# CSSE 220

More interfaces
More recursion
More fun?

Import *RecursiveHelperFunctions* and *BettingInterfaces* from the repo

## Review

- Object Oriented Design Principles
- Encapsulation
- Cohesion
- Coupling
- Review Solar System Problem
- Static Variables

# 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**

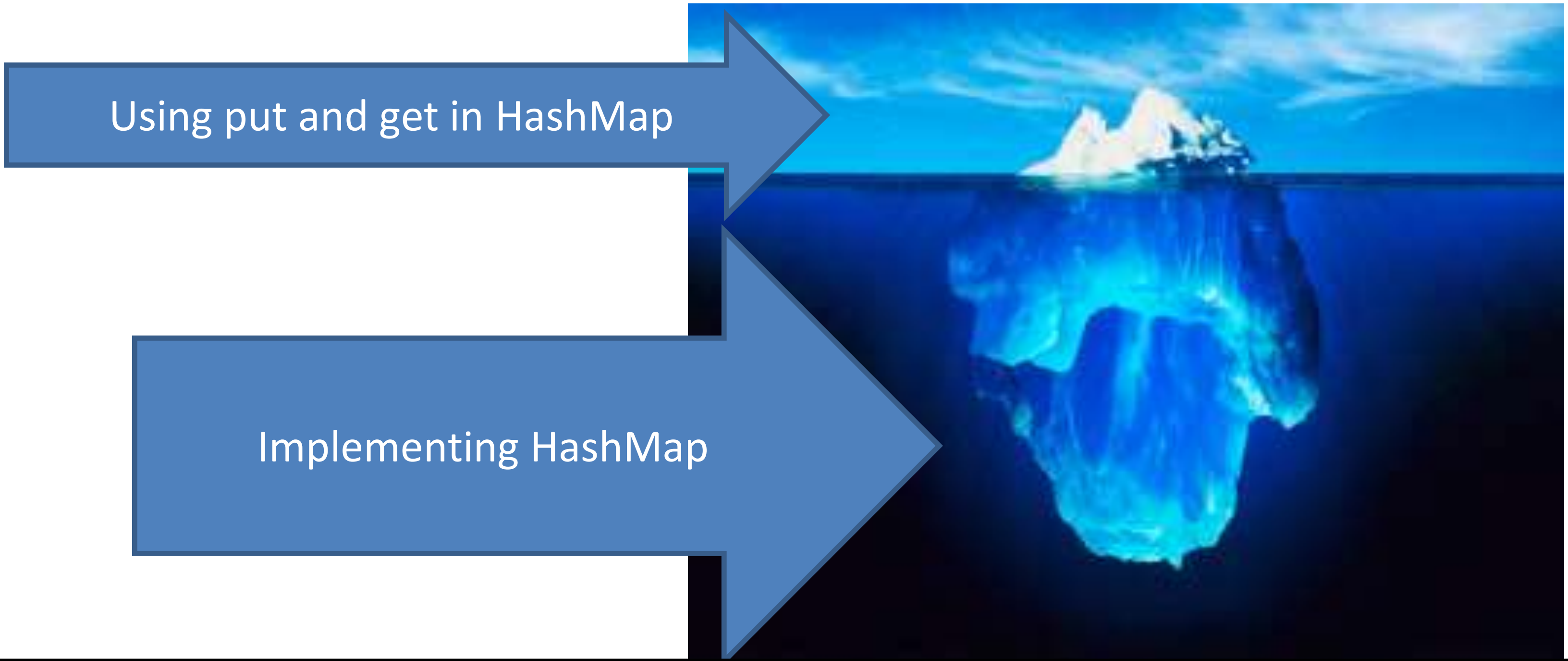
# Encapsulation

- Makes your program easier to understand by
  - Grouping related stuff together

- Rather than passing around data, pass around objects that:
  - Provide a powerful set of operations on the data
  - Protect the data from being used incorrectly

## Encapsulation

- Makes your program easier to understand by…
  - Saving you from having to think about how complicated things might be



## Coupling and Cohesion

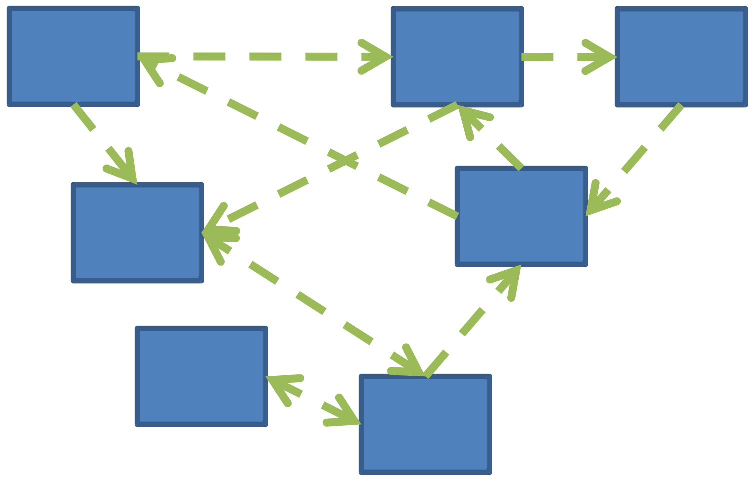
- Two terms you need to memorize
- Good designs have:
  - High coHesion
  - Low coupLing

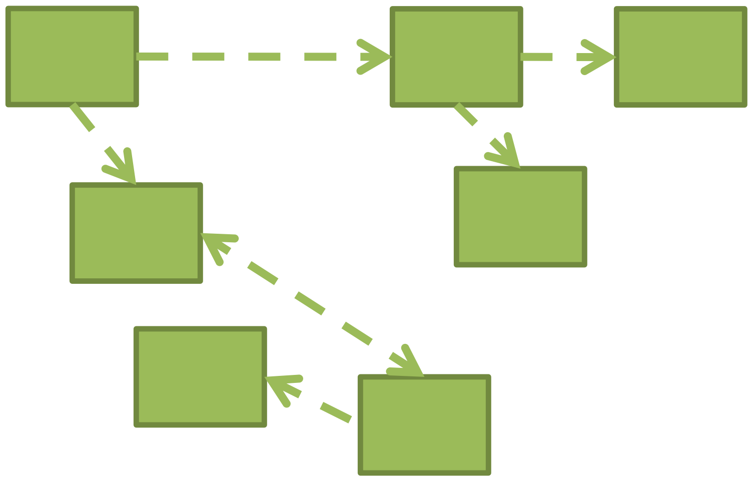
Consider the opposite:

- Low cohesion means that you have a small number of really large classes that do too much stuff (i.e., do more than one thing)
- High coupling means you have many classes that depend ("know") too much on each other

# Coupling – UML Diagrams

- Lot's of dependencies ➔ high coupling
- Few dependencies ➔ low coupling

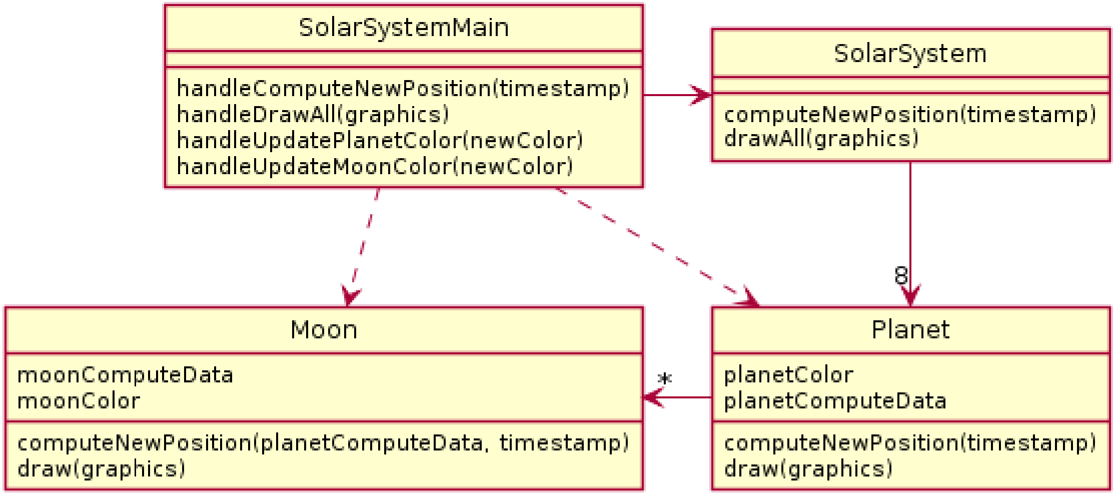


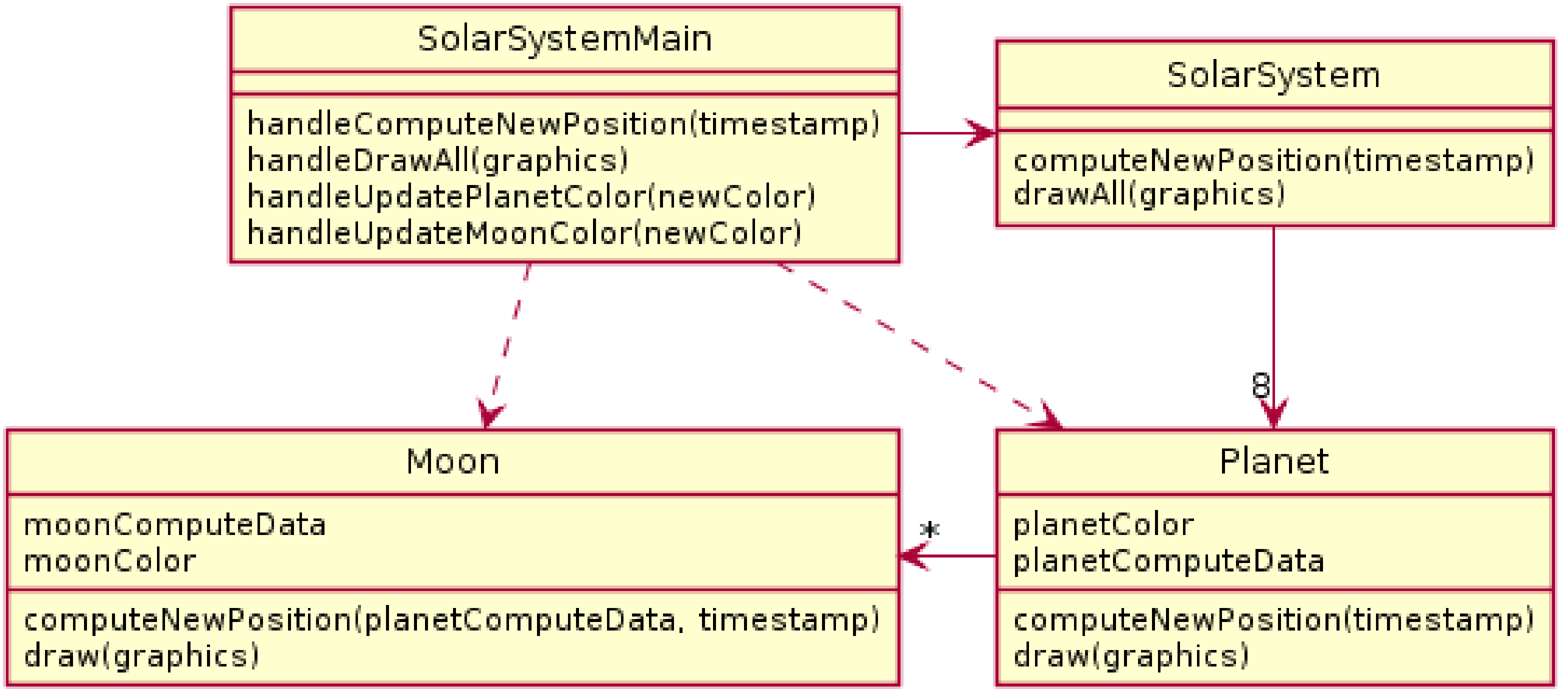


How hard will it be to change code with:
High coupling?   Low coupling?

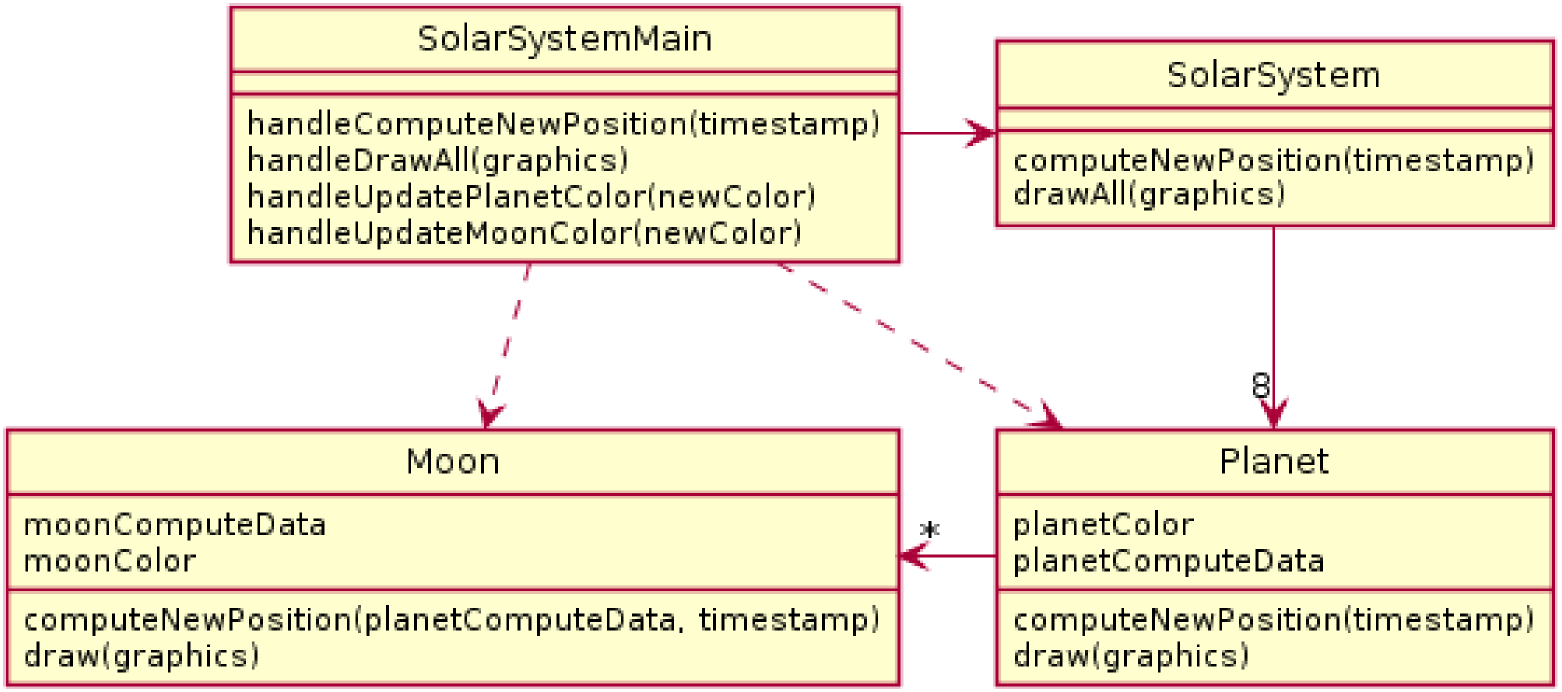
# Review: Solar System Problem

A Java program draws a minute by minute updated diagram of the solar system including all planets and moons. To update the moon's position, the moon's calculations must have the updated position of the planet it is orbiting. The diagram is colored - all planets are drawn the same color and all moons are drawn the same color. However, it needs to be possible to reset the planet color or the moon color and the diagram should reflect that.


SolarSystemMain
handleComputeNewPosition(timestamp)
handleDrawAll(graphics)
handleUpdatePlanetColor(newColor)
handleUpdateMoonColor(newColor)
SolarSystem
computeNewPosition(timestamp)
drawAll(graphics)
8
Moon
moonComputeData
moonColor
computeNewPosition(planetComputeData, timestamp)
draw(graphics)
*
Planet
planetColor
planetComputeData
computeNewPosition(timestamp)
draw(graphics)


SolarSystemMain
handleComputeNewPosition(timestamp)
handleDrawAll(graphics)
handleUpdatePlanetColor(newColor)
handleUpdateMoonColor(newColor)
SolarSystem
computeNewPosition(timestamp)
drawAll(graphics)
8
Moon
moonComputeData
moonColor
computeNewPosition(planetComputeData, timestamp)
draw(graphics)
*
Planet
planetColor
planetComputeData
computeNewPosition(timestamp)
draw(graphics)

- What is wrong here?

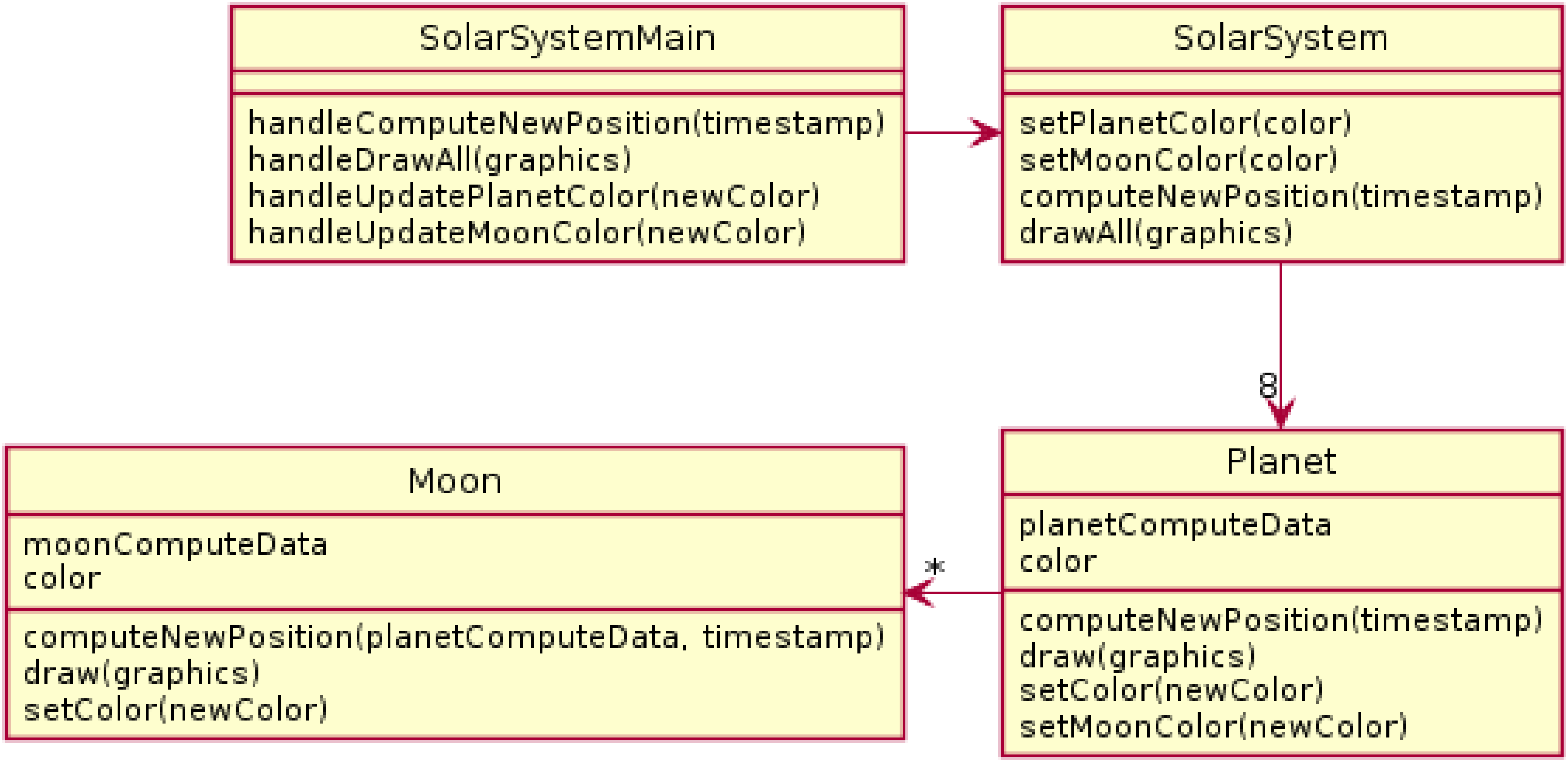

SolarSystemMain
handleComputeNewPosition(timestamp)
handleDrawAll(graphics)
handleUpdatePlanetColor(newColor)
handleUpdateMoonColor(newColor)
SolarSystem
computeNewPosition(timestamp)
drawAll(graphics)
8
Moon
moonComputeData
moonColor
computeNewPosition(planetComputeData, timestamp)
draw(graphics)
*
Planet
planetColor
planetComputeData
computeNewPosition(timestamp)
draw(graphics)

- What is wrong here?

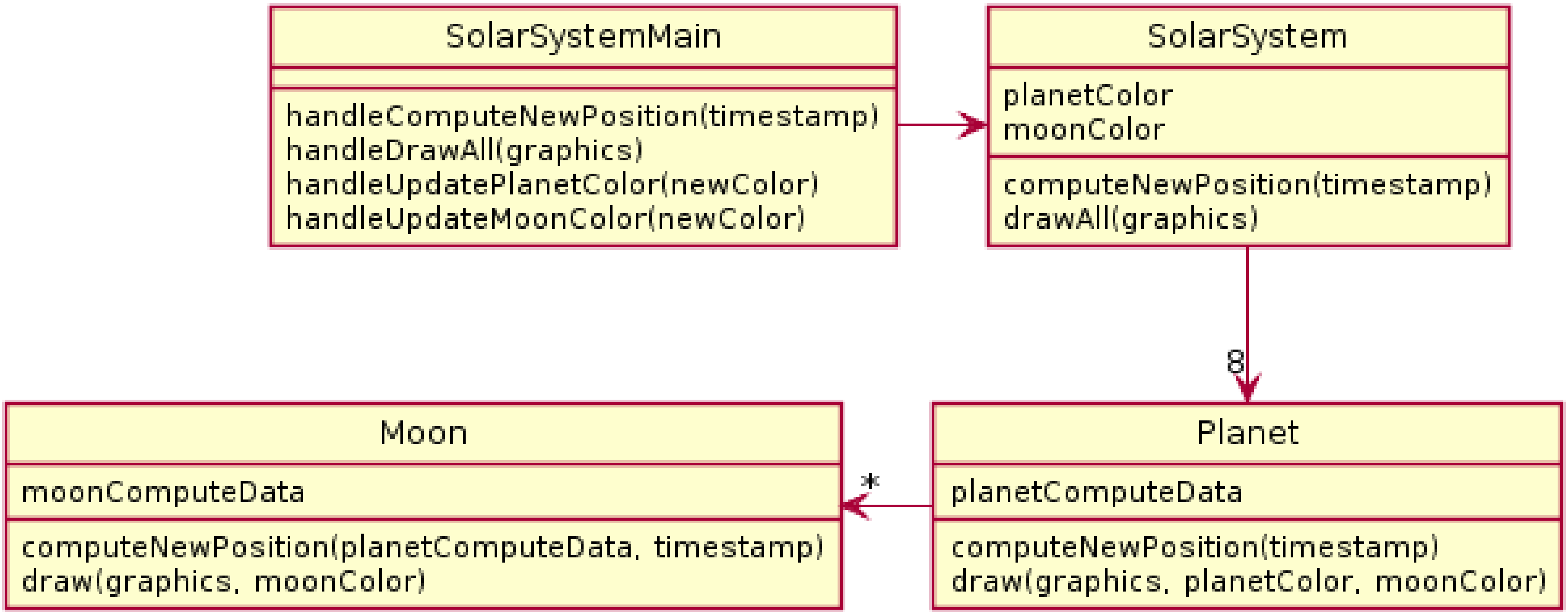
4b. methodChain to update moon

```
ss.getPlanets().get(0).getMoons().get(0).setColor(color);
```

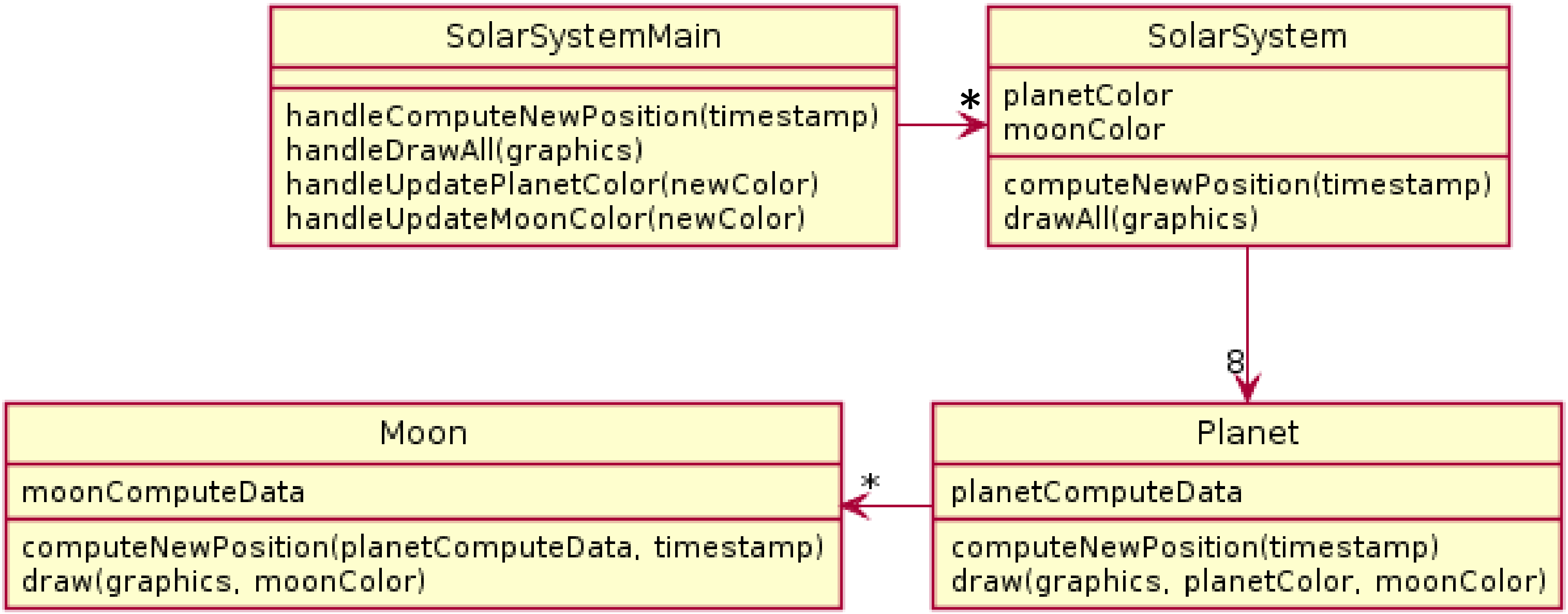
# Partial Solution



# Why not use static here?
# All moons, planets have same color!


SolarSystemMain
handleComputeNewPosition(timestamp)
handleDrawAll(graphics)
handleUpdatePlanetColor(newColor)
handleUpdateMoonColor(newColor)
SolarSystem
planetColor
moonColor
computeNewPosition(timestamp)
drawAll(graphics)
8
Planet
planetComputeData
computeNewPosition(timestamp)
draw(graphics, planetColor, moonColor)
*
Moon
moonComputeData
computeNewPosition(planetComputeData, timestamp)
draw(graphics, moonColor)

# What if we had many solar systems? Would our design be easily extended?


SolarSystemMain
handleComputeNewPosition(timestamp)
handleDrawAll(graphics)
handleUpdatePlanetColor(newColor)
handleUpdateMoonColor(newColor)
*
SolarSystem
planetColor
moonColor
computeNewPosition(timestamp)
drawAll(graphics)
8
Planet
planetComputeData
computeNewPosition(timestamp)
draw(graphics, planetColor, moonColor)
*
Moon
moonComputeData
computeNewPosition(planetComputeData, timestamp)
draw(graphics, moonColor)

## Rule of Thumb: No Global Variables

- Or static variables that are used like globals
- A static variable can be accessed/modified in any function at any time
- As a result many parts of the code can be coupled to a single class

## Rule of Thumb: No Global Variables

- Or static variables that are used like globals
- A static variable can be accessed/modified in any function at any time
- As a result many parts of the code can be coupled to a single class

- Why?
- Increases coupling among all the clients that get or change value of the global variable

# 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

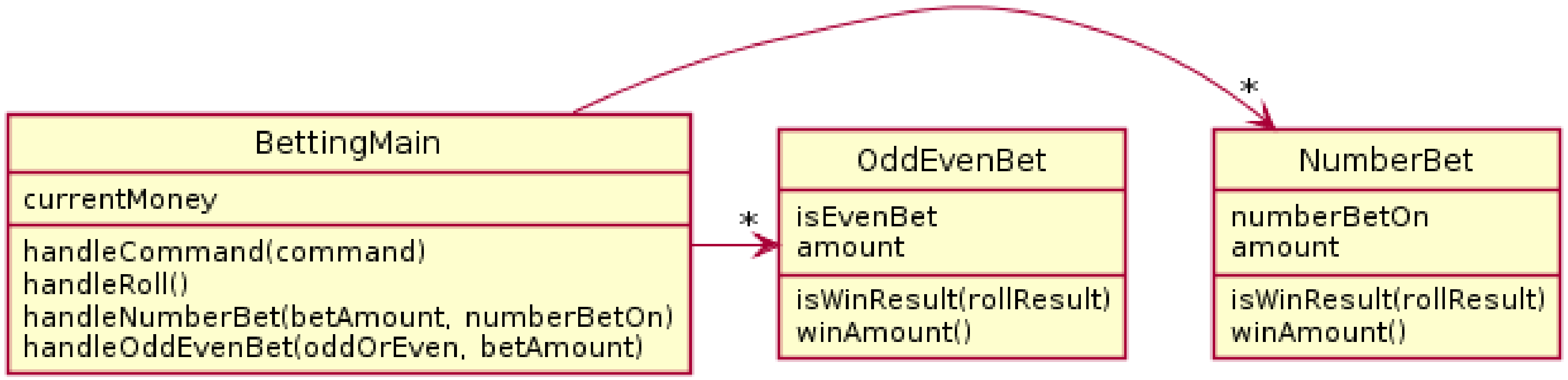
- Get in groups of 2-3…no one working alone
- Understand the given code, the duplication, plus the additional features you will be adding. Look at 3 TODOs in BettingMain.
- Design a solution for all 3 TODOs using interfaces and make a UML diagram describing it
- Get myself or a TA to check out your UML
- 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?

# 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)