

CSSE 220

More interfaces

More recursion

More fun?

Check out *RecursiveHelperFunctions* and *BettingInterfaces* from SVN

Exercise time

- Solve the `sumArray` function recursively
 - It's in the *RecursiveHelperFunctions* project
- You can work with friends, but each of you should get the code working on your own computer

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

- What:
 - A recursive function that is called by another (non-recursive) function
 - The non-recursive function (the caller) doesn't do much
- 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 – What, When, Why, How?

- Why:
 - Makes function 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
- How:
 - Methods named `coolFunction` & `coolFunctionHelper`
 - 90% of the code is in `coolFunctionHelper`

RecursiveHelperFunctions

- Solve the remaining problems
 - **all the problems will require you to create a recursive helper function**
- You can work with a friend but make sure both of you write the code

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?

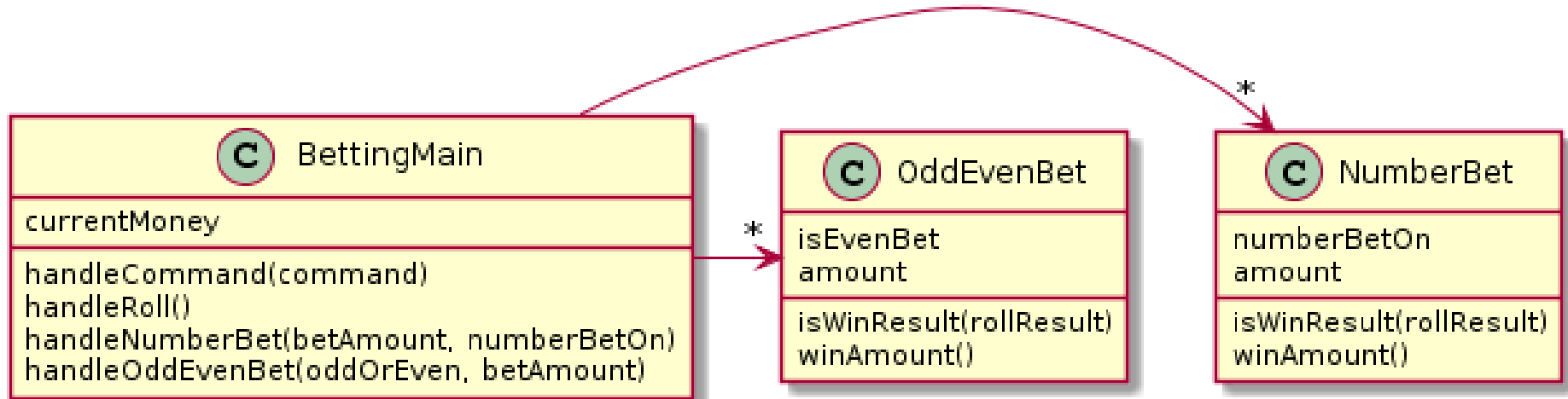
- Let's start the quiz problem together, then you can finish it on your own.

BettingInterfaces

- 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.
- Design a solution for all 3 TODOs using interfaces and make a UML diagram describing it
- 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

UML as it currently stands

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



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)