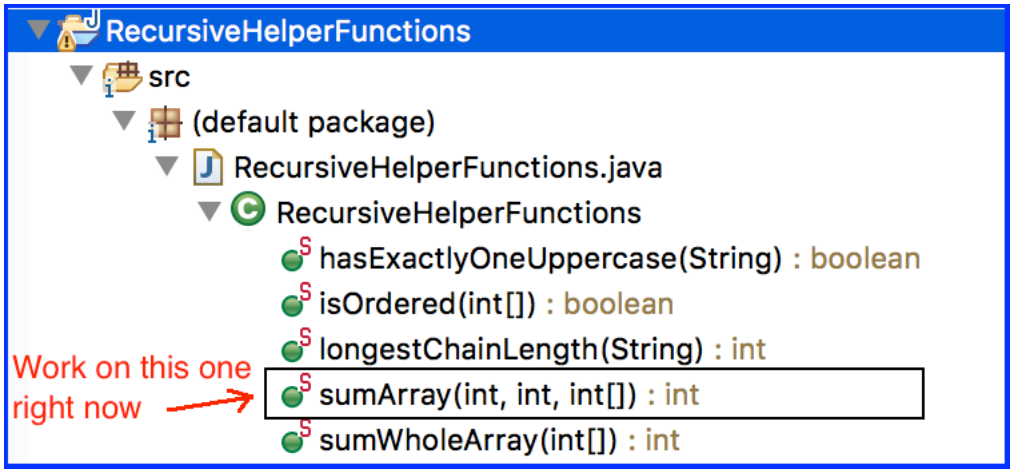
CSSE 220

More interfaces
More recursion
More fun?

Check out *RecursiveHelperFunctions* and *BettingInterfaces* from repo

Exercise time

Solve the sumArray function – use recursion



5 minutes for this

Talk out the details with a partner, but each of you should get the code working on your own machine

Recursive Helper Functions – What, When, Why, How?

What:

- A recursive function that is called by another (non-recursive) function
- The non-recursive *top-level* function (the caller) doesn't do much – it sets up the first call to the recursive operation

Recursive Helper Functions – When?

When:

- Additional parameters are needed
  - Often the initial function you're given is not in the ideal form for a recursive solution
- Return values need to be updated

Recursive Helper Functions –Why?

Why:

- Makes function (i.e., the non-recursive operation) called by external code cleaner/easier to use
  - Does not rely on caller to understand how to initialize the information for the helper
- Easier to understand by breaking problem down to smaller pieces

Recursive Helper Functions –How?

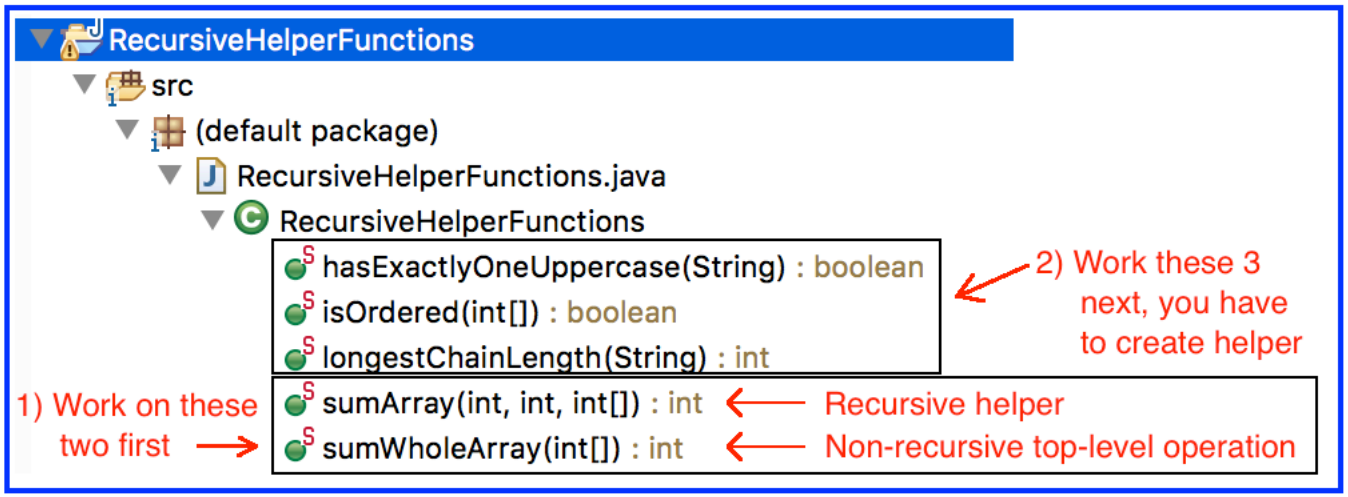
How:

- Methods named coolFunction & coolFunctionHelper
  - 90% (or more) of the code is in coolFunctionHelper
- coolFunction
  - Is the top-level operation
  - It is not recursive
  - Its job is to correctly call coolFunctionHelper with the correct parameters
- coolFunctionHelper
  - Is recursive
  - It takes the same parameter as coolFunction but no progress is made toward base-case on this parameter
  - An additional parameter is added and on this additional parameter is where progress toward base case is made

RecursiveHelperFunctions

20 minutes

1. Now solve sumWholeArray - make it call sumArray
2. Then solve the remaining problems



Memoization

- Save every solution we find to sub-problems
- Before recursively computing a solution:
  - Look it up
  - If found, use it
  - Otherwise do the recursive computation
- Study the memoization code in the RecursiveHelperFunctions project

What if the recursive call isn't in the return?

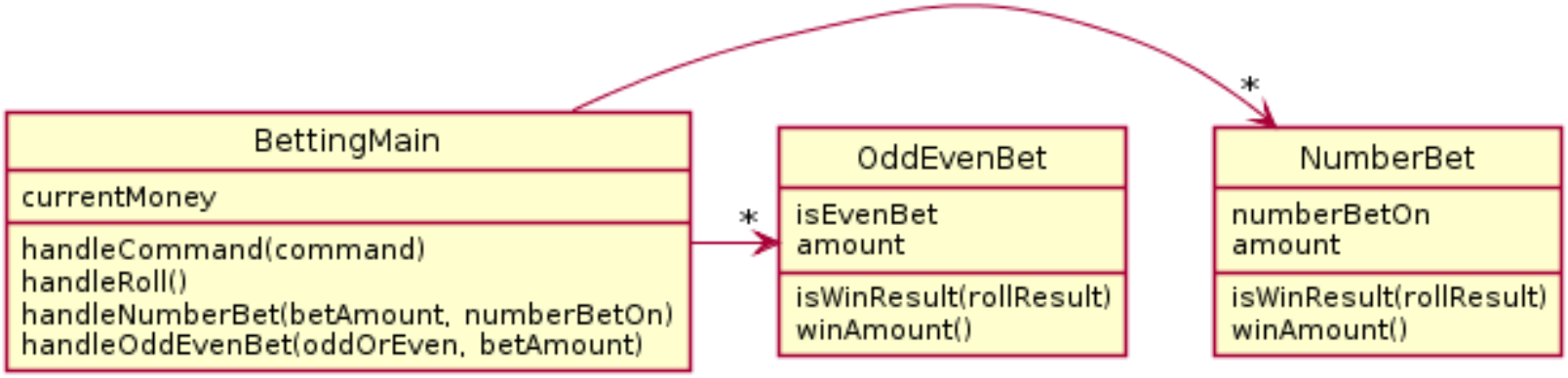
- Let's start the quiz problem together, then you can finish it on your own.

BettingInterfaces

- Demo in Eclipse
  - Show how to bet
  - Show how to create test cases

UML as it currently stands

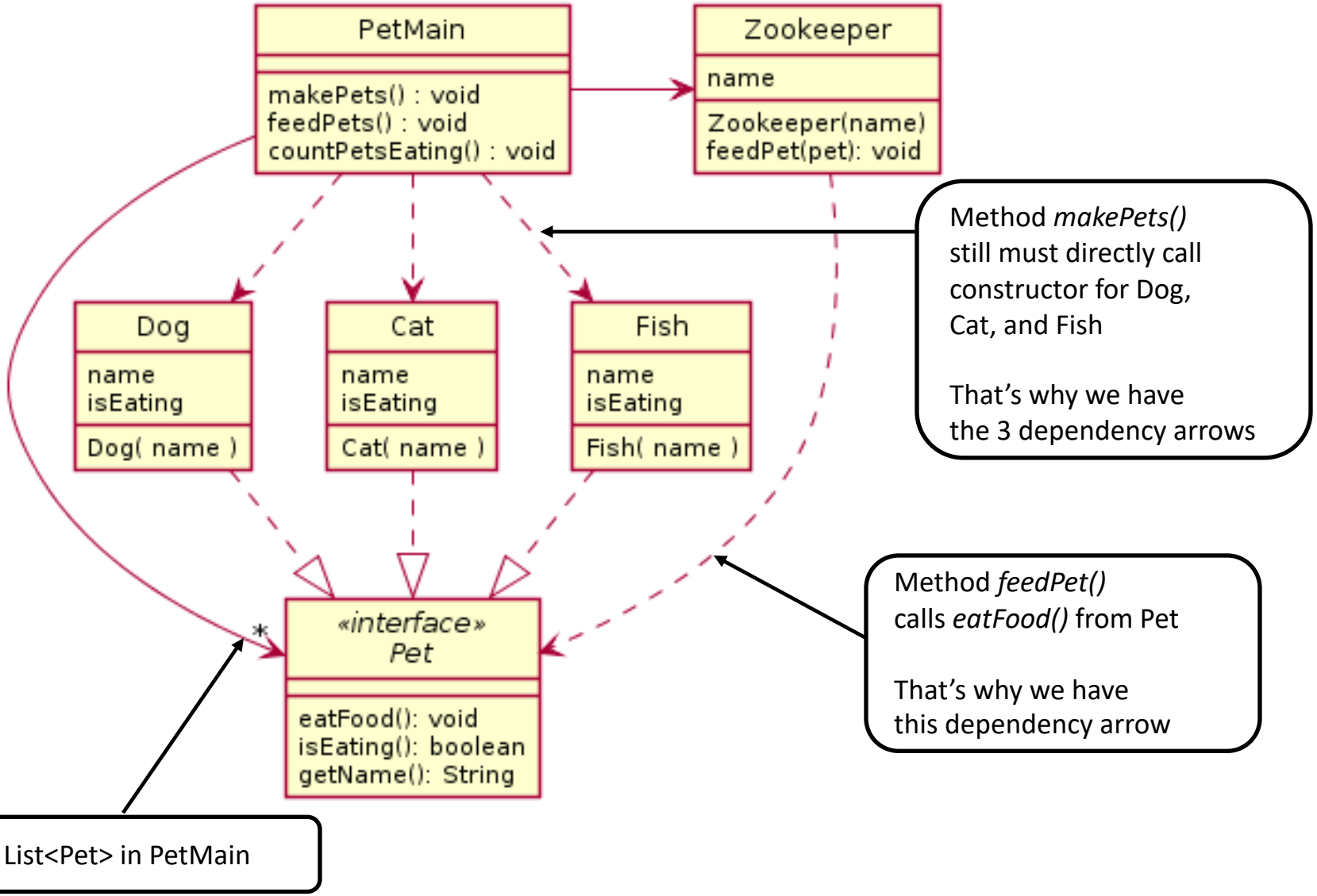
- What do you need to add?
- What do the Bet classes have in common?



BettingInterfaces

- Get in groups of 2-3…no one working alone
- Understand the given code, the duplication, plus the additional features you will be adding. Look at 3 TODOs in BettingMain.
- Set up some test cases in a text file
- Design a solution for all 3 TODOs using interfaces and make a UML diagram describing it (page 2 of in-class quiz)
- Get myself or a TA to check out your UML
- Once we sign off – start coding
  - You only need 1 computer for this one.
  - I recommend you do each TODO one by one rather than doing everything in one go

Solution From PetMain



Hints

1) Your interface will likely be called Bet
2) You should have 3 classes implementing Bet, one for each of the current types of bets in the code, one for the new one you're being asked to implement
3) You'll need to update the lists in main to a single ArrayList<Bet> (or some other storage method to main)