CSSE 220

A Software Engineering Technique:
(Class Diagrams)

Download FirstOODesignPractice from SVN

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

ClassName
Field names
Method names

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

Example

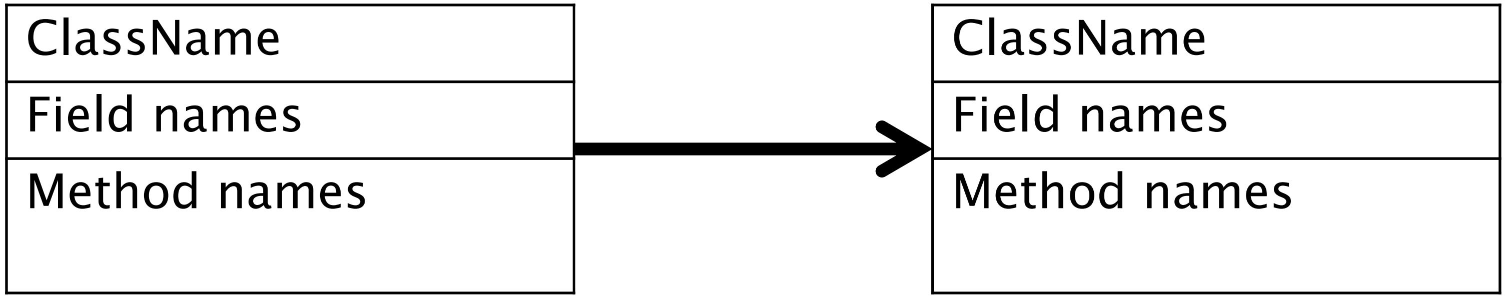
Team
teamAverage name students

addGrade(grade)
getTeamAverage()

Student
grades name
addGrade(grade)

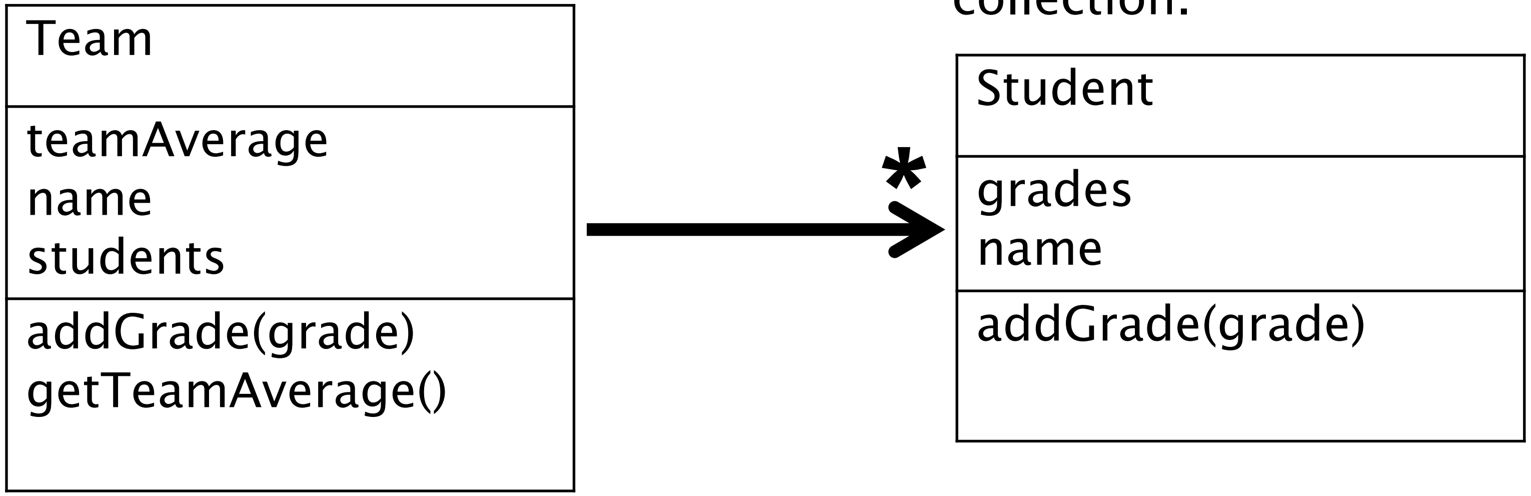
Lines

A has a B (field)



Example

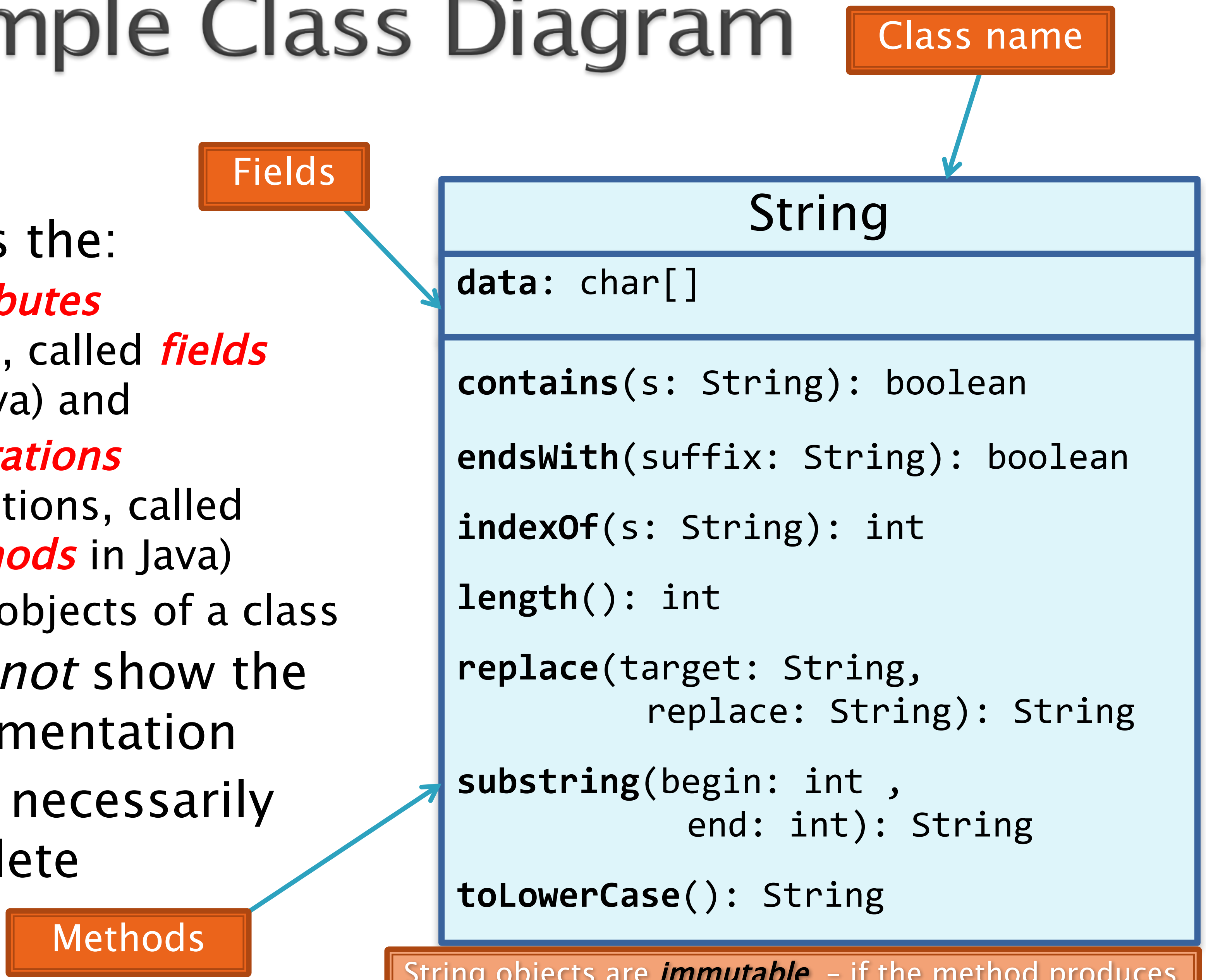
Note the star means several… usually a list or collection.



Example Class Diagram

- Shows the:
  - ***Attributes*** (data, called ***fields*** in Java) and
  - ***Operations*** (functions, called ***methods*** in Java)

  of the objects of a class
- Does *not* show the implementation
- Is *not* necessarily complete

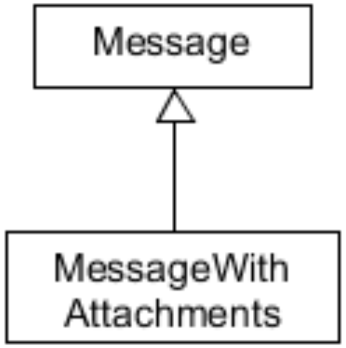


```
String
-----------------------------------------
data: char[]
-----------------------------------------
contains(s: String): boolean

endsWith(suffix: String): boolean

indexOf(s: String): int

length(): int

replace(target: String,
         replace: String): String

substring(begin: int ,
           end: int): String

toLowerCase(): String
```

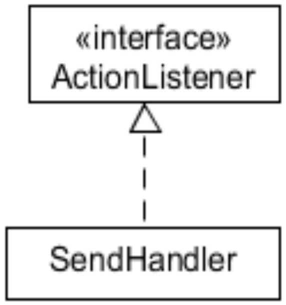
String objects are ***immutable*** – if the method produces a String, the method *returns* that String rather than mutating (changing) the implicit argument

Summary of UML Class Diagram Arrows

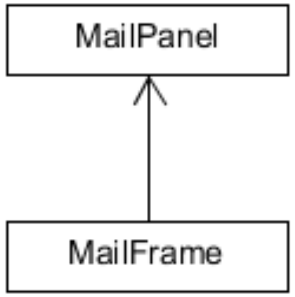
Inheritance (is-a)



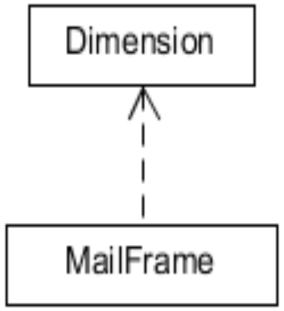
Interface Implementation (is-a)

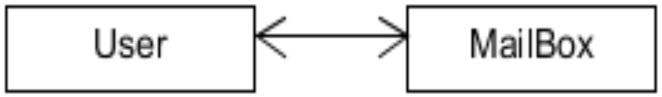


Association (has-a-field)

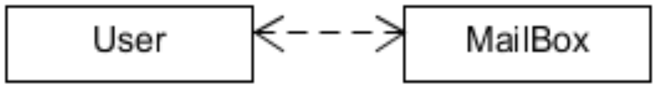


Dependency (depends-on)

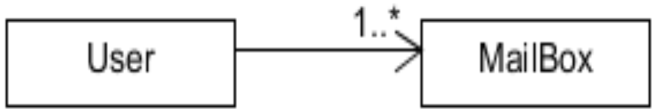




Two-way Association



Two-Way Dependency



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

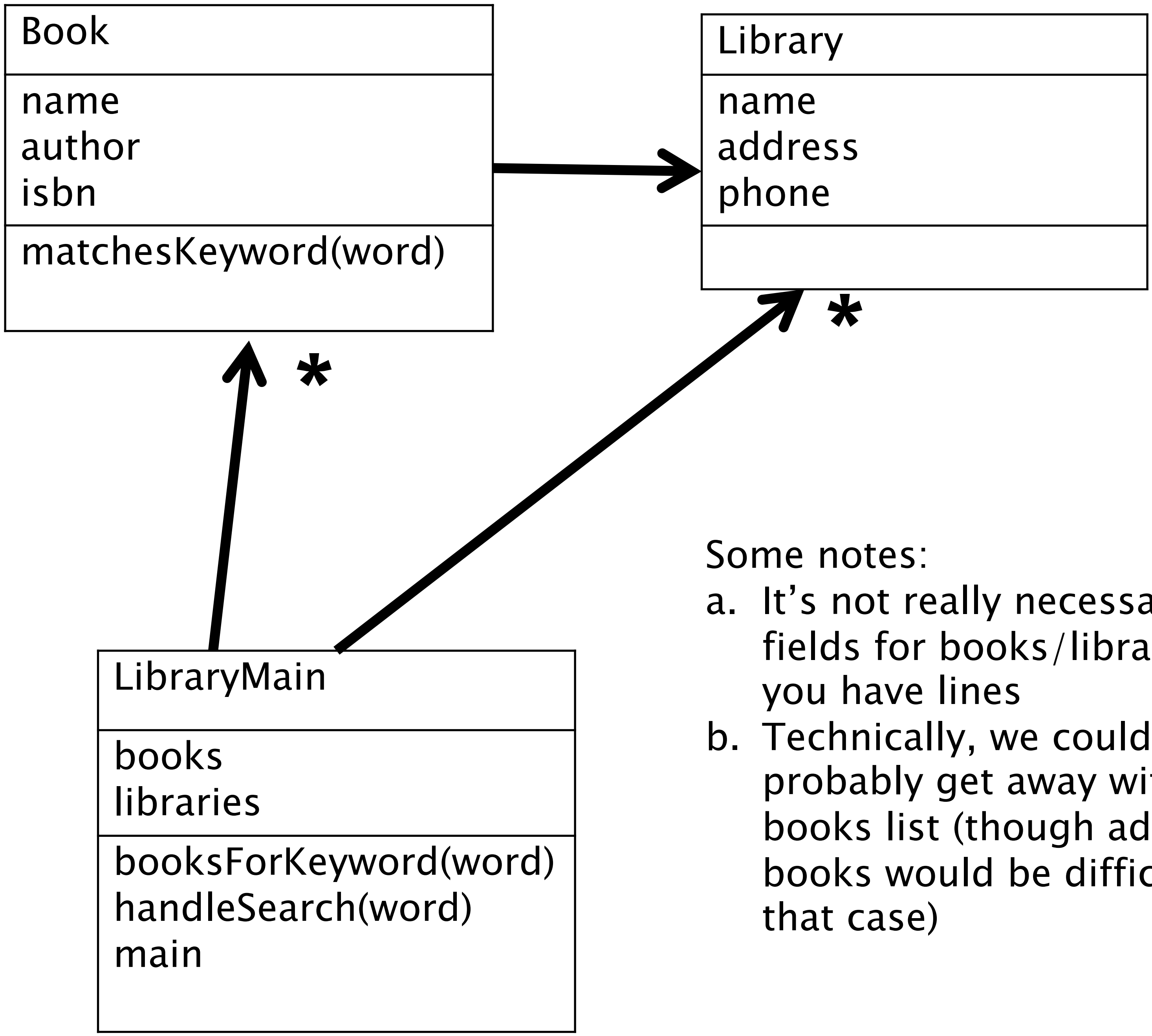
A problem

We want to have a system that lets us search the catalogs of various nearby libraries for books.

We need to track the names, phone numbers, and addresses of the libraries. We need to track the titles, authors, and ISBN numbers of the books.

The main operation of the system is to search for a particular book by keyword (could be author, title, or ISBN), and get a listing of its info and the library that has it.

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

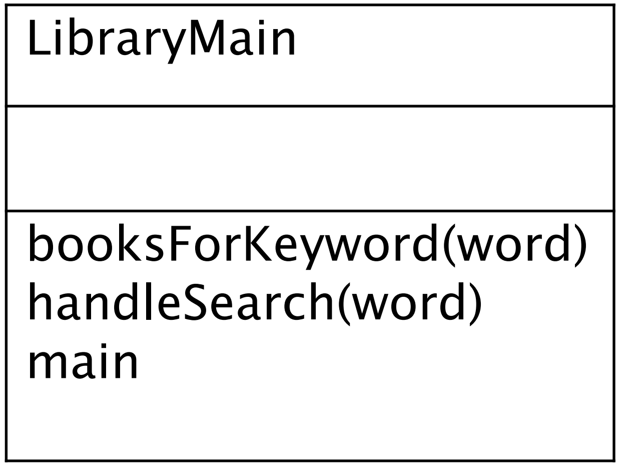


Some notes:

a. It's not really necessary to list fields for books/libraries *if* you have lines
b. Technically, we could probably get away with just a books list (though adding new books would be difficult in that case)

Main class

- Your design might have just included classes Book and Library – but remember all books and libraries must be stored somewhere
- 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 user commands are handled



3 Things…

- The “things” of what you’re describing usually become the classes
  - The verbs usually become methods of the classes
- Avoid using plurals
  - We make an ArrayList of Face objects, not Faces.
- Make it work!
  - Go through it with some “use case” in mind and make sure that when this object is created, you can accomplish that case. Otherwise, redesign that design until it “works!!!”

Good Classes Typically

- Come from nouns in the problem description
- May…
  - Represent single concepts
    - `Circle`, `Investment`
  - Represent visual elements of the project
    - `FacesComponent`, `UpdateButton`
  - Be abstractions of real-life entities
    - `BankAccount`, `TicTacToeBoard`
  - Be actors
    - `Scanner`, `CircleViewer`
  - Be utility classes that mainly contain static methods
    - `Math, Arrays, Collections`

What Stinks? Bad Class Smells*

- Can’t tell what it does from its name
  - `PayCheckProgram`
- Turning a single action into a class
  - `ComputePaycheck`
- Name isn’t a noun
  - `Interpolate`, `Spend`

*See http://en.wikipedia.org/wiki/Code_smell
http://c2.com/xp/CodeSmell.html

A problem

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.

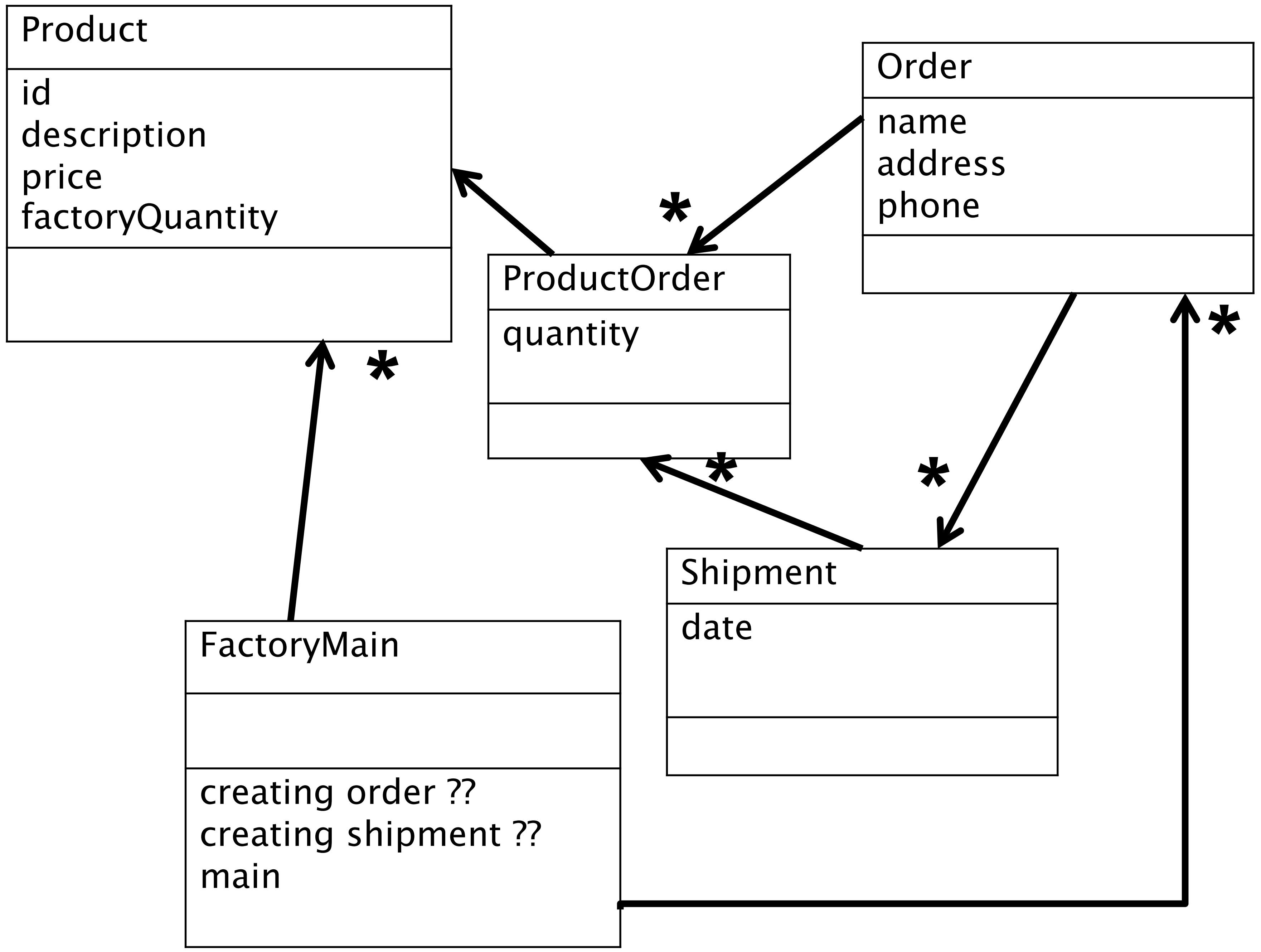
In a group of 2-3, make with an object design for this system and document it in UML (on paper).

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.


Product
id
description
price
factoryQuantity
Order
name
address
phone
ProductOrder
quantity
Shipment
date
FactoryMain
creating order ??
creating shipment ??
main
*
*
*
*
*

OrderTaker (individual)

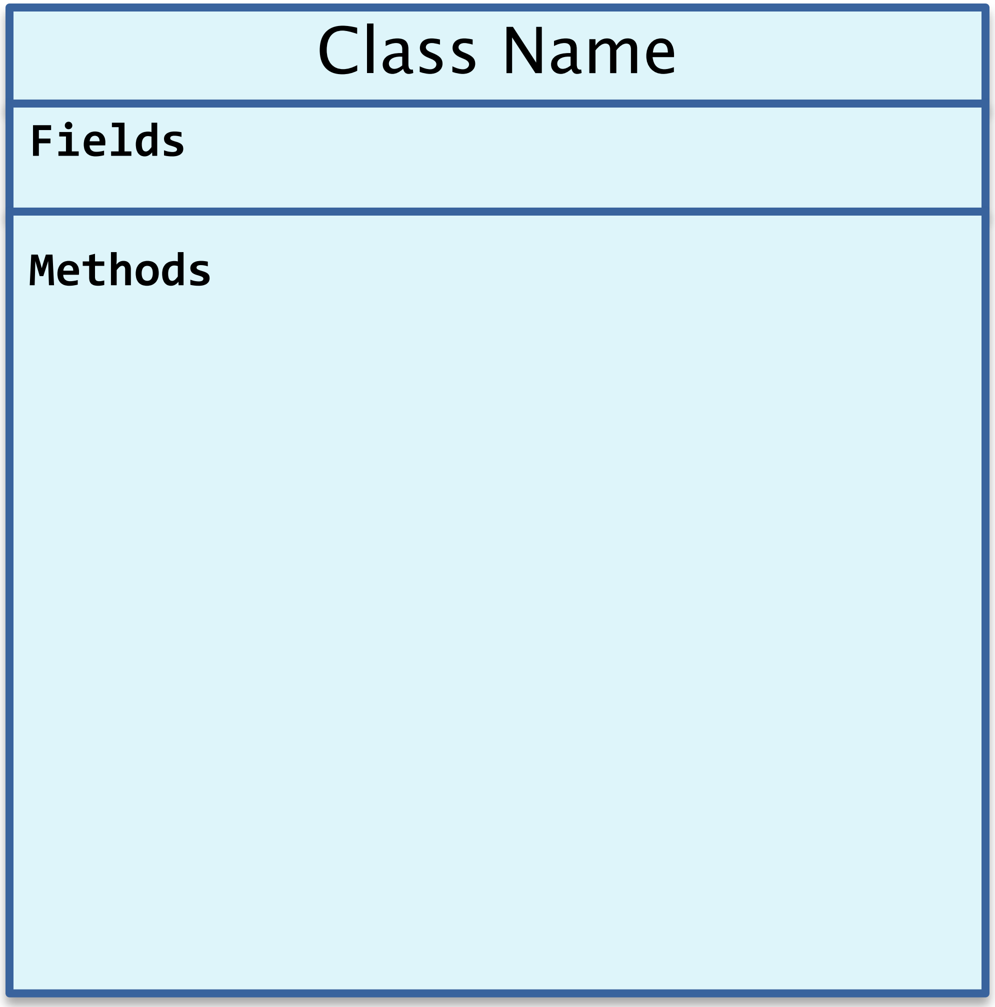
Design it
Let me or a TA look at it
Implement it in code

Work with your groups of 3/4

- Decide what classes ought to be in the system and what methods/fields those classes should have (your design should have at least 2 classes)
- Don't forget one class needs to have a main method
- Make sure your design works!
- Write down your answers on a piece of paper with all of your team's names on it
- Call me over when you think you're done – then you'll implement it

Exercise: Class Diagrams

- Task: Make Class diagrams for the Invoice example from OrderTaker



Blackjack – Work with groups of 4

- Decide what classes ought to be in the system and what methods/fields those classes should have (your design should have at least 3 classes)
- Don’t forget one class needs to have a main method
- Make sure your design works!
- Write down your answers on a piece of paper with all of your team’s names on it
- Call me over when you think you’re done
- This will be your quiz for today – I’ll be lenient