of the system is to add a new order and enter shipments for orders.

In a group of 2-3, revise your design to accommodate this new issue.


FactoryMain
handleCreateOrder(??)
handleCreateShipment(??)
*
*
Order
name
address
phone
*
*
Shipment
date
*
ProductOrder
quantity
Product
id
description
price
factoryQuantity