CSSE 220

Software Engineering Techniques
Design Principles
Encapsulation

Today’s Agenda

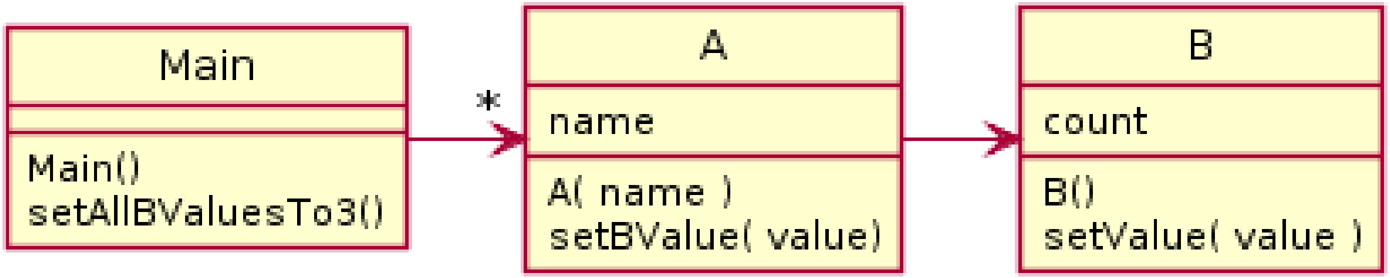
- How to submit your ImplementingDesign1
- Focus on more OO Design principles:
  - **Spread** functionality throughout the system
  - Encapsulation

Turning in UML for ImplementingDesign1

- We have shown you a solution, but we want you to learn how to use tools to generate them
- http://www.plantuml.com/plantuml
  - Free web generator
  - Integration with Google Docs
- http://www.umlet.com/
  - Install program (works offline)

PlantUML

```
@startuml
skinparam style strictuml
class Main {
Main()
setAllBValuesTo3()
}
class A{
name
A( name )
setBValue( value)
}
class B{
count
B()
setValue( value )
}
Main -> "*" A
A->  B
@enduml
```



Moving on….

- More Object-Oriented Principles for Design
- Learn about next set of principles
  - What are they?
  - Why are they useful?
  - When are they most important?
  - How can we apply them?

Major Goals of ALL Program Design

- Say someone has written a program that works and it has no bugs, but it is *poorly designed*.
  - What does that mean?
  - Why do we care?
- There are two major goals:

1. It would be good if it was easy to understand

2. It would be good if it was easy to make changes to it

Q1

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**

What are the principles?

3. Functionality should be **distributed efficiently**
    a) No single part of the system should get too large
    b) Each class should have a single responsibility it accomplishes

Why do we want to spread things out?

Why is it good to have a single responsibility?

Why do we even have classes?

Think/Pair/Share on no String class

- Instead, what if we just passed around arrays of characters - char[]
- And every String function that exists now, would instead be a function that operated on arrays of characters
- E.g. char[] stringSubstring(char[] input, int start, int end)
- Would things be any different? Think about this for a minute then discuss it with the person next to you.

Concatenate…

```
String stringName1 = "jason";
String stringName2 = "yoder";
String stringConcat = stringName1.concat( stringName2 );
System.out.println(  stringConcat );
-----------------------------------------------------------
char[] charName1 = {'j','a','s','o','n'};
char[] charName2 = {'y','o','d','e','r'};
char[] charConcat =
      new char[charName1.length + charName2.length];

for (int i=0; i< charName1.length; i++) {
      charConcat[i] = charName1[i];
}
for (int i=0; i< charName2.length; i++) {
      charConcat[charName1.length + i] = charName2[i];
}
System.out.println( Arrays.toString(charConcat)  );
```

Class sizes

- Why not put all the Math utilities in the String class?
  - We could just get anything we need done with one library!

Q2

Pizza Restaurant Scenario

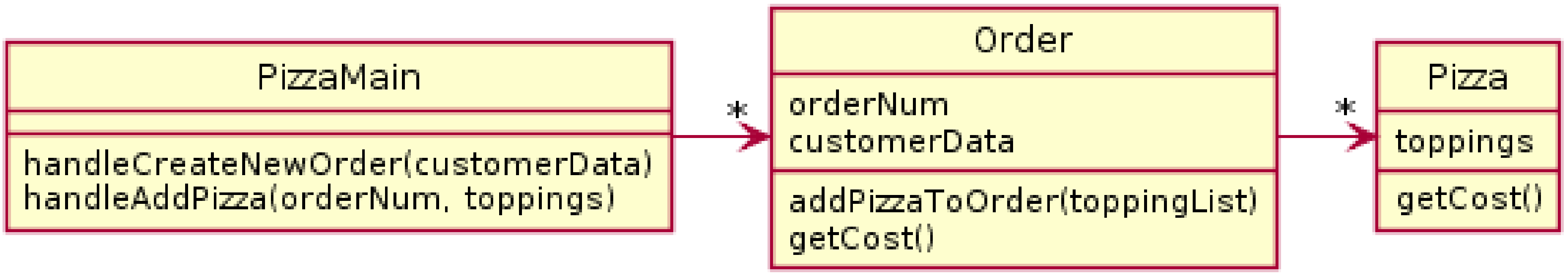
A pizza restaurant needs to calculate the costs of orders and record what pizzas need to be made. An order consists of a number of pizzas which might have toppings as well as a customer’s name and an order date. Each pizza costs $8 with no toppings. The first 2 toppings cost $2 apiece. Additional toppings beyond that cost $1. If a pizza has just peppers, onions, and sausage - that's "The special" and it costs $12.

Design a UML diagram to model this.

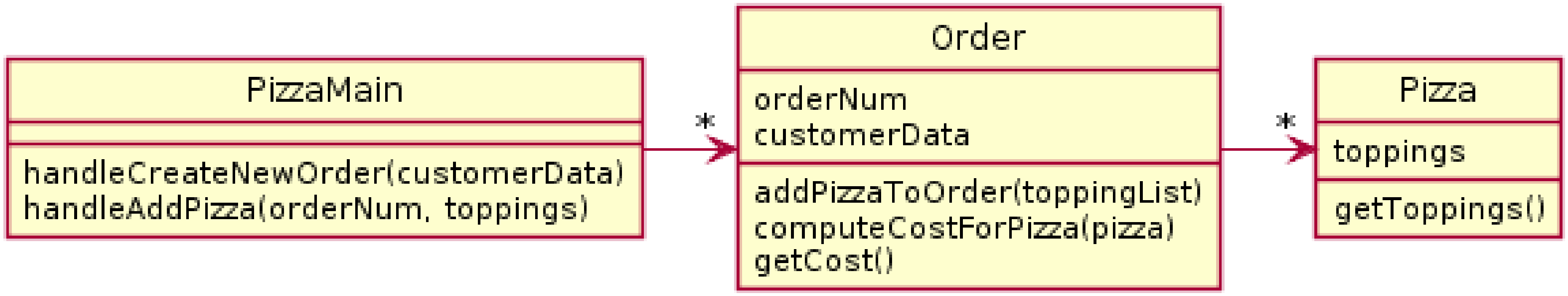
UML

1. What classes did you have?
2. Where did you put “costOfPizza()”?

Solution A

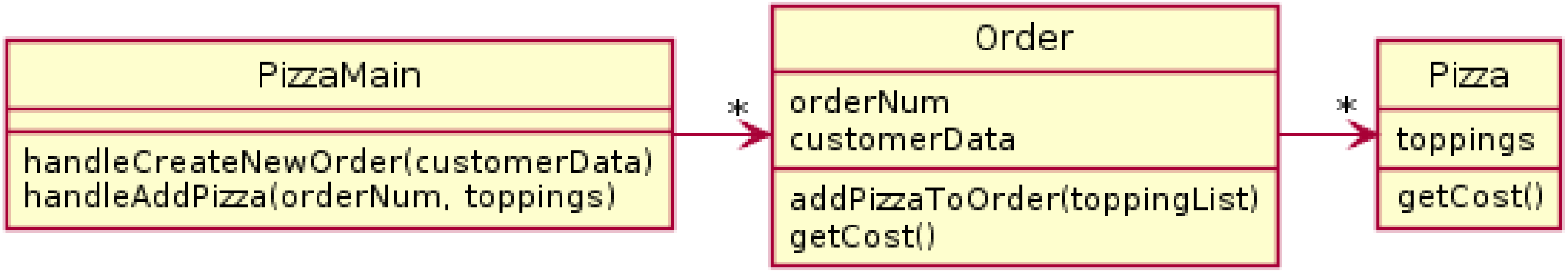

PizzaMain
handleCreateNewOrder(customerData)
handleAddPizza(orderNum, toppings)
*
Order
orderNum
customerData
addPizzaToOrder(toppingList)
getCost()
*
Pizza
toppings
getCost()

Solution B

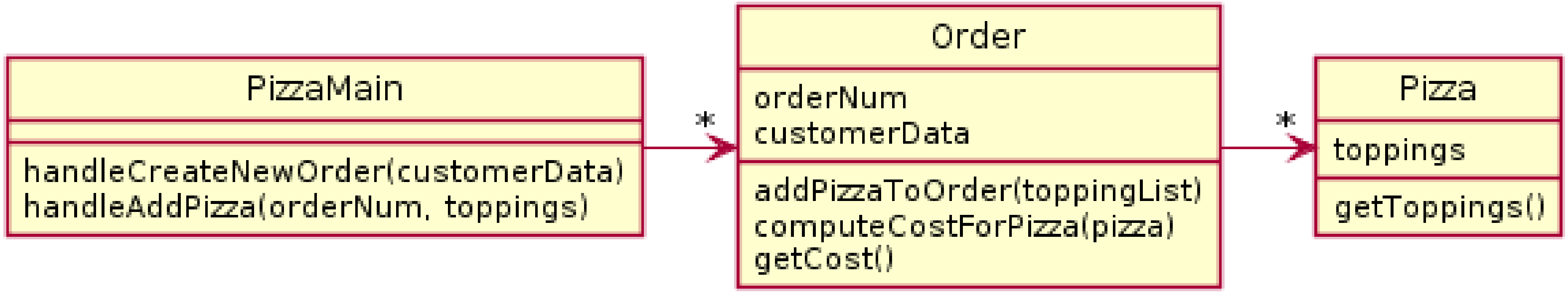

PizzaMain
handleCreateNewOrder(customerData)
handleAddPizza(orderNum, toppings)
*
Order
orderNum
customerData
addPizzaToOrder(toppingList)
computeCostForPizza(pizza)
getCost()
*
Pizza
toppings
getToppings()

Which is better?

Solution A


PizzaMain
handleCreateNewOrder(customerData)
handleAddPizza(orderNum, toppings)
*
Order
orderNum
customerData
addPizzaToOrder(toppingList)
getCost()
*
Pizza
toppings
getCost()

Solution B


PizzaMain
handleCreateNewOrder(customerData)
handleAddPizza(orderNum, toppings)
*
Order
orderNum
customerData
addPizzaToOrder(toppingList)
computeCostForPizza(pizza)
getCost()
*
Pizza
toppings
getToppings()

Conceptually, calculating costs could belong in either order or pizza. But order is doing a lot of stuff – Pizza is just a dumb data holder. So by spreading the functionality into the pizza, we improve the design.

Alternate Pizza Restaurant

Consider now the ability to add a discount to an order, such that a coupon can be added to an order and then it changes how the cost is calculated. A coupon may offer a discount percentage for toppings (50% off all toppings) and/or a percentage off of entire orders. In addition, there should be a way to calculate how long it takes to create a pizza based on its size and toppings.

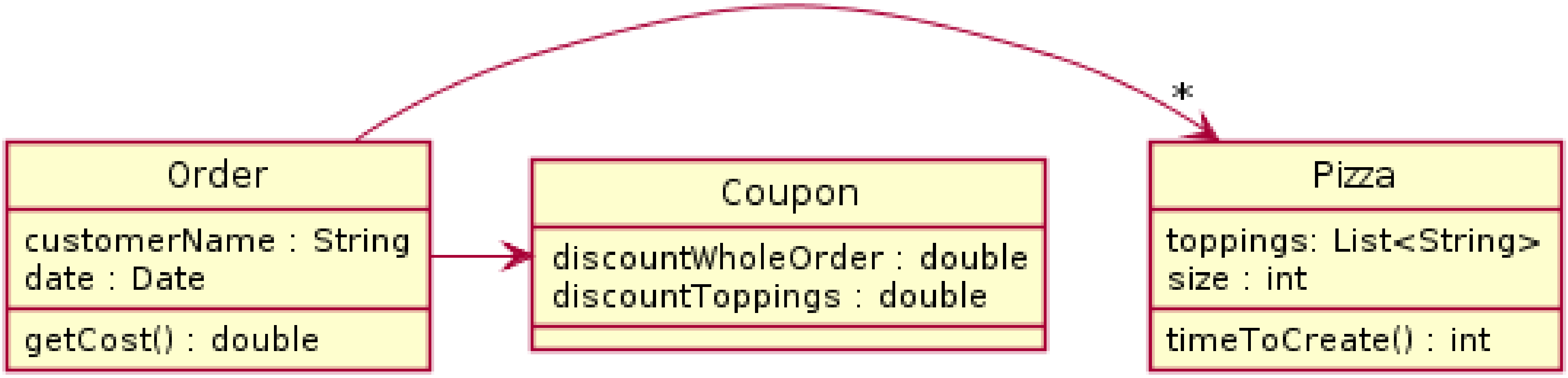
Design a UML diagram to model this.

UML

1. What classes did you have?
2. Where did you put “getCost()”?

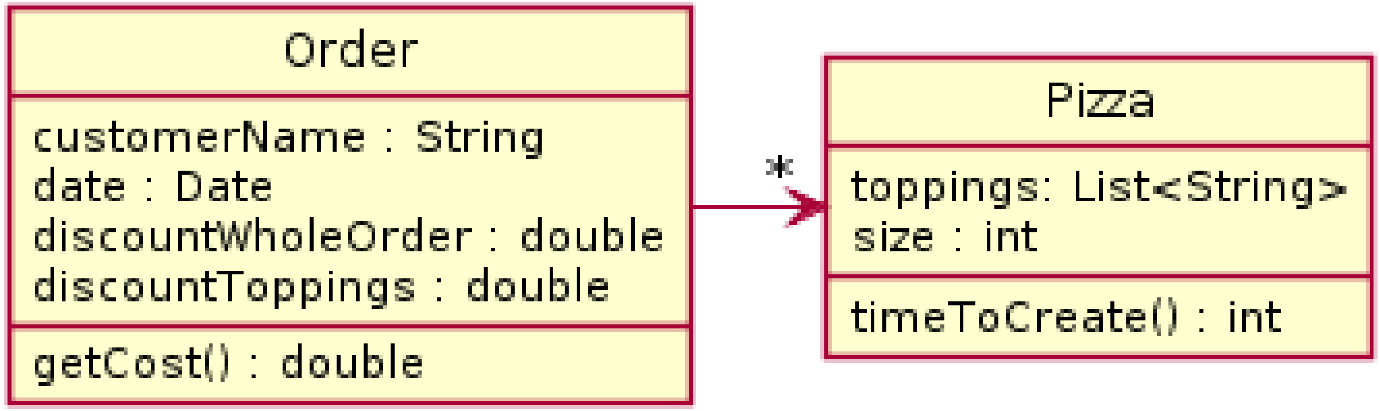
One Solution

3. Functionality should be **distributed efficiently**
   a) No single part of the system should get too large
   b) Each class should have a single responsibility it accomplishes



Do we need Coupon or Topping?

- It depends, do the classes do anything ***with*** their data, or are they just ***data classes*** that simply all you to get and set values?



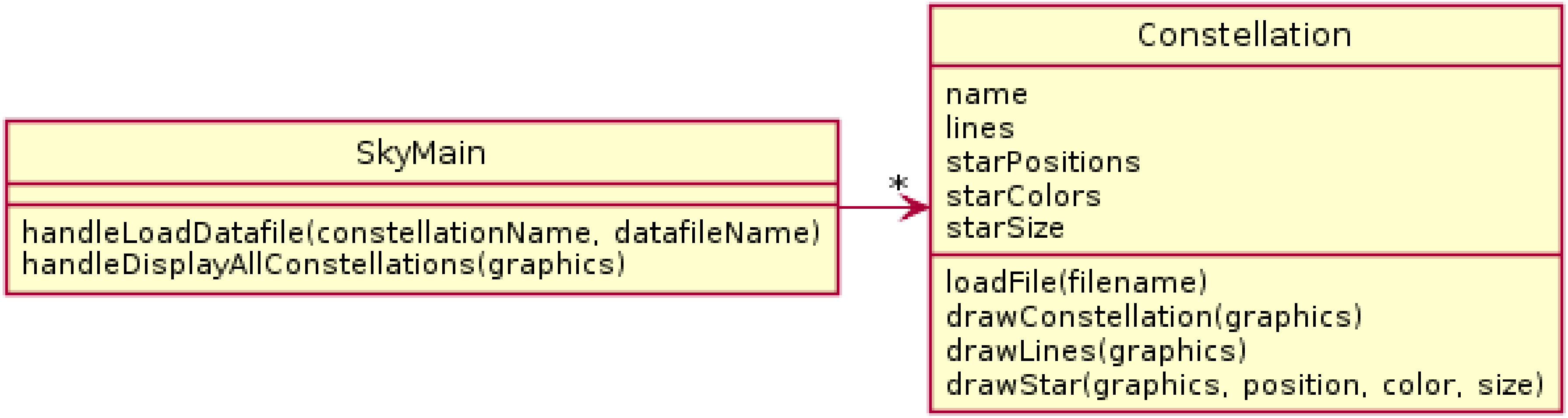
Rule of Thumb - Avoid Data Classes!

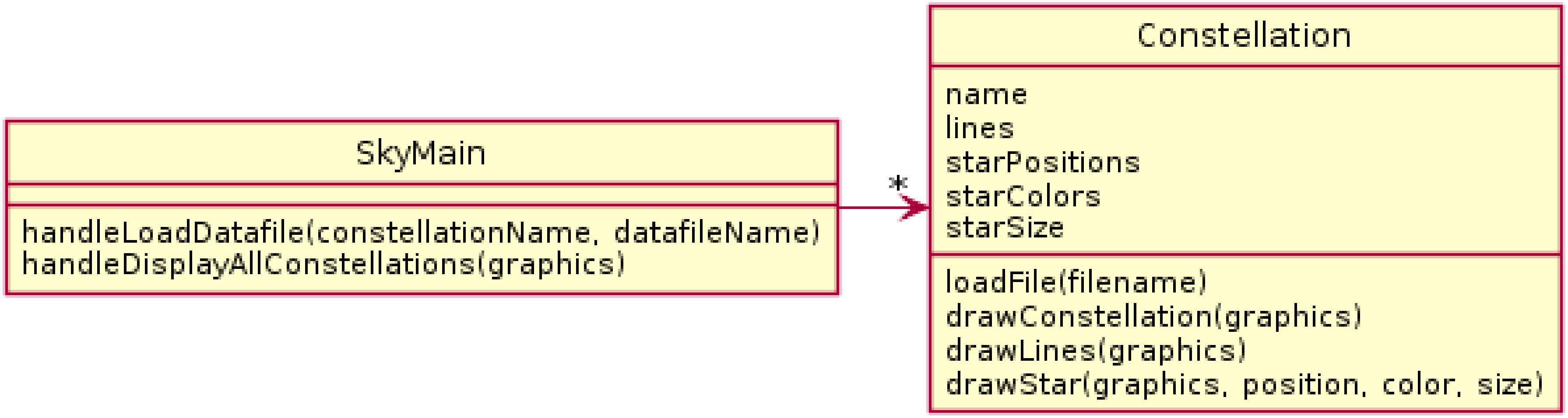
- A data class is a class that just contains getters and setters
- Often, we think of Data Classes as violating a principle of OOD called ***encapsulation*** because they aren’t in control of their own data – they are just dumb repositories for other classes to use
- Usually you can improve a data class by finding functionality to add to them

A particular program is designed to load constellations from datafiles and draw them on the screen. The datafiles include details about star location, size, and color, as well as which stars ought to be connected to draw the constellation.

Depending on the star data, each star should be drawn differently (e.g. right size, right color).

Explain the problem with the given solution and then propose a UML solution of your own.

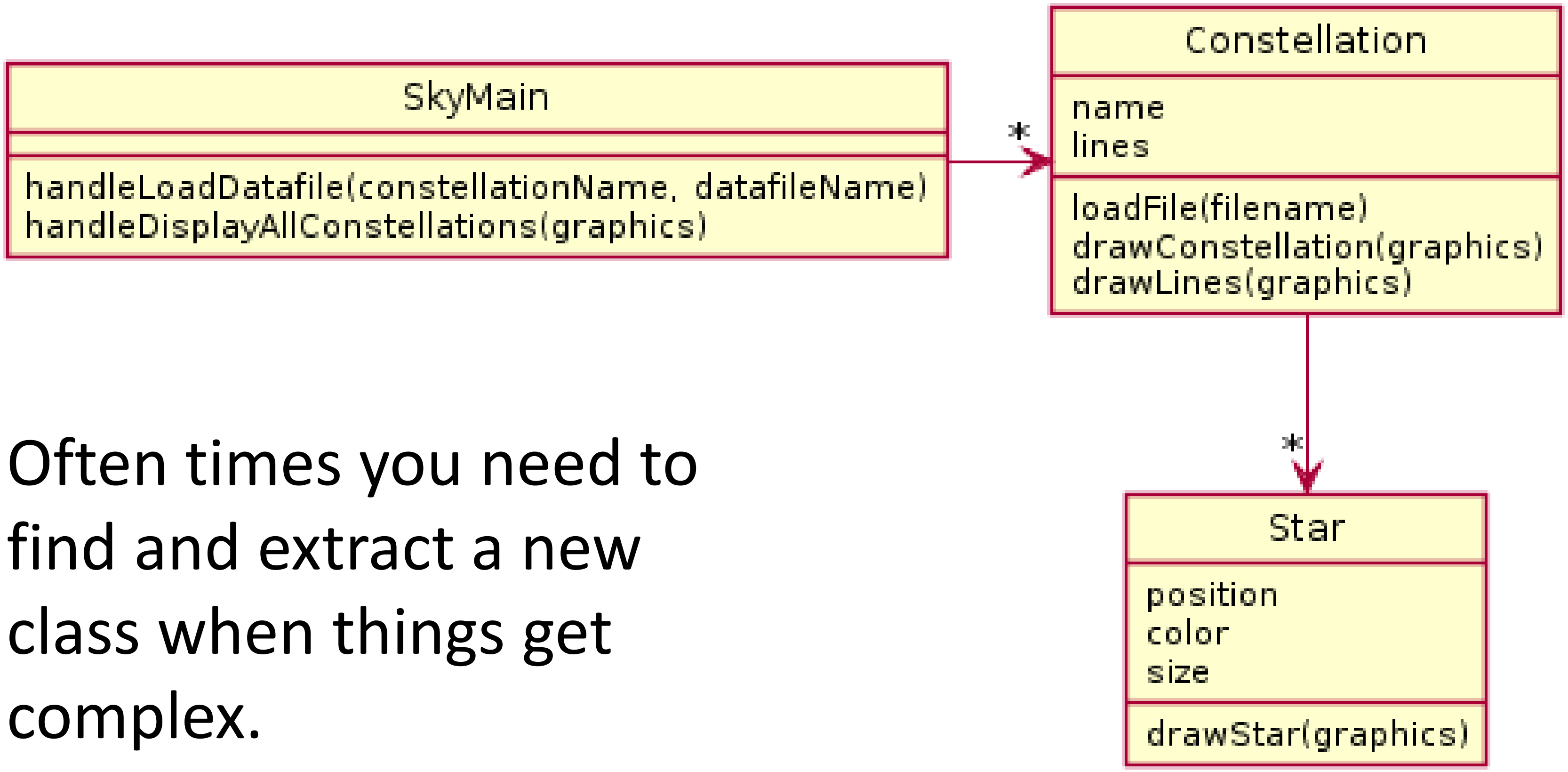

SkyMain
handleLoadDatafile(constellationName, datafileName)
handleDisplayAllConstellations(graphics)
*
Constellation
name
lines
starPositions
starColors
starSize
loadFile(filename)
drawConstellation(graphics)
drawLines(graphics)
drawStar(graphics, position, color, size)



SkyMain
handleLoadDatafile(constellationName, datafileName)
handleDisplayAllConstellations(graphics)
*
Constellation
name
lines
starPositions
starColors
starSize
loadFile(filename)
drawConstellation(graphics)
drawLines(graphics)
drawStar(graphics, position, color, size)

3a. Constellation does everything (except maybe the parsing done by main).

My solution


SkyMain
handleLoadDatafile(constellationName, datafileName)
handleDisplayAllConstellations(graphics)
*
Constellation
name
lines
loadFile(filename)
drawConstellation(graphics)
drawLines(graphics)
*
Star
position
color
size
drawStar(graphics)

Often times you need to find and extract a new class when things get complex.

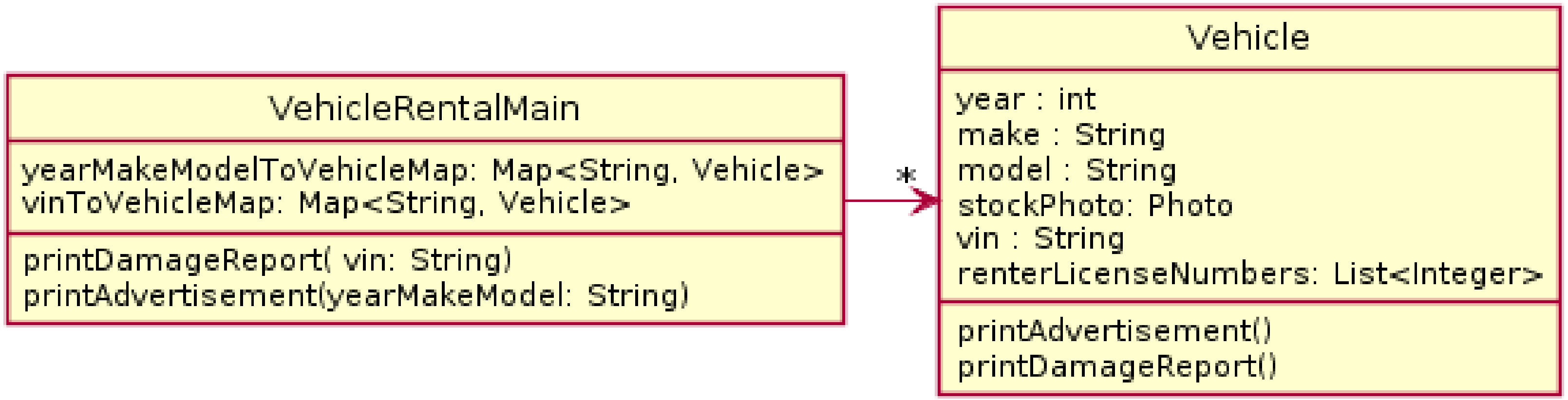
Your turn!

- Try to design UML for the following scenario

Rental Company

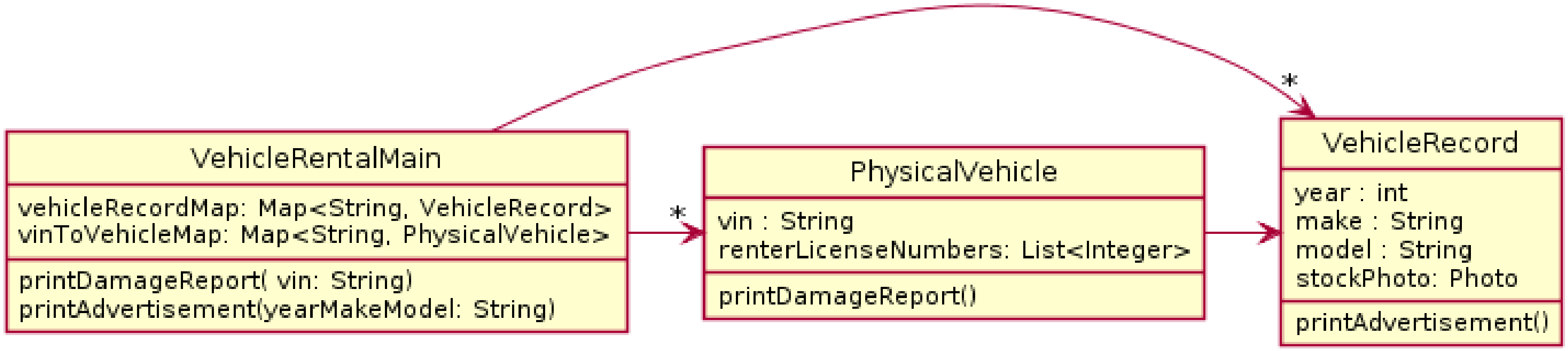
- A rental company has many vehicles that it rents. Vehicles have a year, make, and model, and a stock photo advertising the vehicle on file. There are multiple vehicles that are the same year, make, model.
- However, additional information on the specific physical vehicles is also required. For instance, each physical vehicle has a vin (vehicle identification number unique to each), a description of any damage to it, and the driver's license numbers of everyone who has rented it.
- The company also needs to be able to print out the damage report of a vehicle given a VIN. The company also has to be able to print out an advertisement using the stock photo for a given year, make, and model.

Operable but poor solution


VehicleRentalMain
yearMakeModelToVehicleMap: Map<String, Vehicle>
vinToVehicleMap: Map<String, Vehicle>
printDamageReport(vin: String)
printAdvertisement(yearMakeModel: String)
*
Vehicle
year : int
make : String
model : String
stockPhoto: Photo
vin : String
renterLicenseNumbers: List<Integer>
printAdvertisement()
printDamageReport()

- What is wrong?

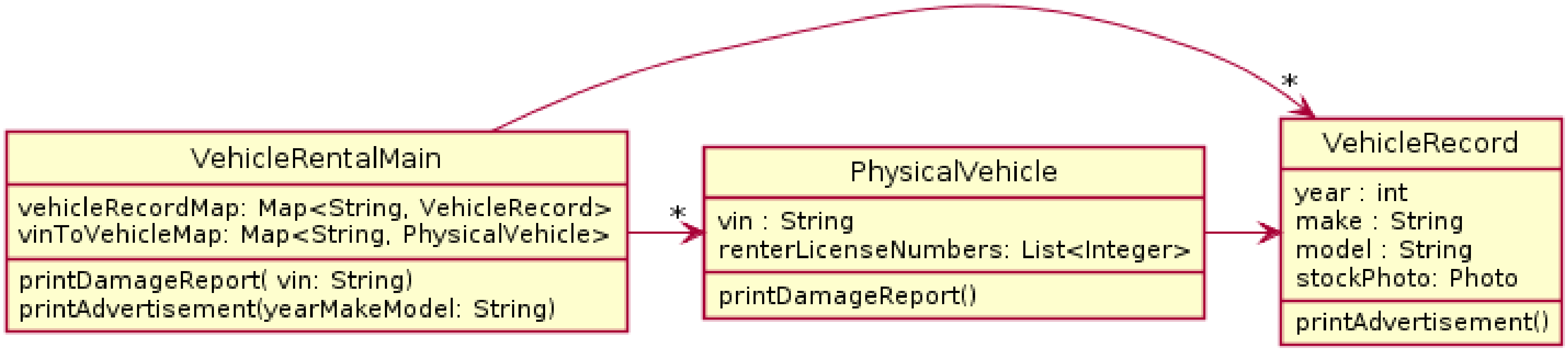
Better Solution


VehicleRentalMain
vehicleRecordMap: Map<String, VehicleRecord>
vinToVehicleMap: Map<String, PhysicalVehicle>
printDamageReport(vin: String)
printAdvertisement(yearMakeModel: String)
*
PhysicalVehicle
vin : String
renterLicenseNumbers: List<Integer>
printDamageReport()
*
VehicleRecord
year : int
make : String
model : String
stockPhoto: Photo
printAdvertisement()

- There are two different things:
  - Actual physical vehicle
  - Records of specific vehicles
- Class has own behaviors (reports)
  - Used for specific purposes, specific data

Design Principles

3. Functionality should be **spread** throughout the system
   a) No single part of the system should get too large
   b) Each class should have a single responsibility it accomplishes


VehicleRentalMain
vehicleRecordMap: Map<String, VehicleRecord>
vinToVehicleMap: Map<String, PhysicalVehicle>
printDamageReport(vin: String)
printAdvertisement(yearMakeModel: String)
*
PhysicalVehicle
vin : String
renterLicenseNumbers: List<Integer>
printDamageReport()
*
VehicleRecord
year : int
make : String
model : String
stockPhoto: Photo
printAdvertisement()

Encapsulation

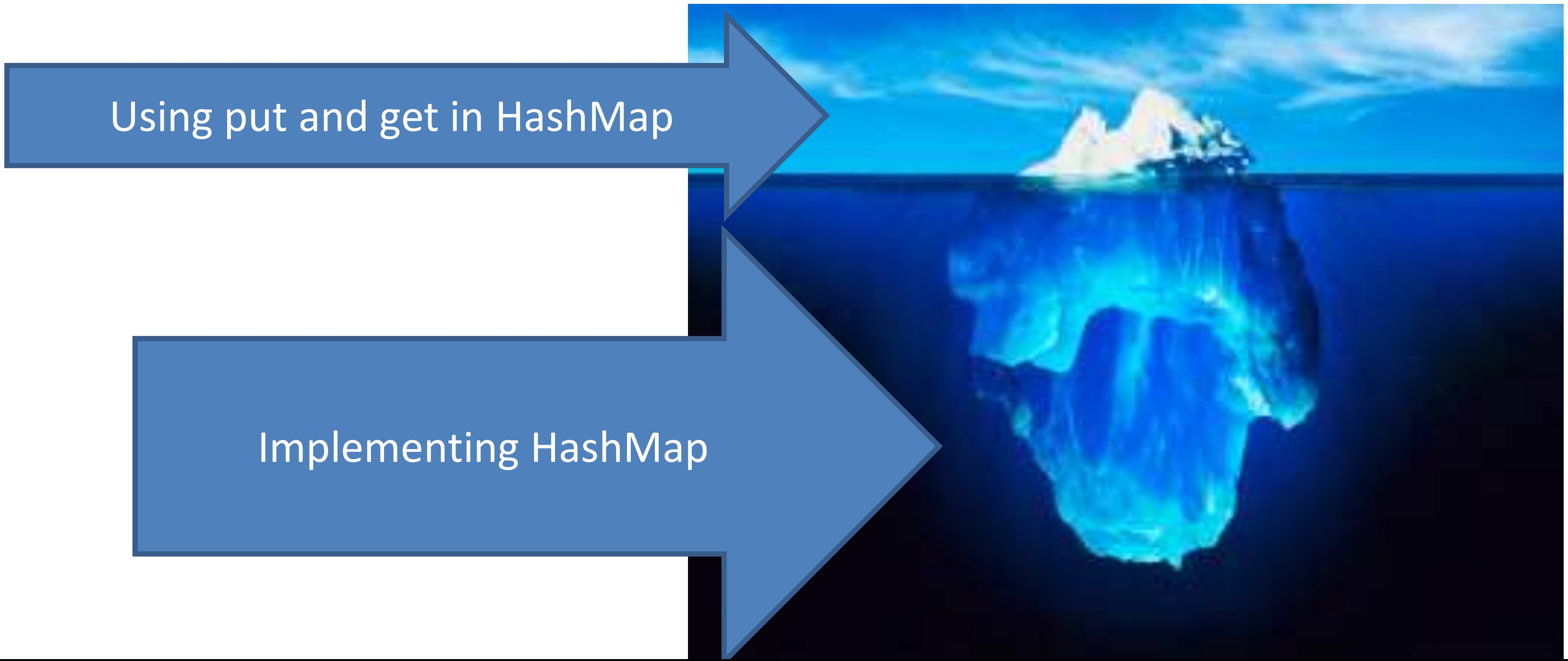
- Makes your program easier to understand by
  - Grouping related stuff together

- 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

Q3

Encapsulation

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



Encapsulation

Makes your program easier to change by…

- Allowing you to change how your data is represented

Q4

Reminders

- Design Problems2 are due at start of next class
- ImplementingDesign1
  - Make sure to submit by THIS Friday night:
    - **code**
    - **UML**
    - **Reflection**