SOFTWARE ENGINEERING

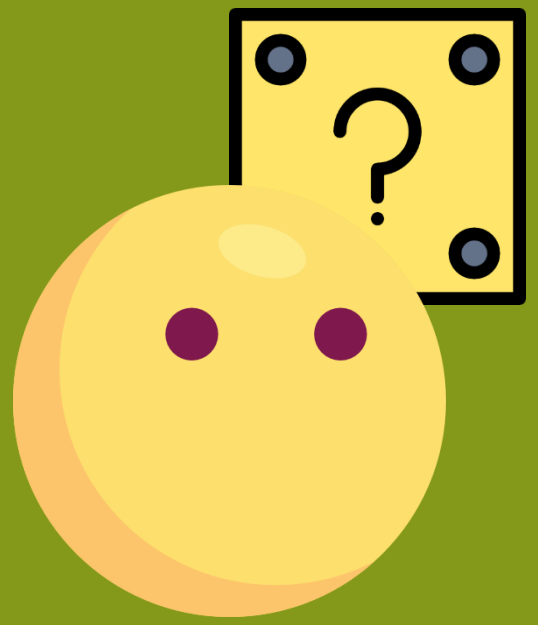
Chapter 3.1: System Design

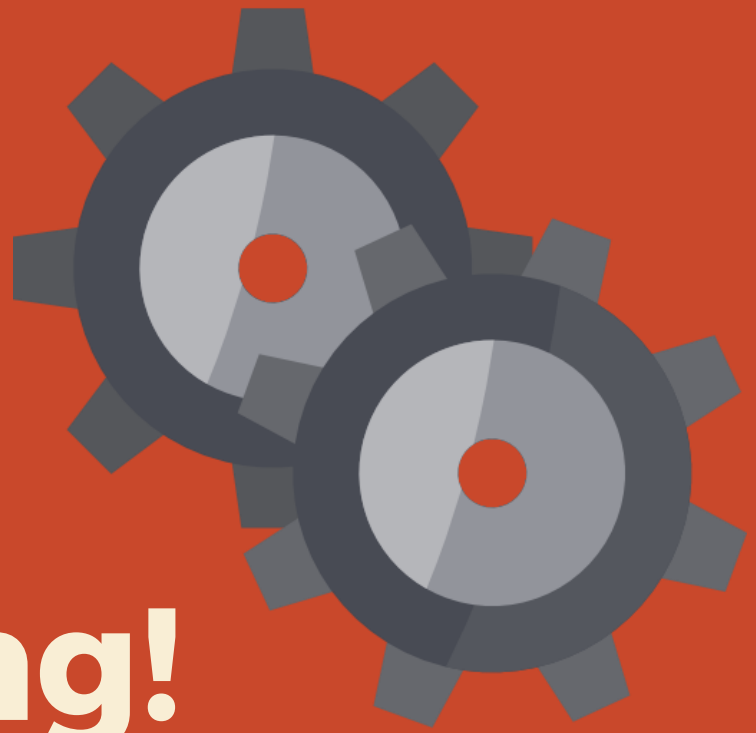
MOTIVATION...

Simplicity is the soul of efficiency.

—Austin Freeman (in *The Eye of Osiris*)

How We Will Achieve That?



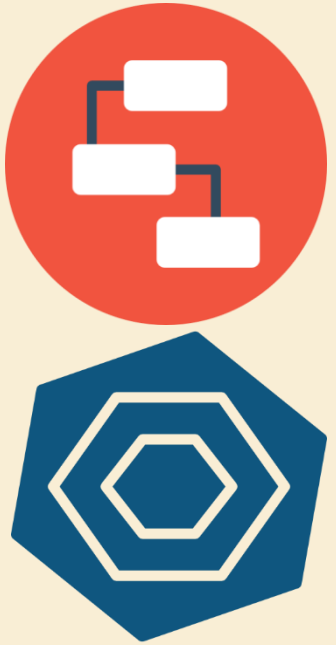


System Modeling!

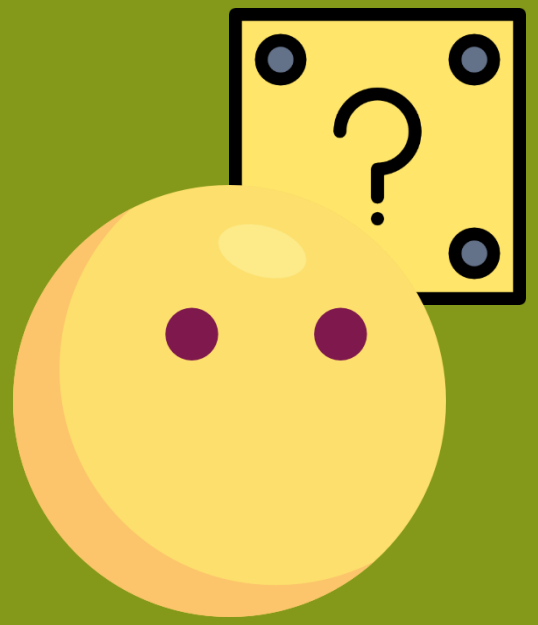
System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system [SOMMERVILLE]

Facts About System Modeling!

- Models are representations of the system on each perspective.
- You require some kind of graphical notation like UML



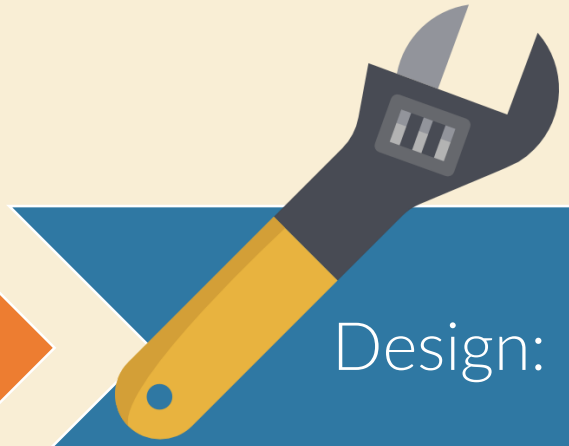
When should I Model Systems?



Software Processes Involved!

Requirements:

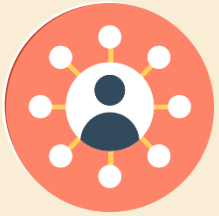
- Validate
- Strengths and weaknesses



Design:

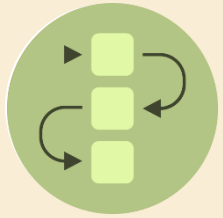
- Document implementation
- Generate implementation plans

Types of models



External
Context

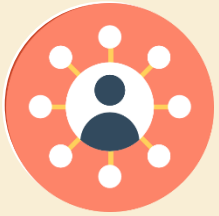
Interactive
Between Context or components



Structural
Organization and Data

Behavioral
Response to Events

Types of models and UML



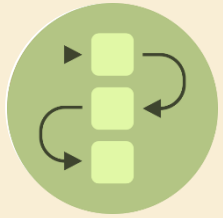
External

Activity Diagrams

Interactive

Use Case Diagrams

Sequence Diagrams



Structural

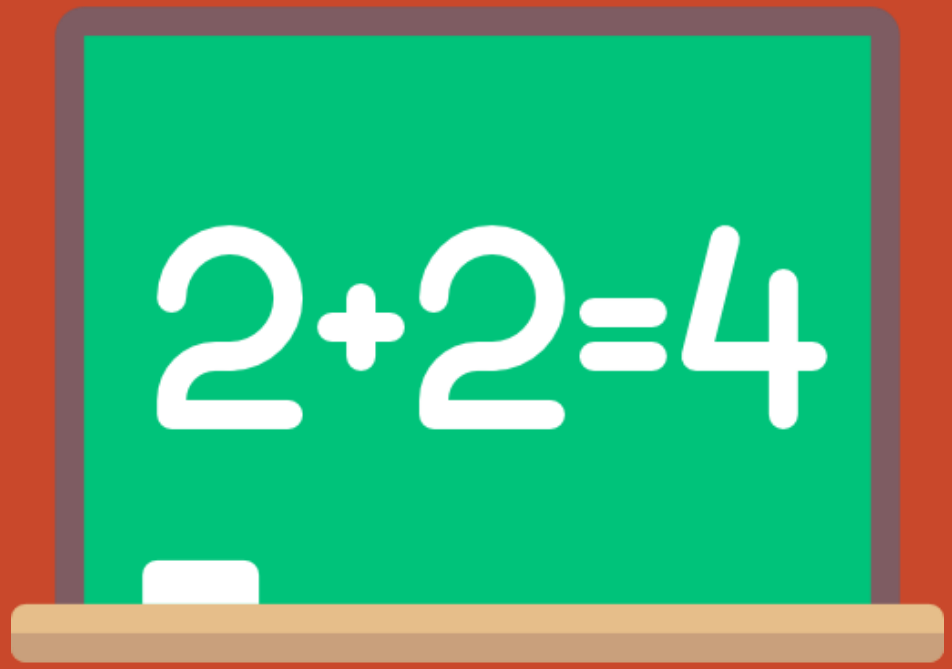
Class Diagrams

Behavioral

State Diagrams

Don’t Forget!

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [LARMAN]



Class Diagrams

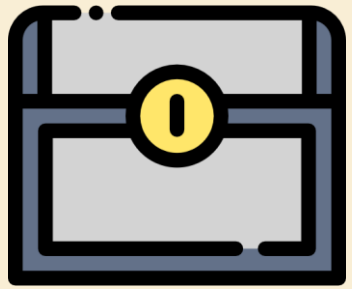
They stablish the Domain Model of the system, are a key element to organize the data structure in OOM (object oriented models).

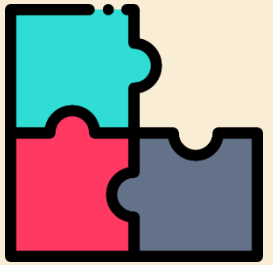
Object Oriented?

Object Oriented Models

- Object oriented design or analysis describes objects or concepts in the problem's domain

- There's an emphasis in giving such descriptions in terms of objects.





Remember about classes, interphaces, objects, methods…

Classes

- Each object belongs to a class, so let's say that a class is an abstract representation (template) for creating objects

- Each object has unique features (call it attributes)

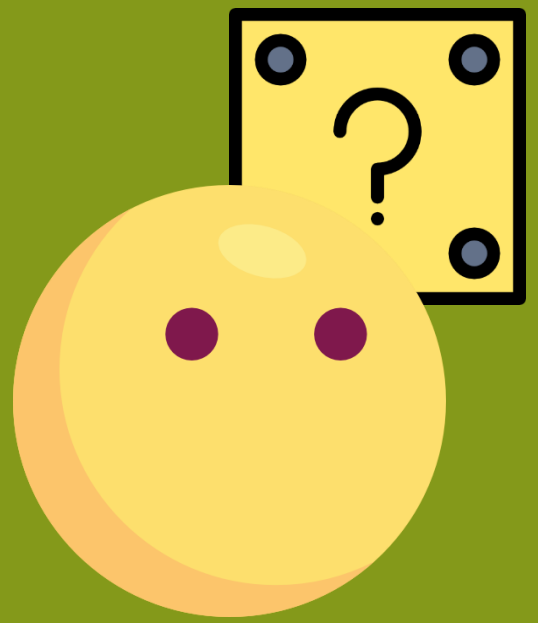
Classes

1) Encapsulation (hide functionality)

2) Inheritance (superclasses)

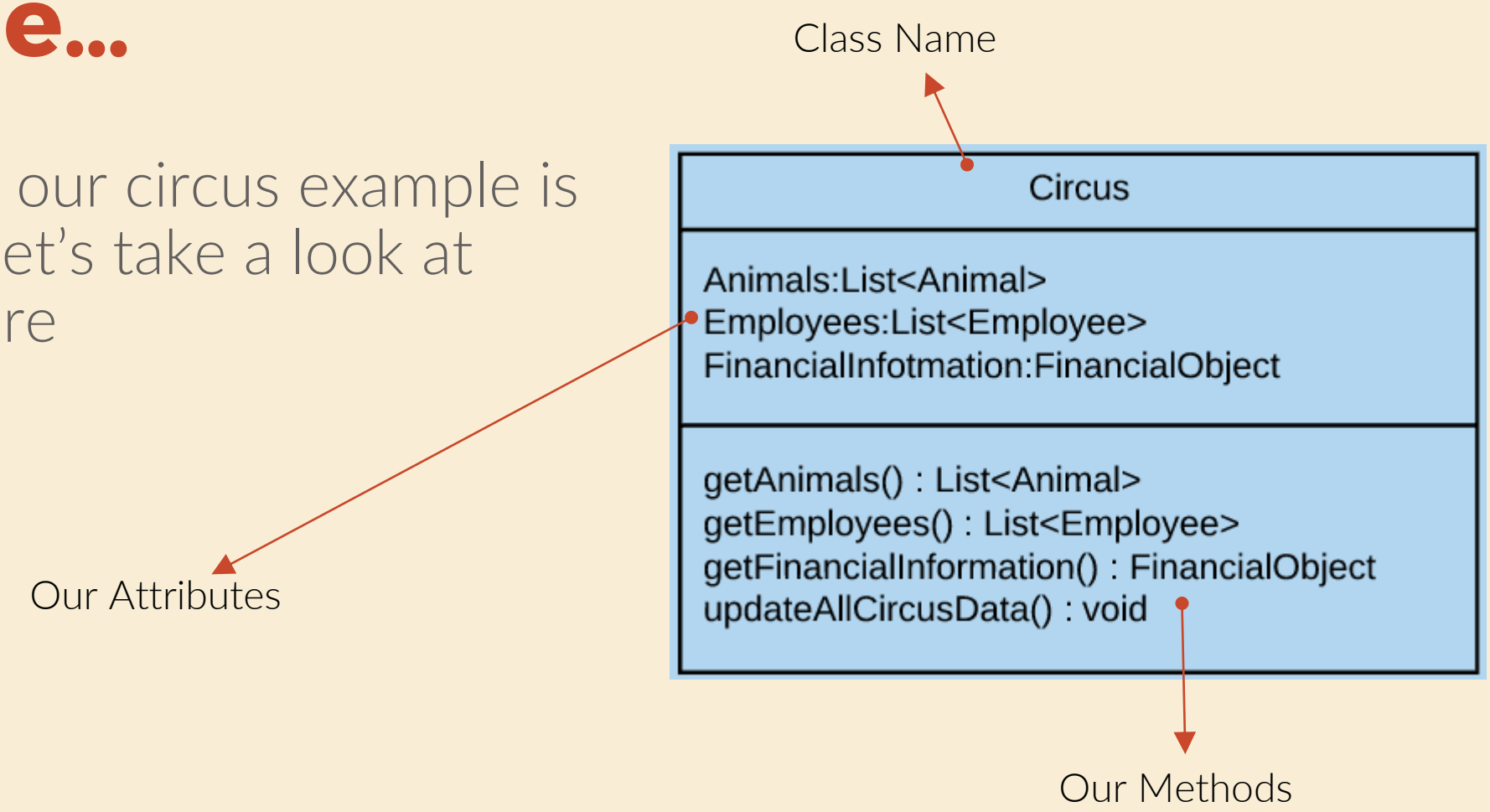
3) Polymorphism (different behavior)

Can we start?



Example...

- Let's suppose our circus example is one class, so let's take a look at everything here

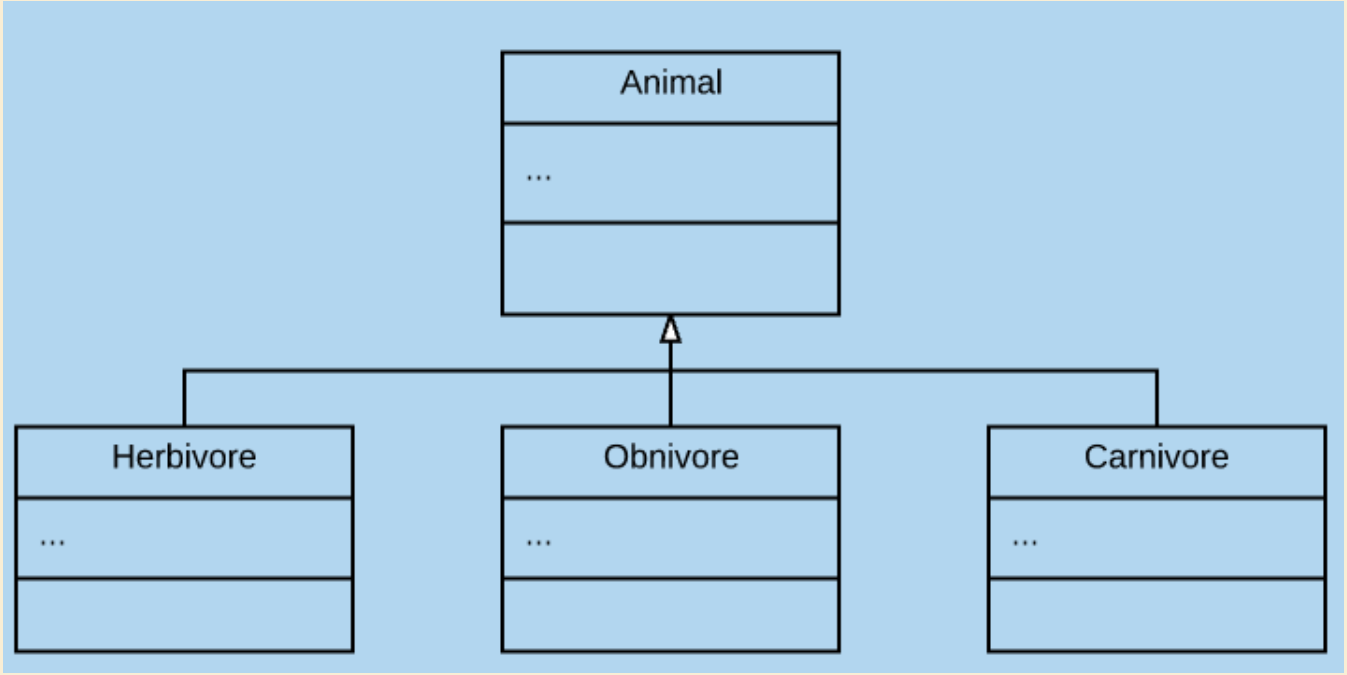


Types of data

Tipe	Sign
Public	+
Protected	#
Private	-
Package	~

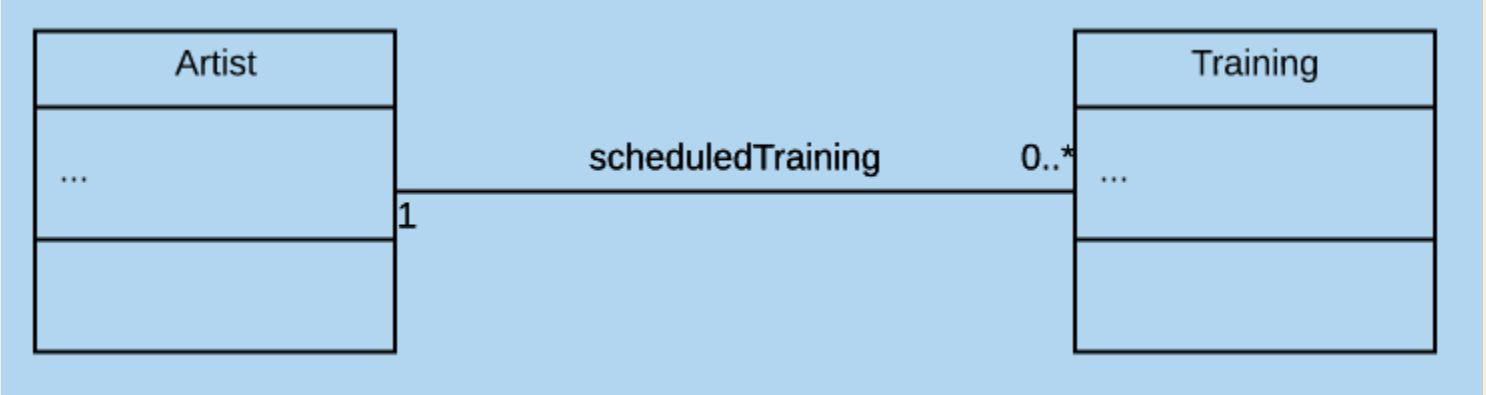
Inheritance

- You use the non-filled arrow pointing to the superclass to call an inheritance.



Standard Assosiation

- Link clases to indicate assosiations, they mean that those clases interact with each other the phrase at the middle is the relationship and the numbers are the multiplicity of the relationship



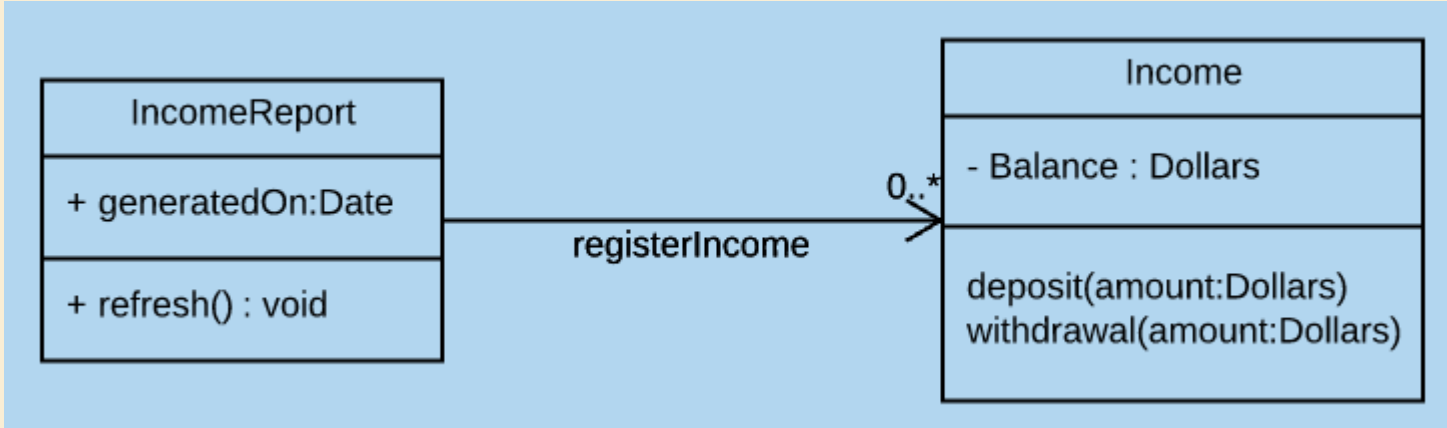
(1 Artist can Schedule from 0 to any training sessions)

Multiplicity table

Indicator	Meaning
0..1	Zero or one
1	Only one
0..*	Zero or more
*	Zero or more
1..*	One or more
6	Six only
11...23	Eleven to twenty three
0...8	Zero to eight

Uni Directional Asossiation

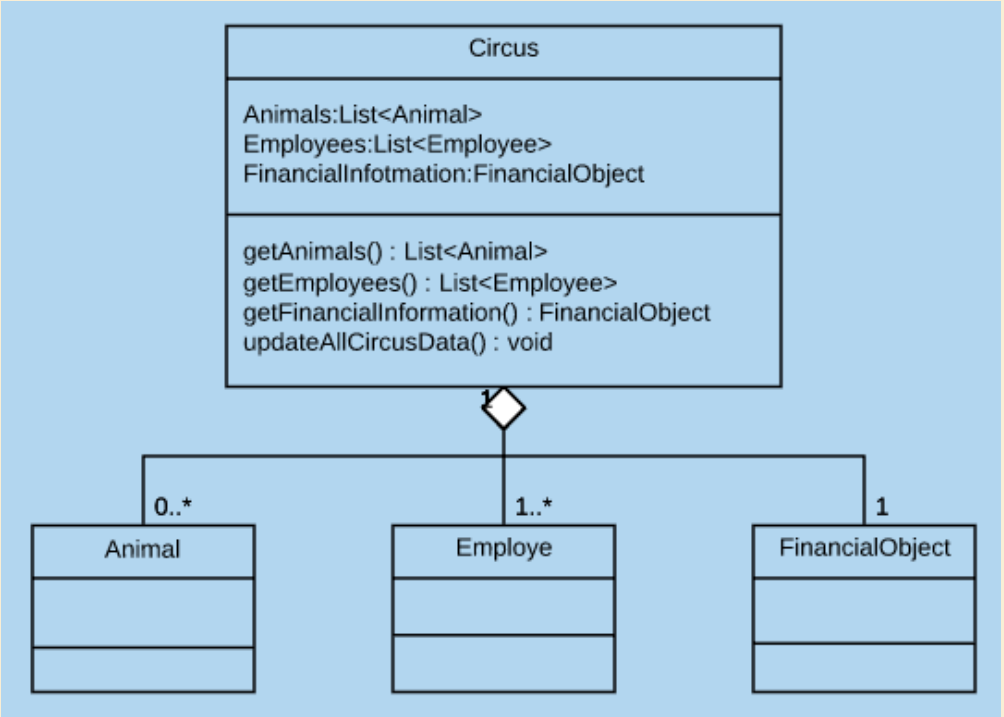
- When the relationship between both classes exist but just one class knows about the relationship the notation changes…



IncomeReport knows about the class Income, but it doesn't happen backwise

Aggregation

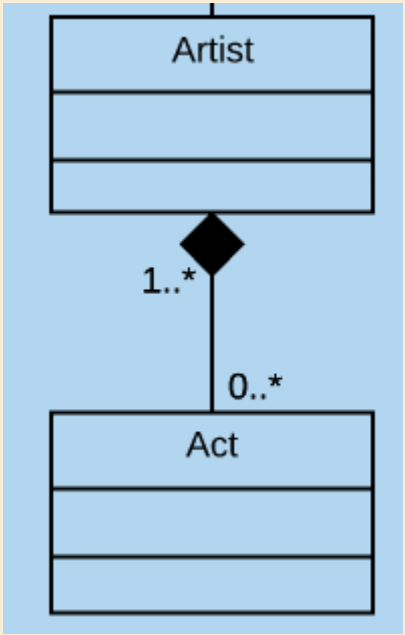
- Is a relationship of hierarchy in which one part is composed by others



The circus is composed by Animals, Employees and a financial object

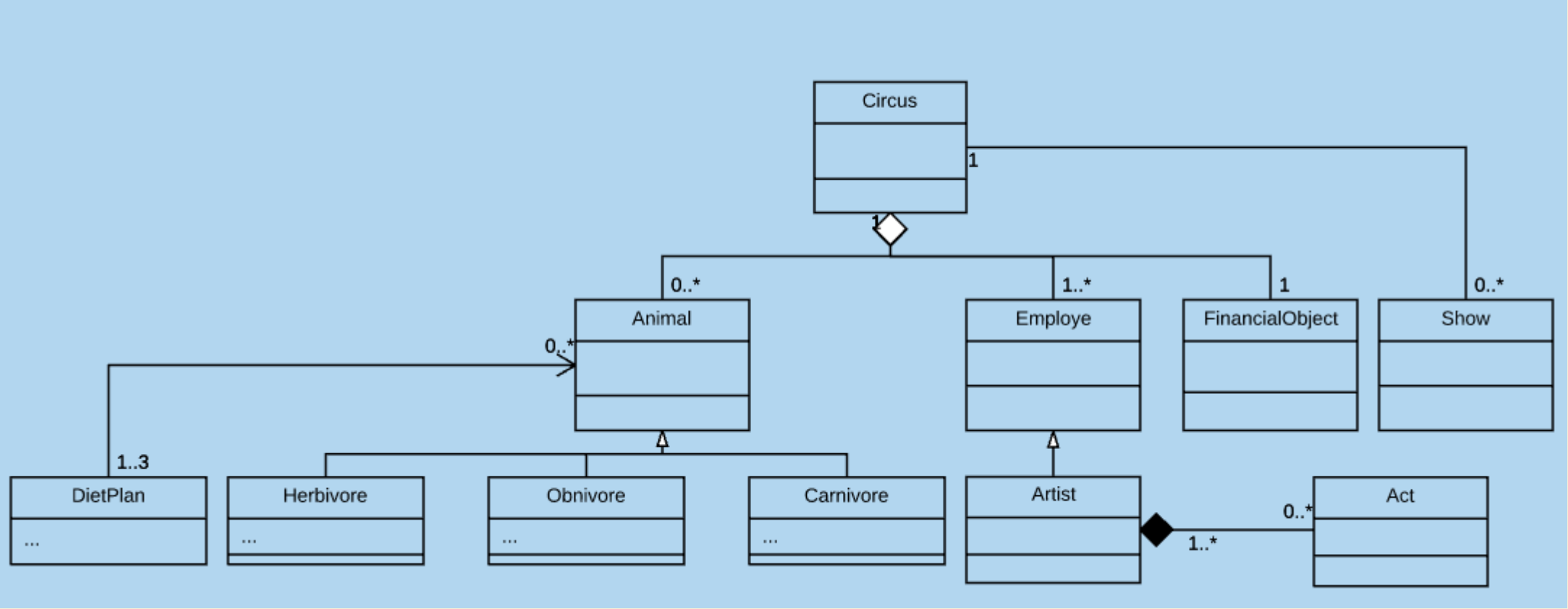
Composition

- They can only exist if the parent class exist, is a strong version of aggregation!



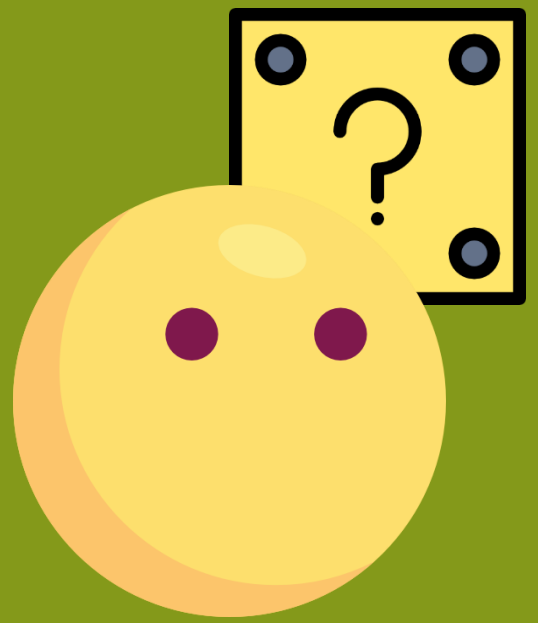
Give me an example!

Look at this sketch!



And... this is not completed! 

What about the attributes inside a class?



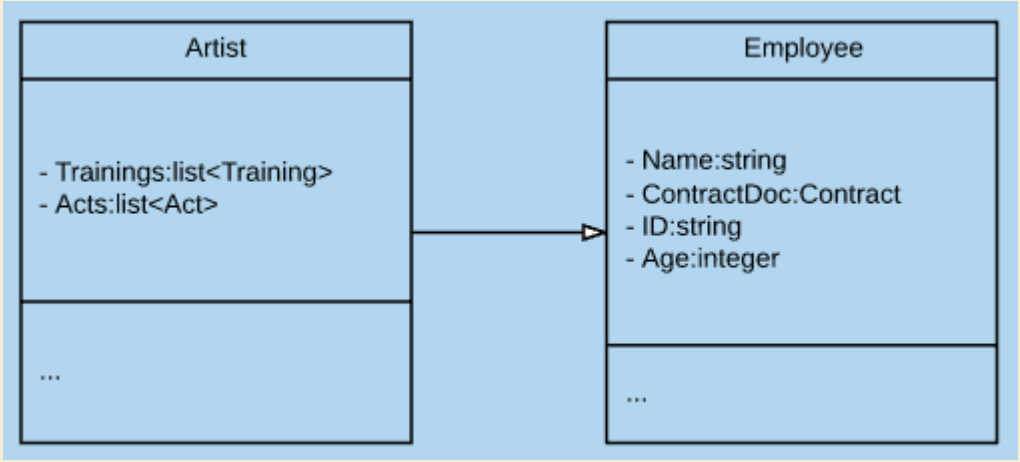
Composite structure diagrams

- You can specify the relationships between class attributes by using Composite structure diagrams

- In older versions of UML or older literature you can find this with ‘Context Diagrams’

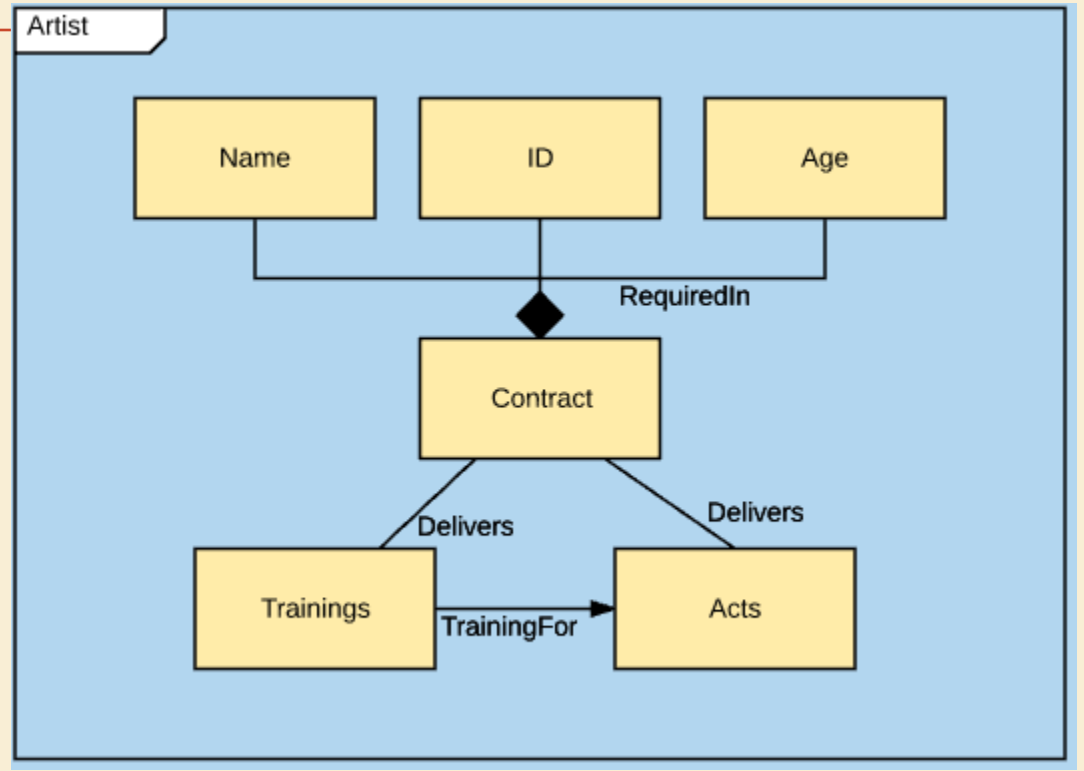
Composite Structure Example

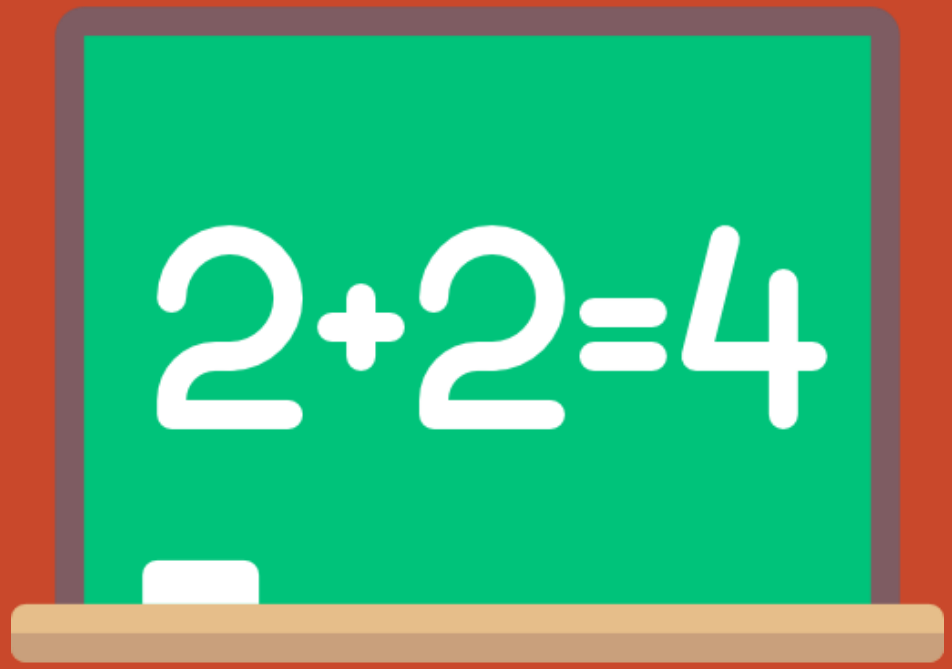
- Let's take this classes, artist and employee!
- Remember, there's an inheritance so the attributes of employee will be in any object of type Artist!



Composite Structure Diagram

- Each attribute is a square.
- Relationships are treated in the same way as in a class diagram.



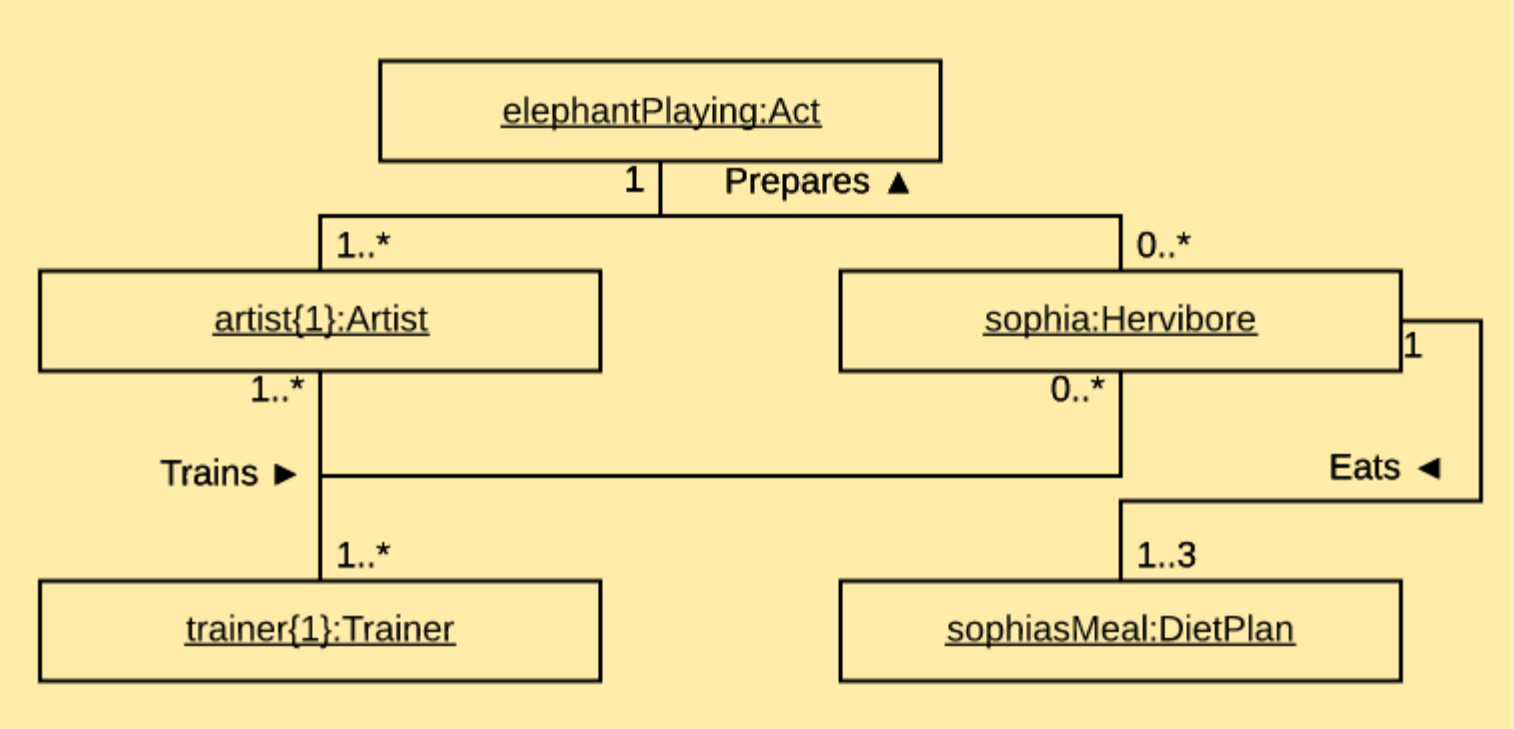


Object Diagrams

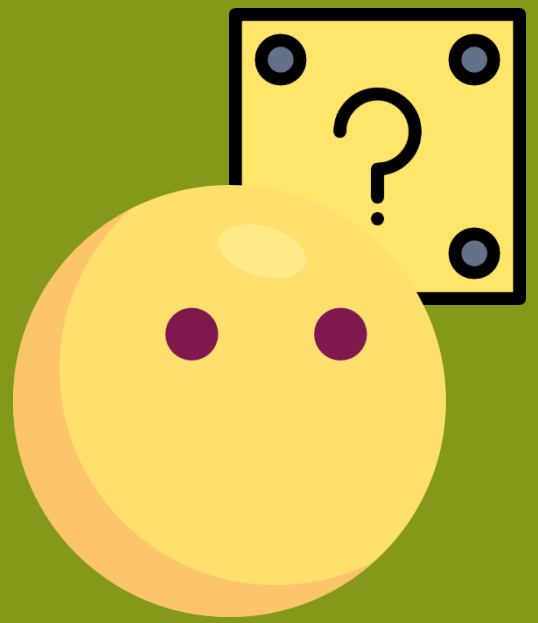
Shows instances of the classes

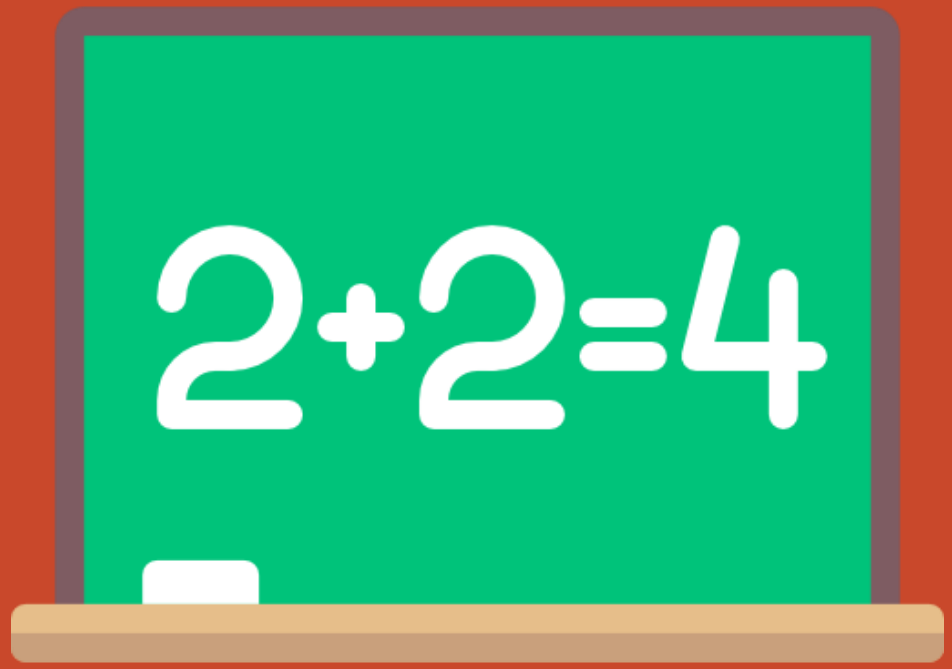
Object Diagram - Example

- Each square is an object (Instance of a class)
- The triangle indicates the direction of the relationship



Something else?



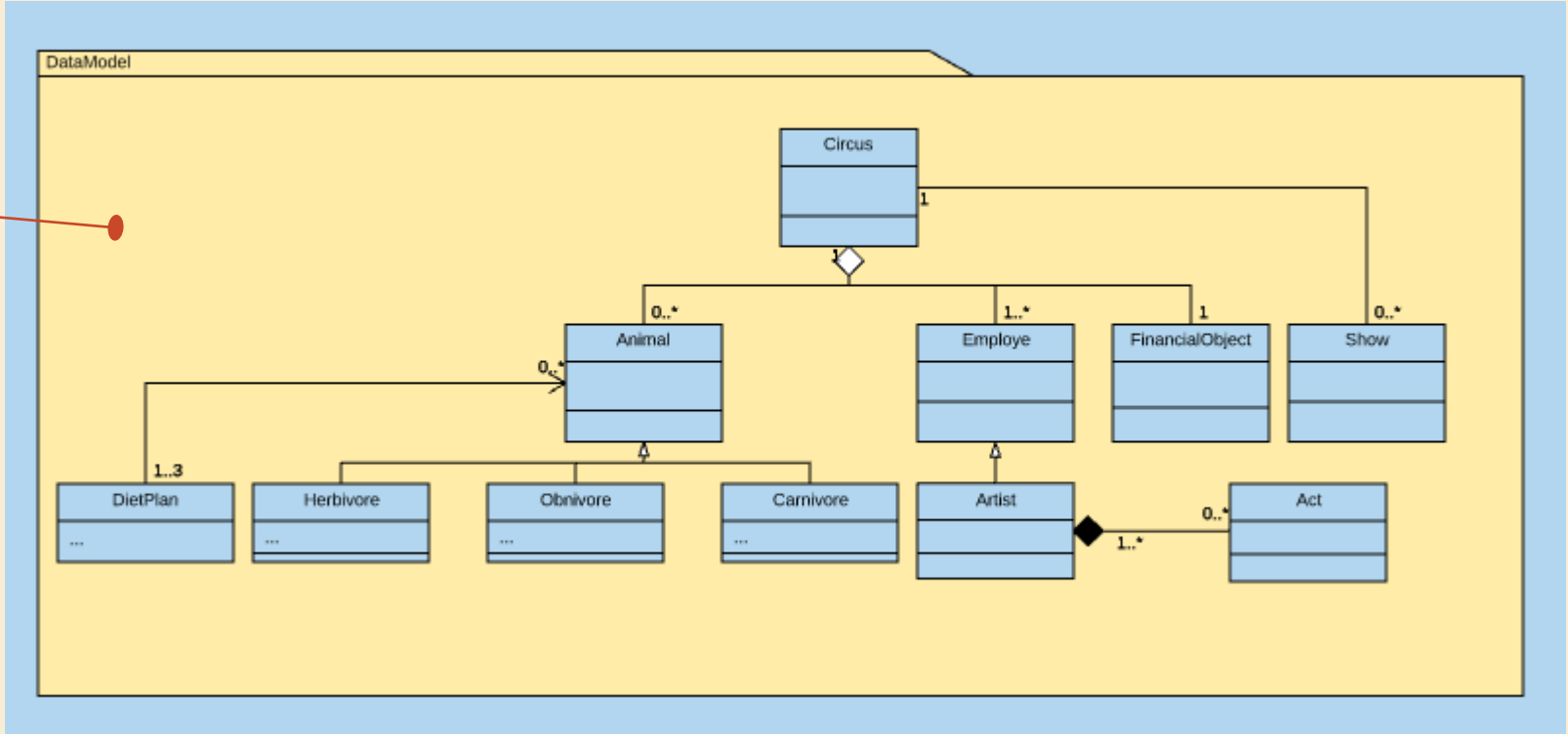


Package Diagrams

A set of classes can belong to a package

What is a package?

This is a Package!

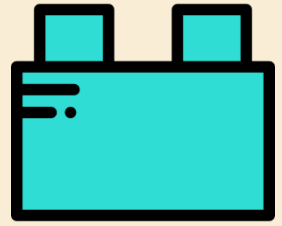


The package DataModel (folder in UML) contains the domain model for this example!

Some insights

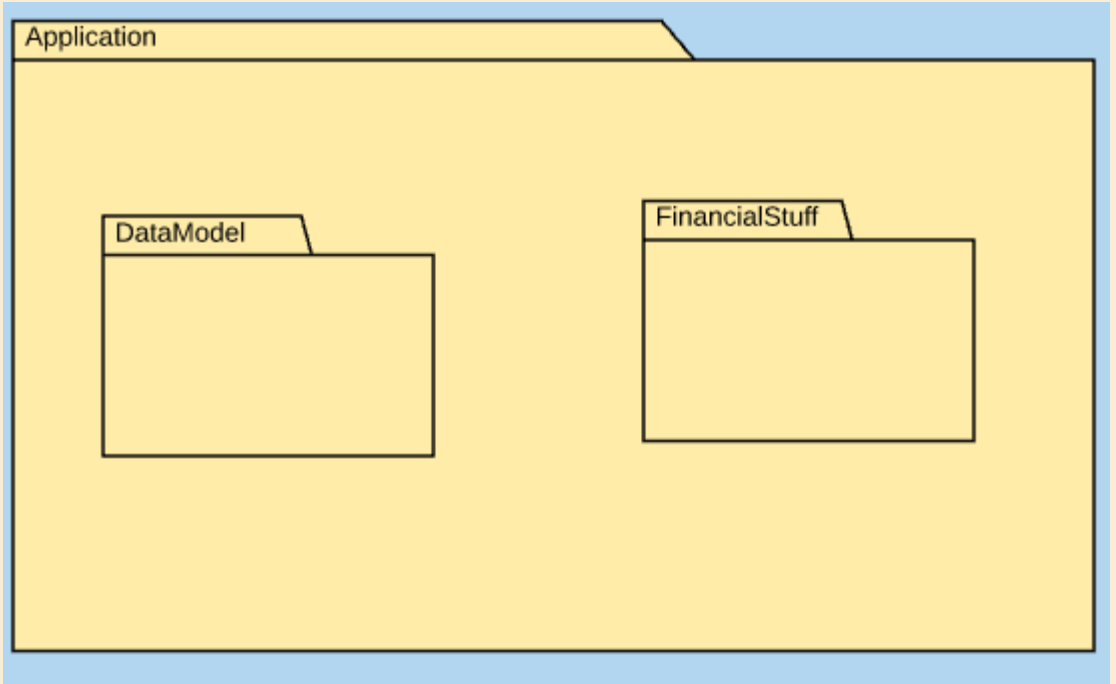
- By now… this model looks collapsed, that ‘FinancialObject’ Class doesn’t look good with the intention of the data model!

- So what about deleting that FinancialObject Class and putting all that financial stuff into another package?



Package Diagrams

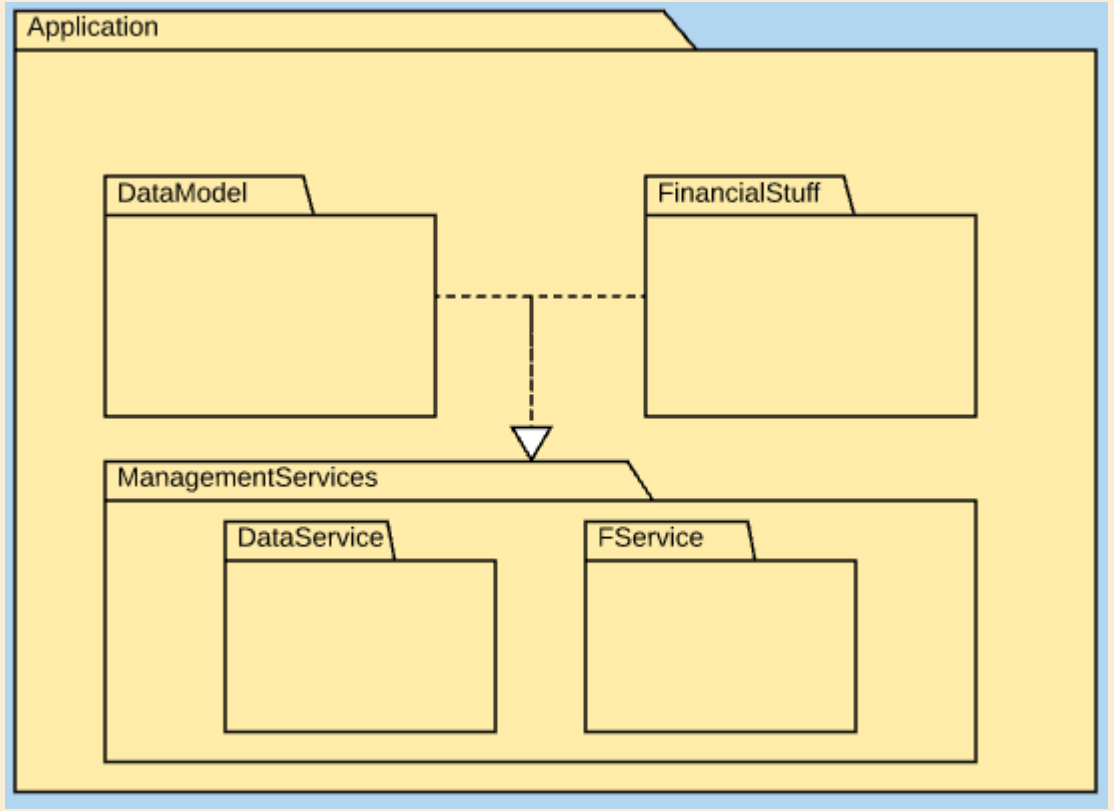
This basic package diagram shows that the DataModel and the FinancialStuff of the circus Project will be sepparated



Package Diagrams only show packages, not the classes inside as the class diagrams are the ones that are perfect for that purpose

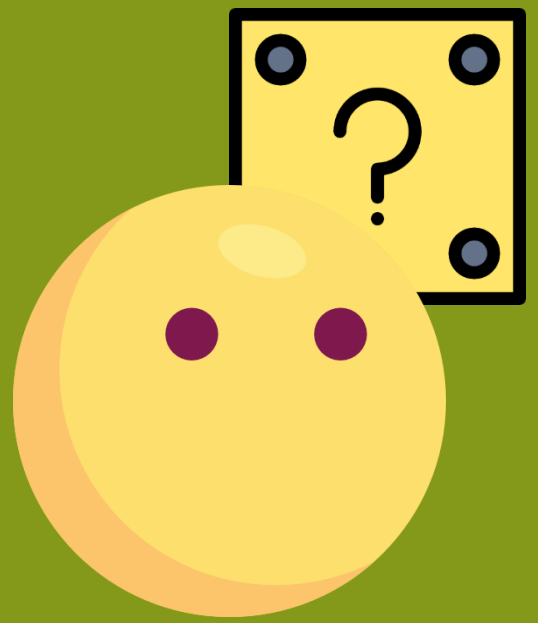
Package Diagrams - Relationships

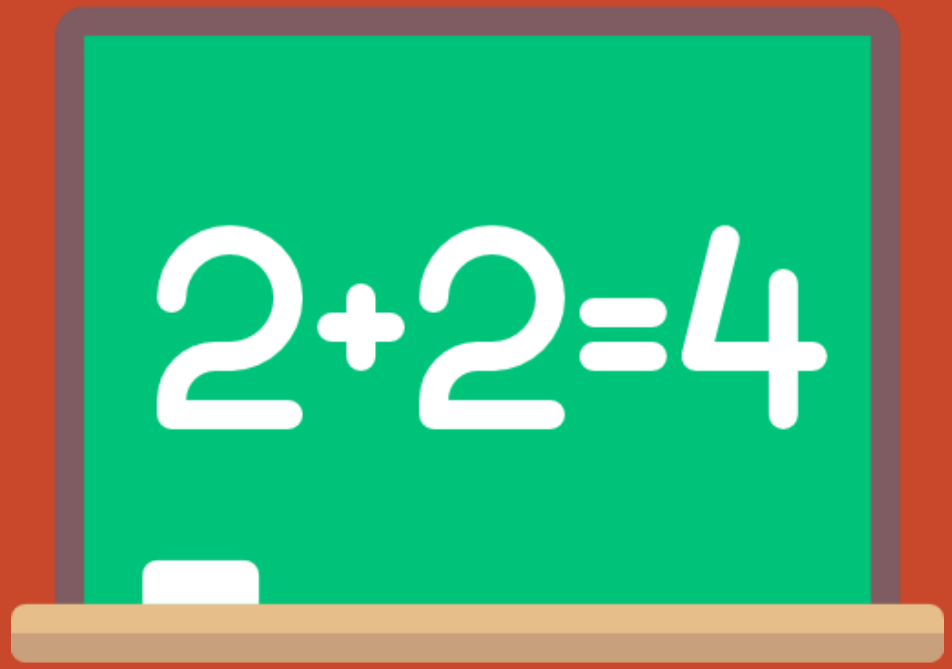
The white arrow with a dotted line indicates that a package interacts with another.



DataModel and FinancialStuff only contains encapsulated data, the management services are the ones that provide, modify and creates the instances of the classes inside of those packages

And now?





Component Diagrams

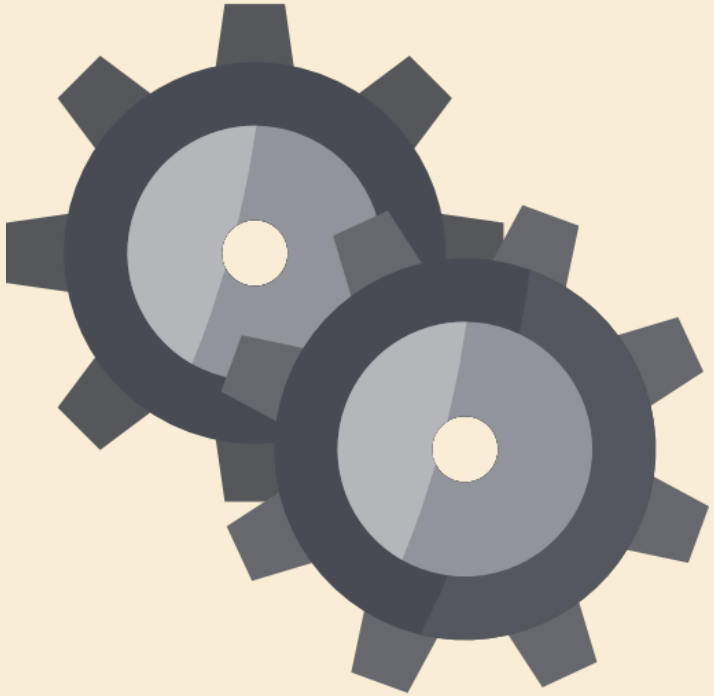
Contains components and their relationships

Eh?

Components

- Represent reusable, repetible and replaceable parts of your software.

- Normally there's a strong relationship between components and interfaces



Components

1) Structure of code

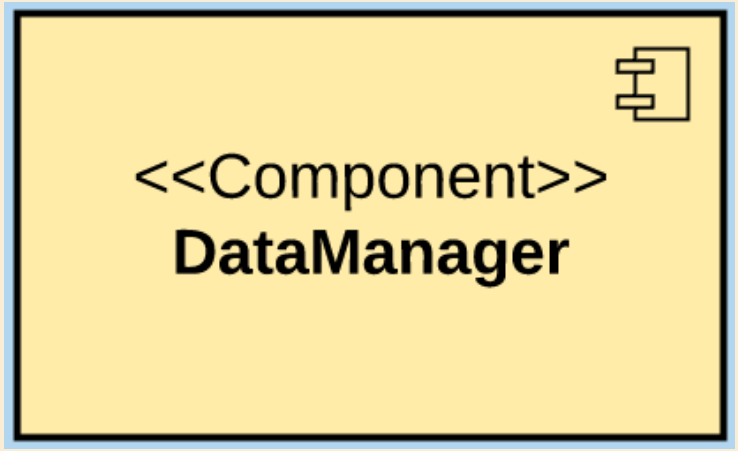
2) Hide detail of specification [Code]

3) Show Relationships between files

4) Specify which files become executables

Components

- You draw them like this

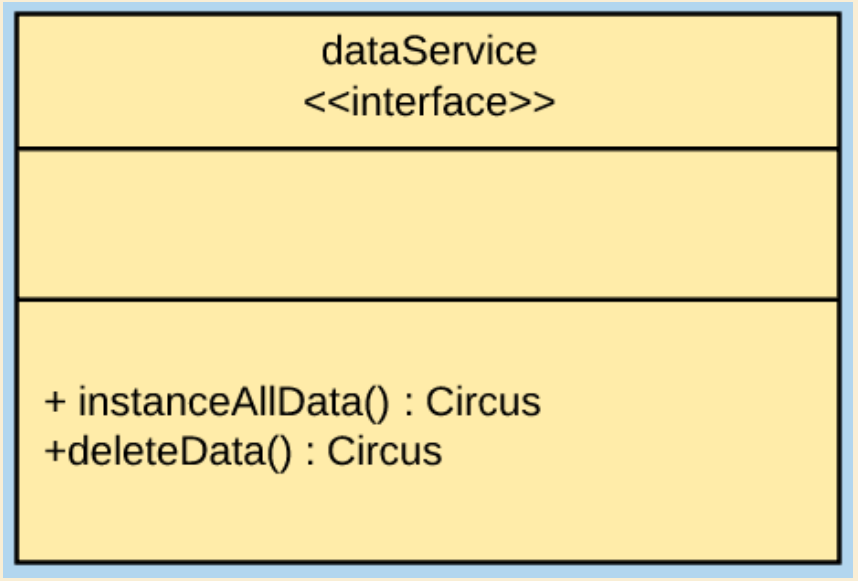


STOP

We need to clarify some things about interfaces!

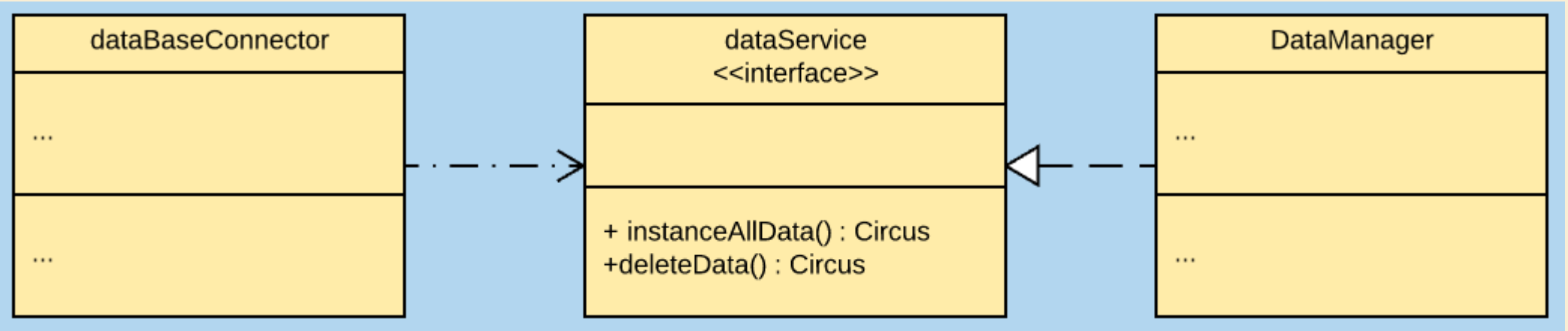
Interfaces!

- In class diagrams and in general UML you can model interfaces by drawing them like this



Interfaces:

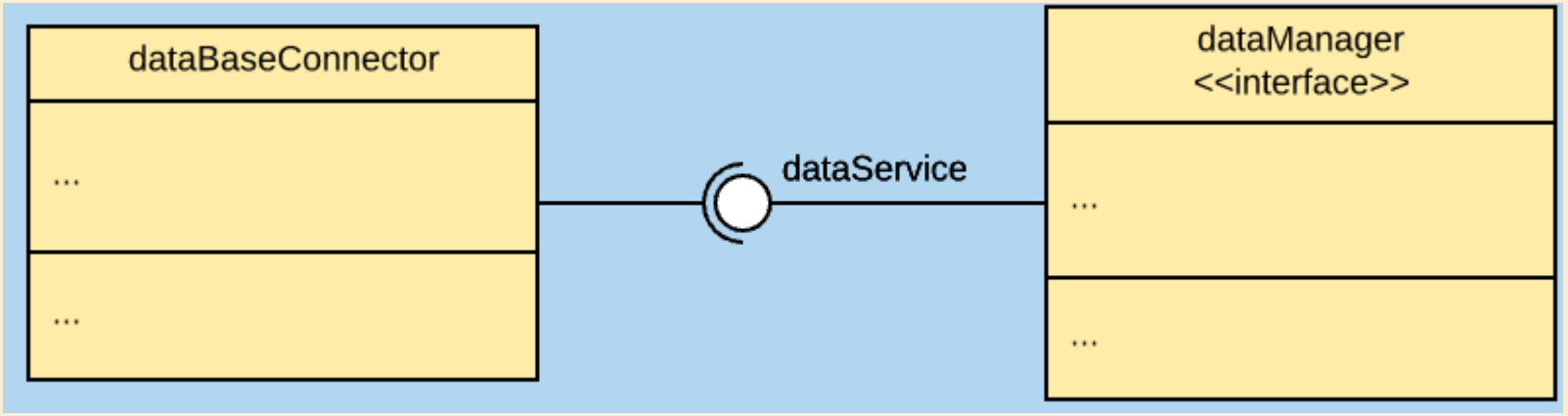
- Some classes can be dependent (left class dataBaseConnector) or implement (right class dataManager) an interface, you clarify that by putting dotted arrows.



dataBaseConnector depends on dataService interface and dataManager implements dataService

Interfaces:

- You can redraw this like (the interface is now a simple dot)



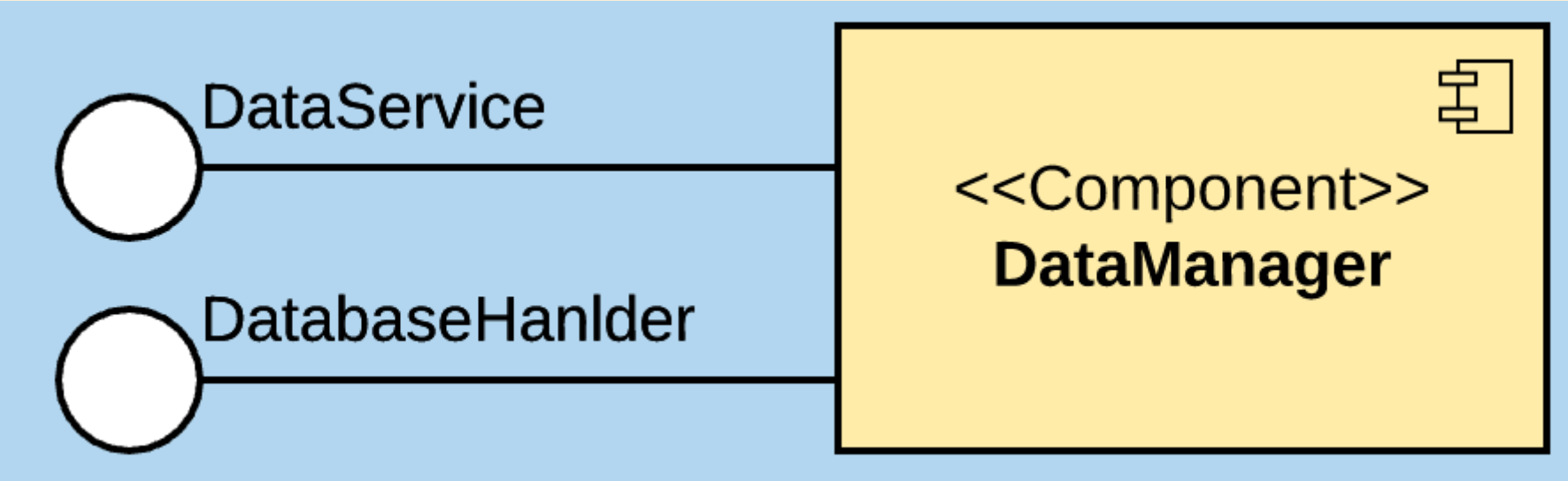
dataBaseConnector depends on dataService interface and dataManager implements dataService

CONTINUE

With our explanation about component diagrams!

Components and interfaces

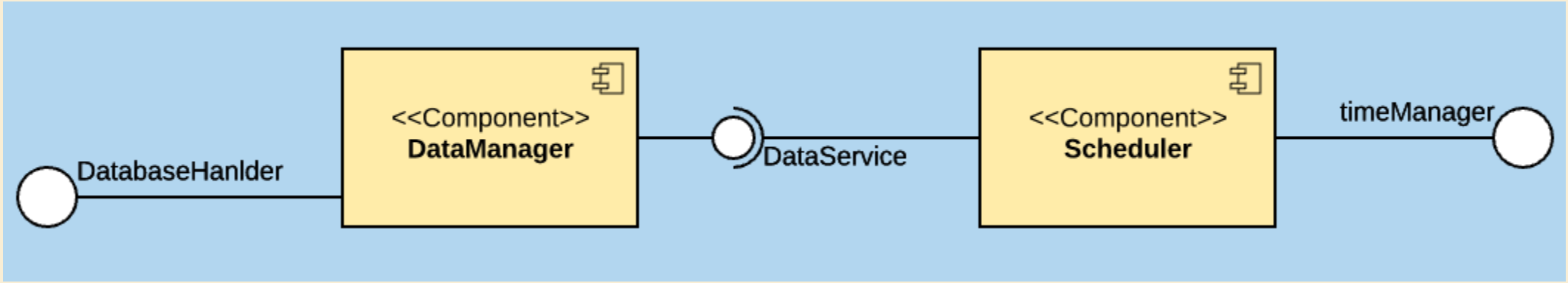
- Each component can implement one or more interfaces (remember their properties)



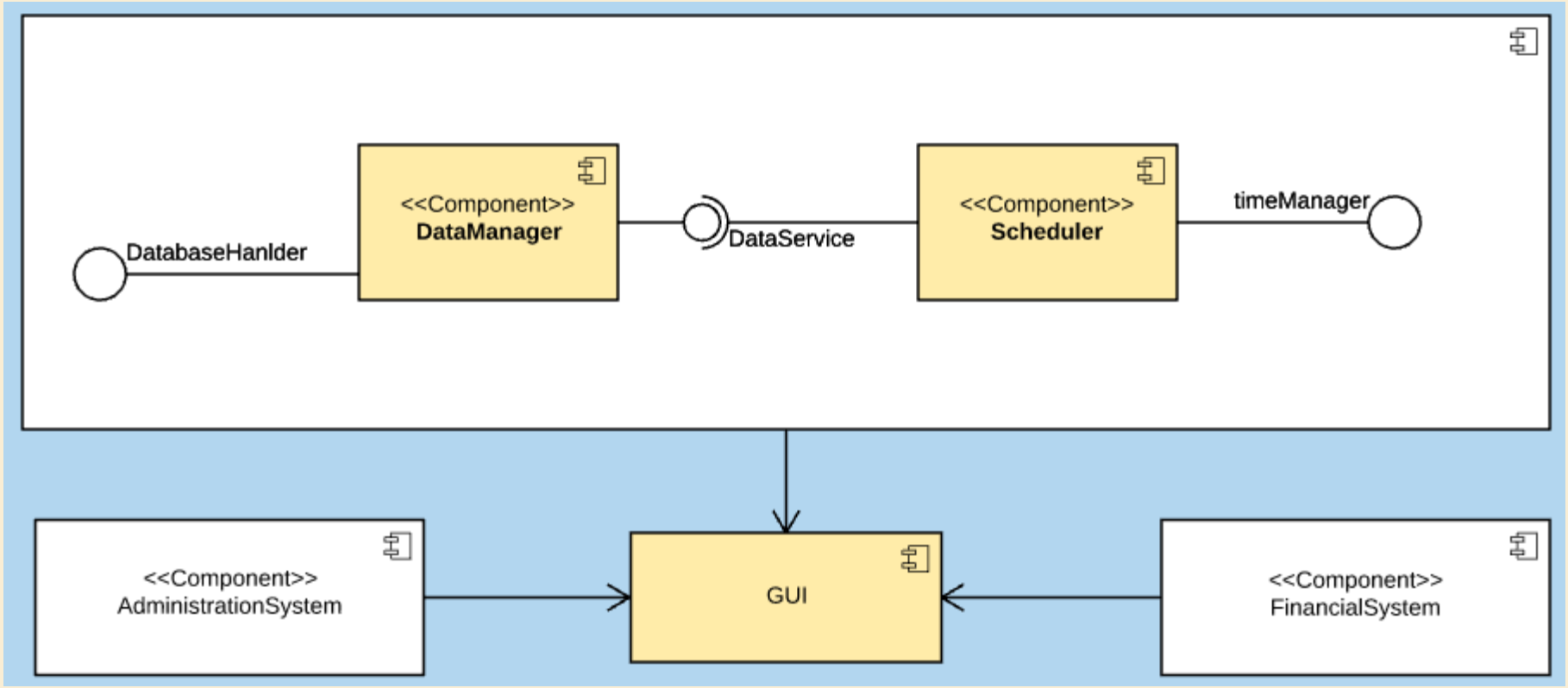
The component DataManager implements the interfaces DataService and DataBaseHandler

The Diagram

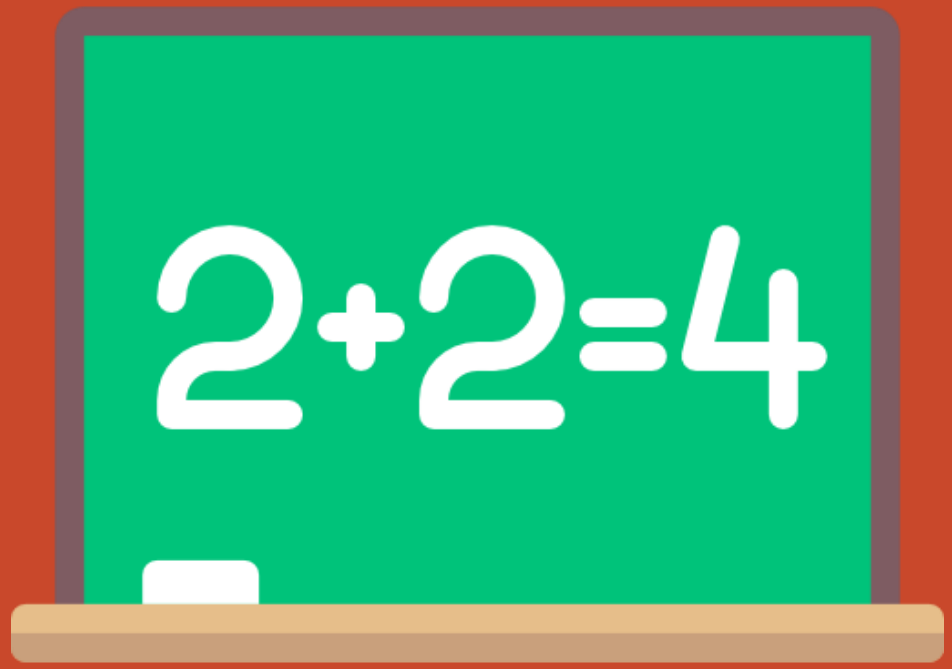
Simple design example! *(here less is not more)*



Adding more things...!



Think about adding components in FinancialSystem and GUI...

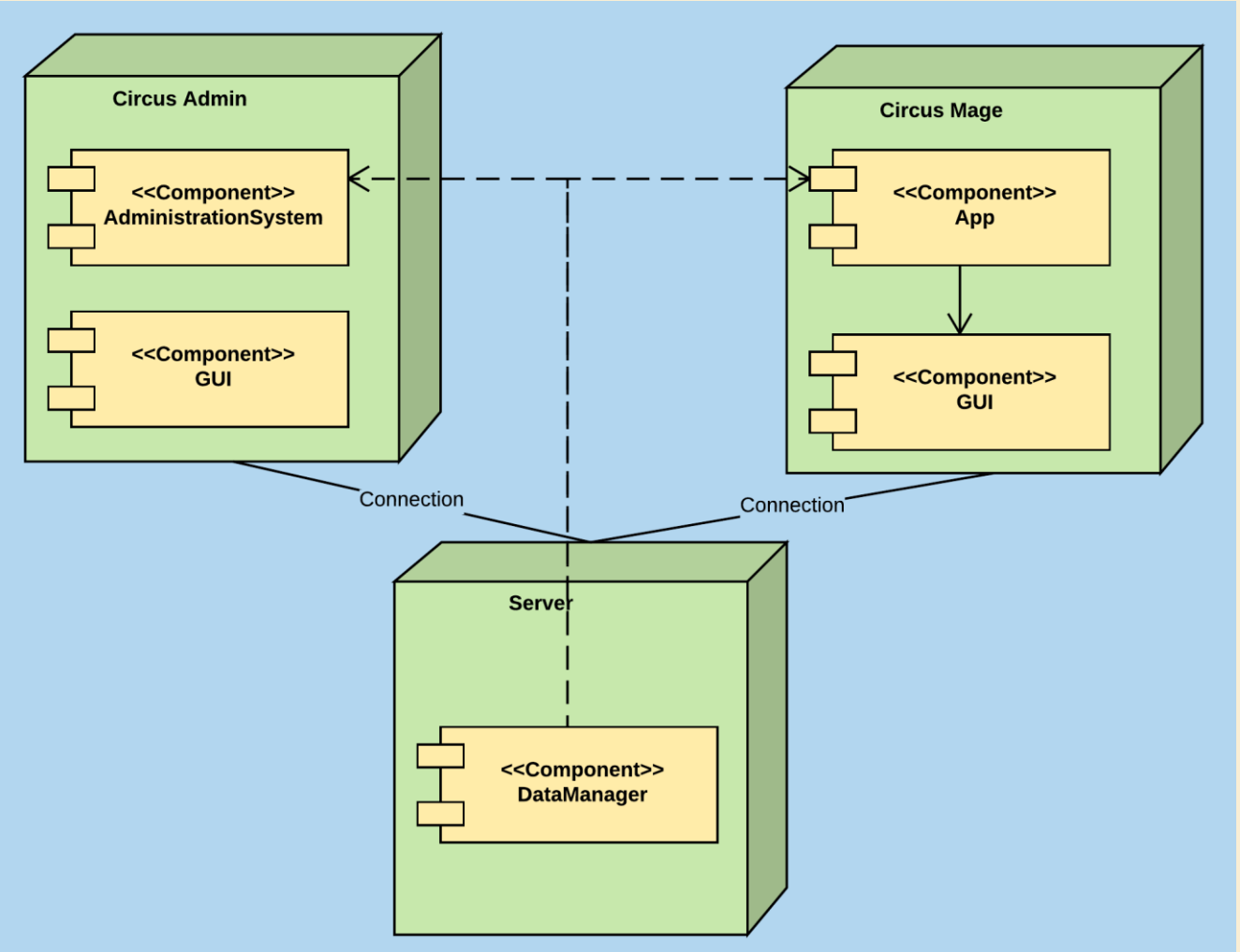


Deployment Diagrams

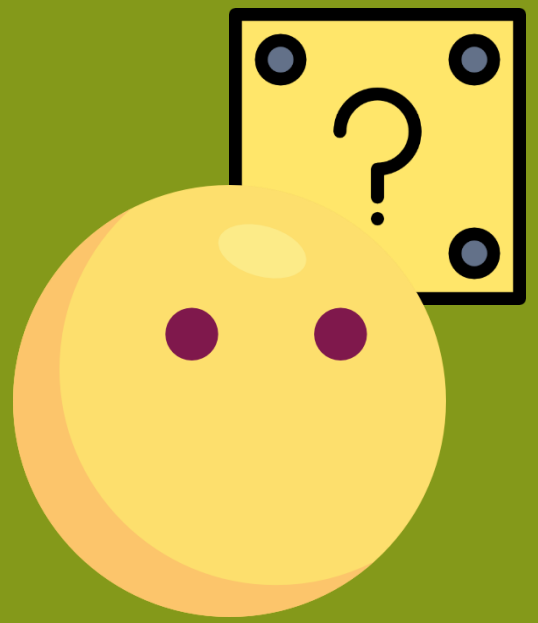
They show the system runtime at different nodes

Deployment Diagram

- Each Green square is what we call a *node*
- Inside each node you put the components working on it
- You can point relationships between components and nodes

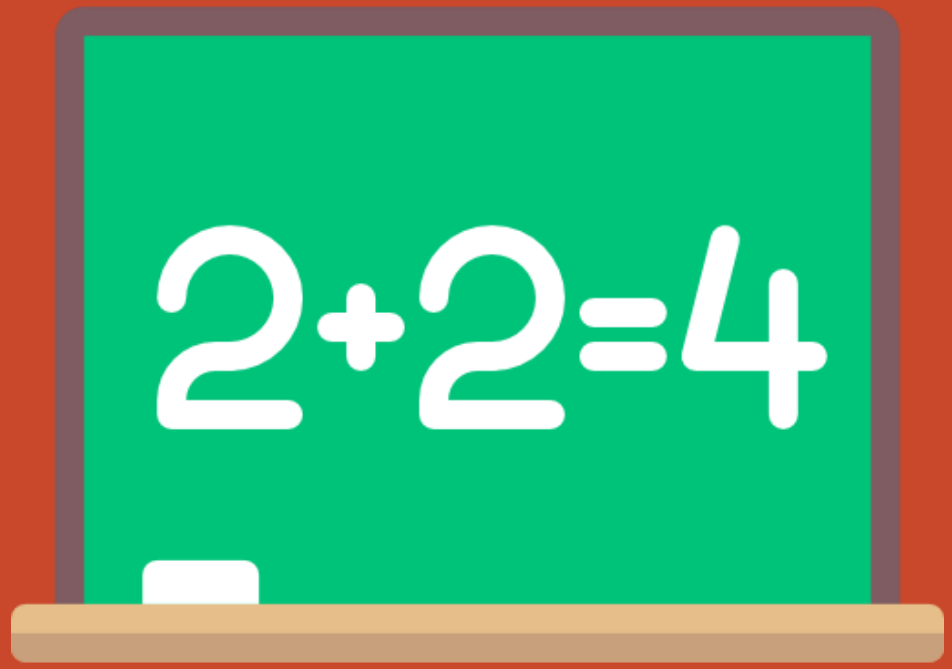


What about dynamic parts of the software?



Behavioral Diagrams

They model dynamic parts of the system in order to:



State Diagrams

Shows the sequence of states (control flow from one to another) and shows the transition from one another

States

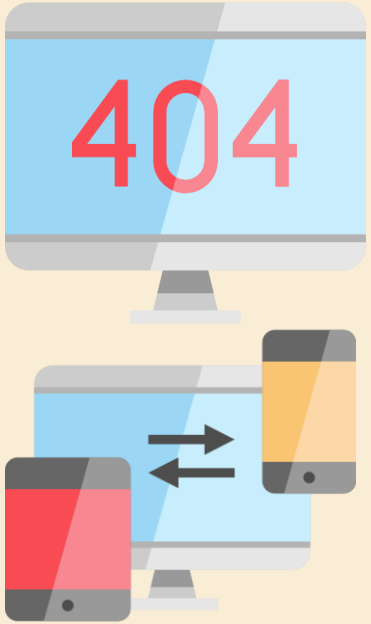
- Think about what we as humans feel for example:

Those changes are what we call *an event*

System States

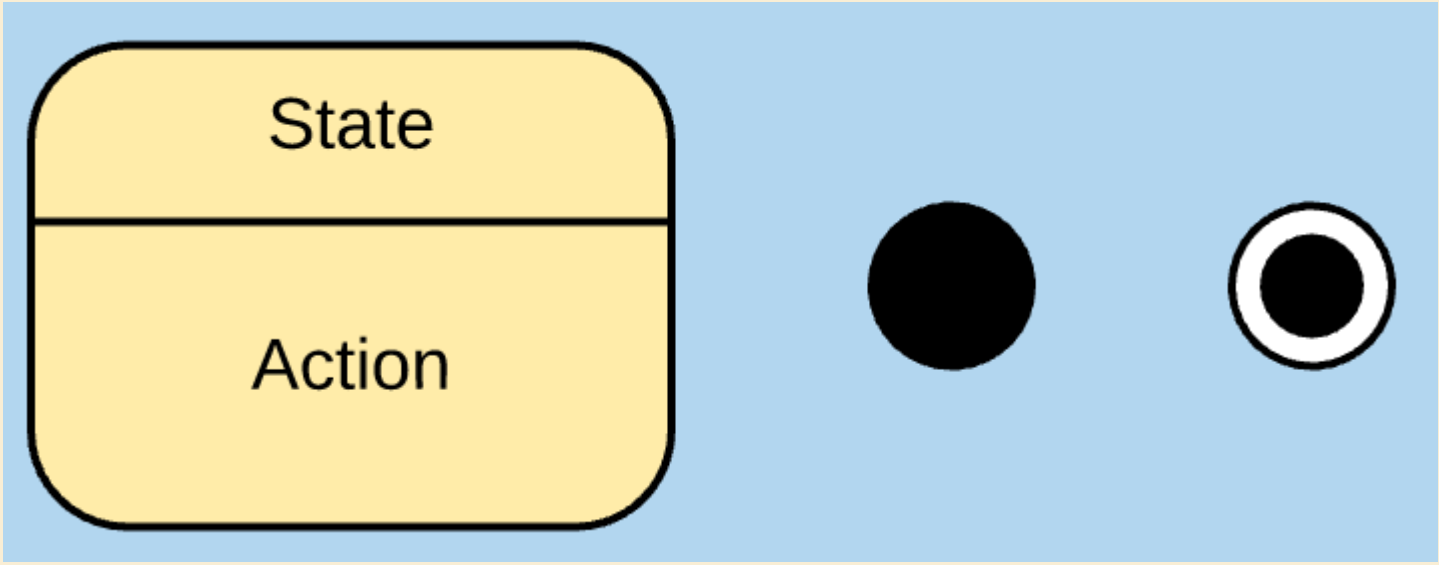
- We think about what's happening in the system, if it's **idle, waiting for something, running something, busy…**

- Whatever the system does at a given time can be considered an state



System states

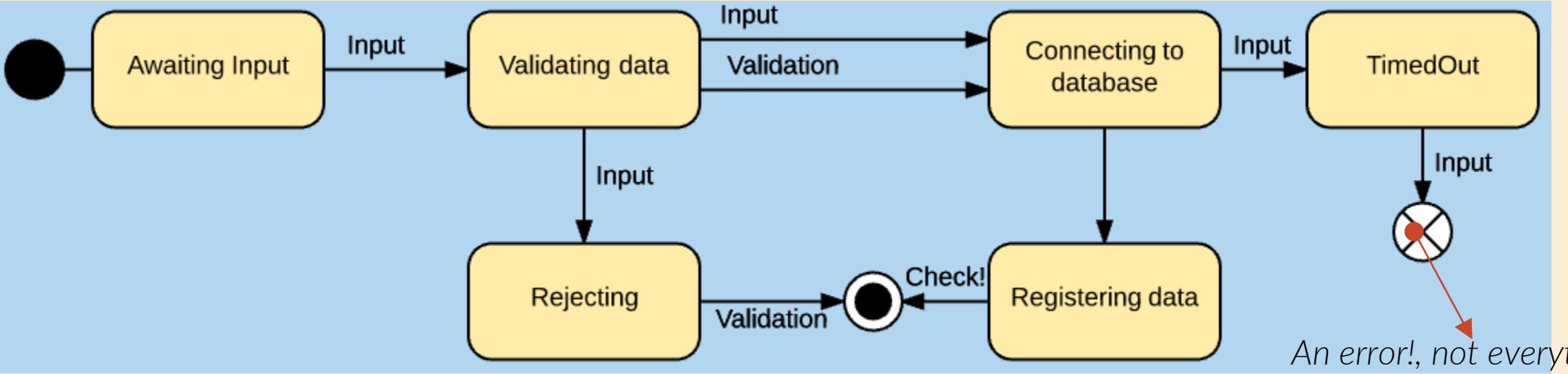
These are the icons for the state diagram in UML



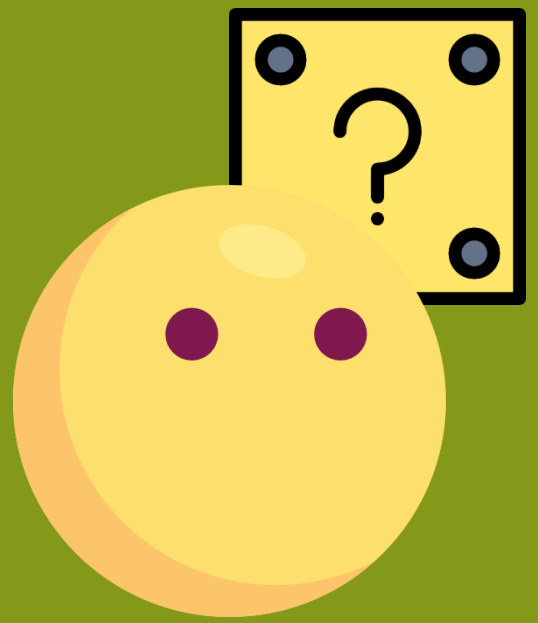
From left to right: A) State, B) Start, C) End

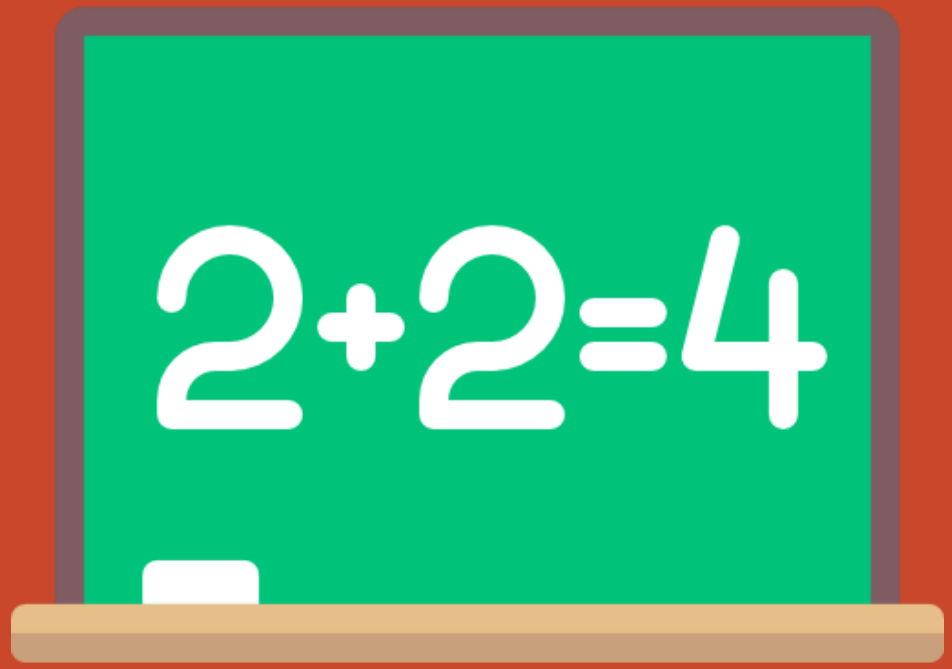
Simple Diagram

- Shows the flow of adding something to the system (example: The administrator adding an artist).



What about changing from one state to another?





Activity Diagrams

Simplified versions of what happens at a opperation or process

Activities

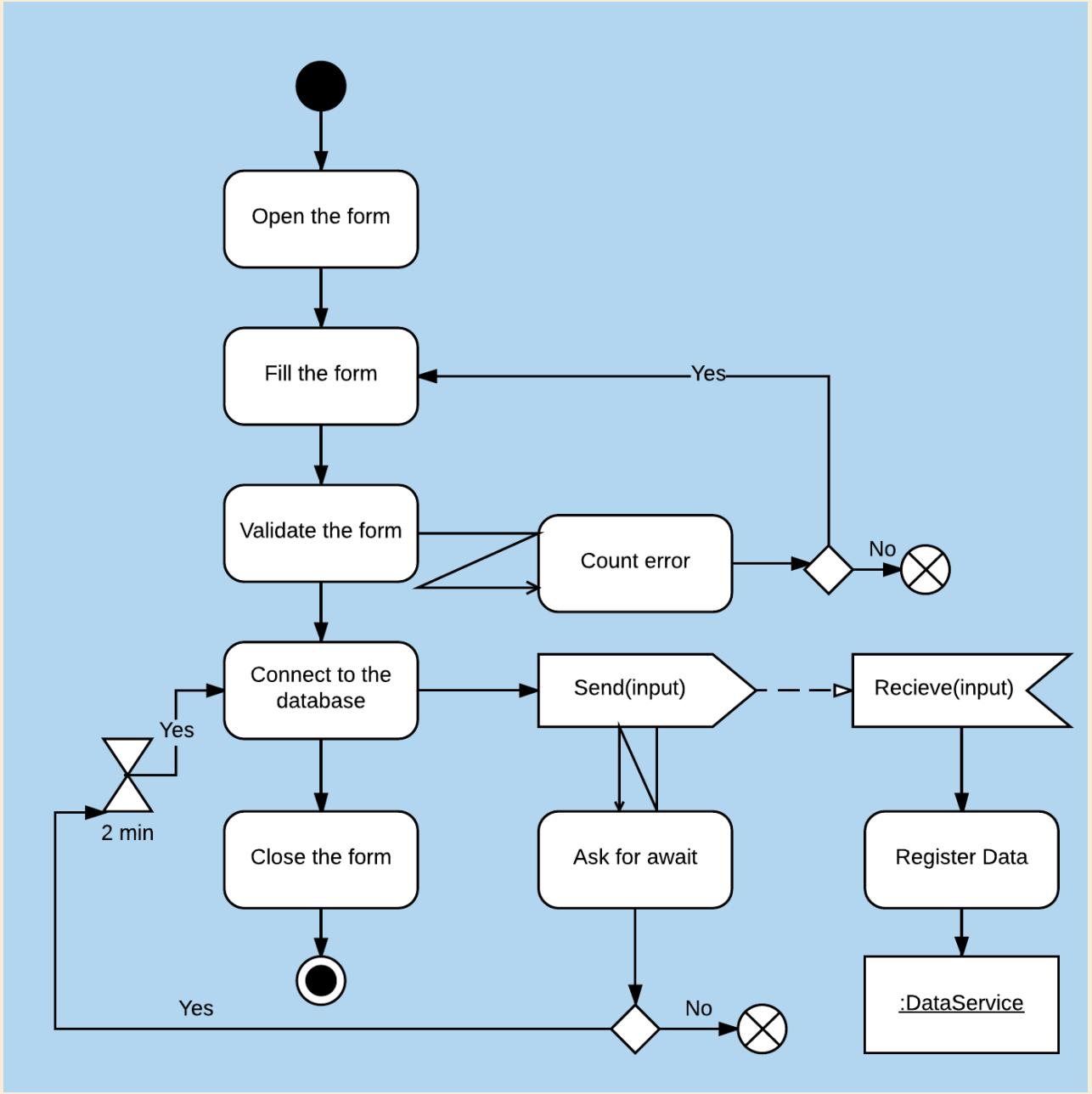
- What the system has to do, we normally think about *methods* when we want an output for a given input, this is the same. (TO DO)

- They have a strong relationship with what we normally do with flux diagrams

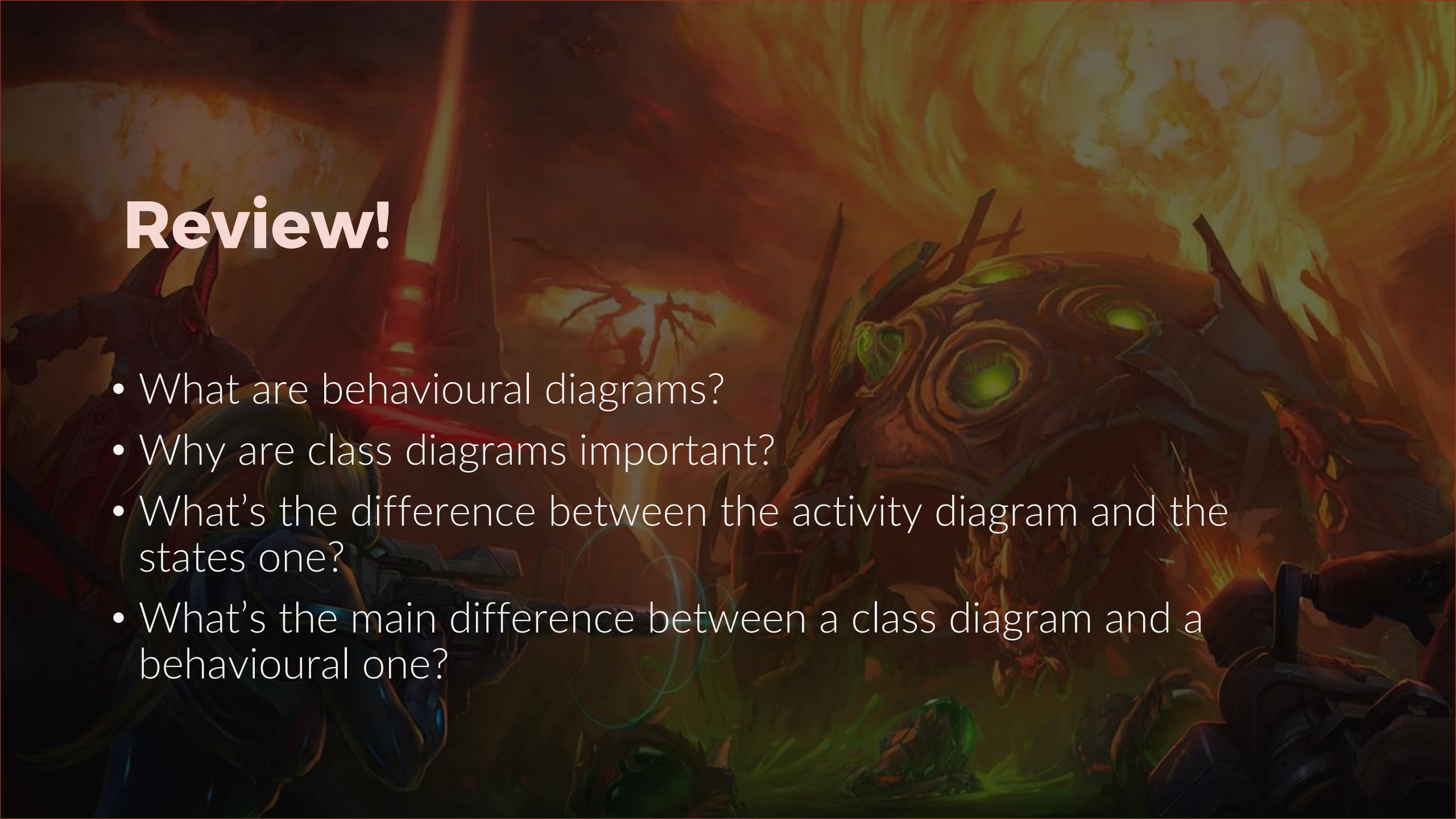
Activity Diagram Ex

- We have more things to take into account like *timers, exceptions and decisions*

- We can model our use cases with this!




One Shot Review



Review!

- What are behavioural diagrams?
- Why are class diagrams important?
- What's the difference between the activity diagram and the states one?
- What's the main difference between a class diagram and a behavioural one?

References

- [SOMMERVILLE] Ian Sommervile. *Software Engineering 9th Edition*
- [SCHMIDT] Richard Schmidt. *Software Development Architecture-Driven Software Development*
- [LARMAN] Craig Larman. *Applying UML and Patterns*
- [IBM] IBM Knowledge Center – UML
- [SCHMULLER] Joseph Schmuller. *Teach yourself UML in 24 hours*


Class has died... for today!