# Recap on Object Oriented Concepts

Parinya Ekparinya

Parinya.Ek@kmitl.ac.th

# Acknowledgement

The content of the following slides are partially based on the listed material as follows:

- Object-Oriented Patterns & Frameworks by Dr. Douglas C. Schmidt
- Object Oriented Programming Introduction by K. Ravi Chythanya
- Object-Oriented Programming by Mimi Opkins
- Object Oriented Programming by Mr Cleary

# Outline

- Procedural Programming
- Object Oriented Programming
  - Encapsulation
  - Inheritance
  - Abstraction
  - Polymorphism

# Procedural Programming

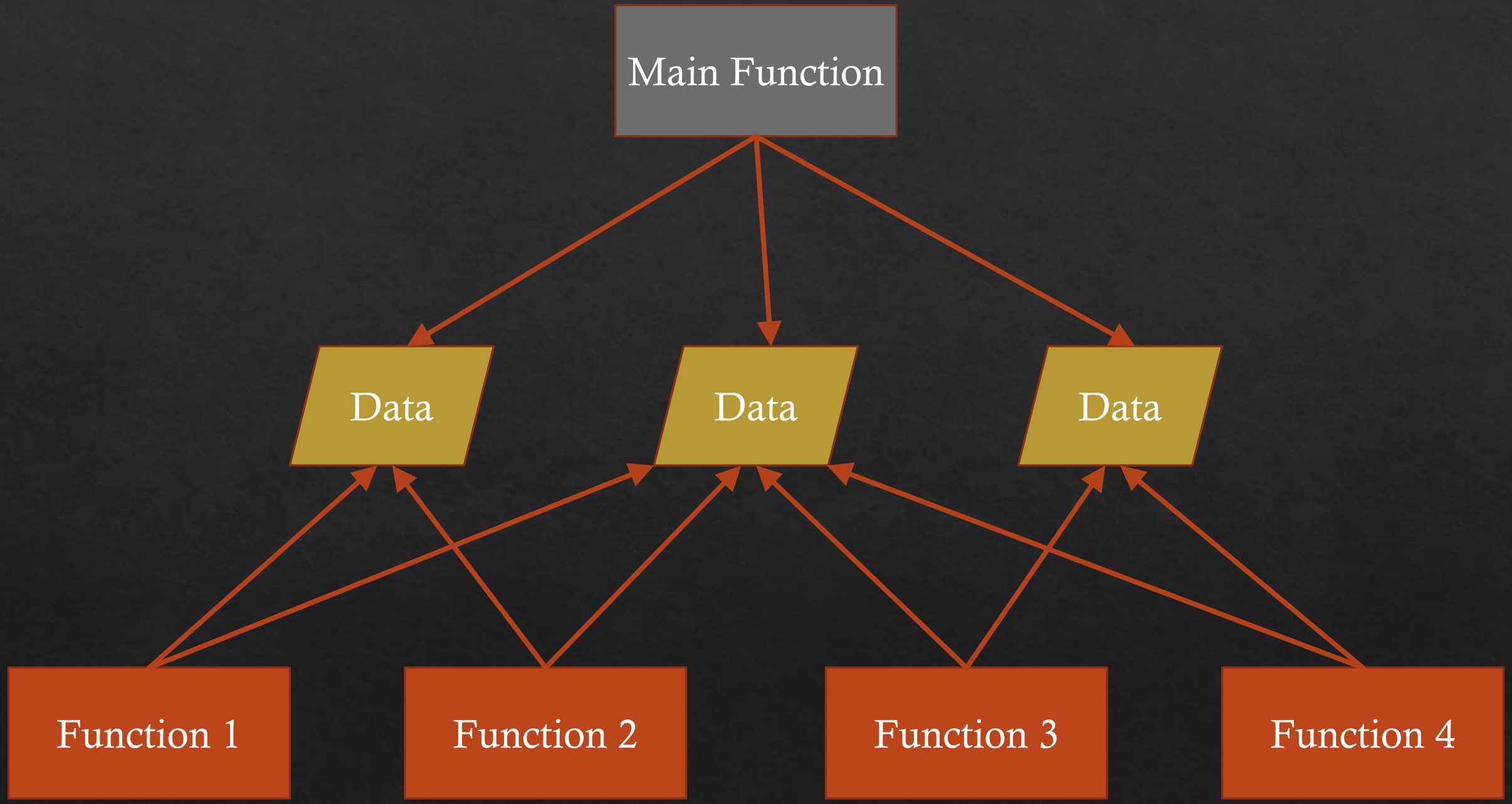
# Procedural Programming

- The code of a program is typically divided into groups of smaller parts of code called **procedures**/functions/subroutines.
- Each procedure/function/subroutine typically consists of a series of computational steps/statements to perform a specific task.
- Each procedure/function/subroutine may take input(s) and may return output(s).
- Procedures/functions/subroutines can access/manipulate global variables and their own local variables (including the data that are passed to them).
- One procedure/function/subroutine can be called many times in one program.
- Software is designed by organizing groups of functions and/or passing data among them.
- Example: Fortran, Cobol, Pascal, C

# Problems of procedural programming

**Warning:** The followings are mostly opinion-based. Some may disagree with them.

1) The importance is given to operations (procedures/functions/subroutines) rather than the data.
2) The data is often separated from the procedures as the paradigm does not provide a common approach to bind the data and operation together. Often, data is global.
3) Data access restrictions cannot be associated directly with the data. Data access restrictions are typically needed to be explicitly defined by operations.
4) It is difficult to represent real world objects.


Main Function
Data
Data
Data
Function 1
Function 2
Function 3
Function 4

# Object Oriented Programming

# Object Oriented Programming

- The code of one program is divided into smaller parts of code called **objects**.
- Each object typically contains both its data (state) and operations (behavior).
- The access to members of one object can be controlled by the definition of an object itself.
- Software is designed by using objects that interact with one another.
- Many programming languages support OOP. Example: C++, Python, PHP, Java, C#

# State and Behavior

| Object | State | Behavior |
| --- | --- | --- |
| Duck | Name, Color, Weight | Quack, Walk, Fly |
| Car | Make, Model, Color | Accelerate, brake, turn |
| Game character | Class, Race, Name, Skills, Hit points | Cast, Fight, Flee, Heal |

# Objects

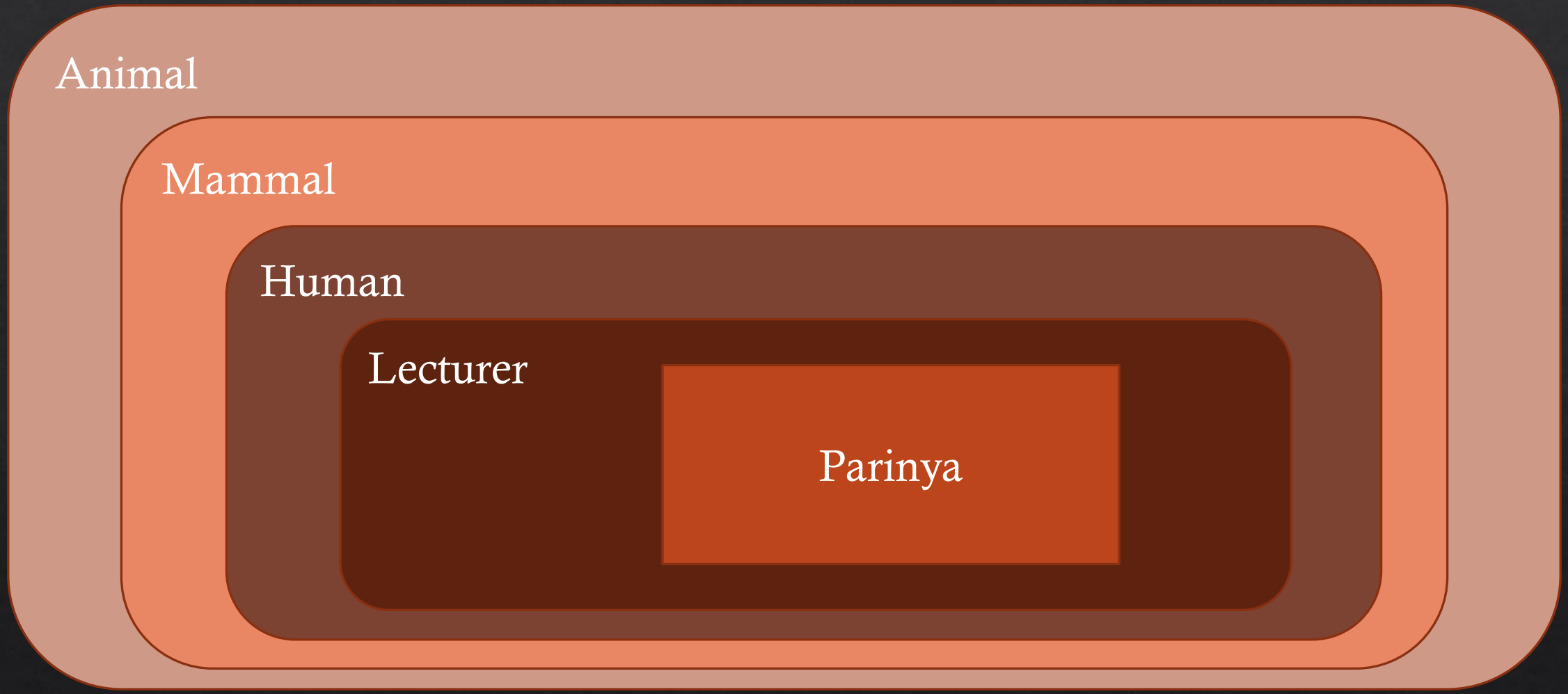
Objects in OOP can be thought of representations of real-world objects.

| :Duck |
| --- |
| name = “Donald”<br>birthday = 1931-03-13<br>weight = 1.2 |
| + quack()<br>+ walk()<br>+ fly()<br>+ swim() |

| Parinya:Person |
| --- |
| firstName = “Parinya”<br>height = 1.7<br>weight = 65 |
| + walk()<br>+ jump()<br>- think() |

# Class

- A class is a data structure that can hold both variables and functions.
- Classes provide a convenient clustering mechanism for related functions (methods) along with related variables (attributes).
- A class can be thought of a blueprint or prototype for creating objects.
- Programmers can use the same class and therefore the same code over and over again to create different objects.
- Classes hide their data representation from all code except their own classes.


Animal
Mammal
Human
Lecturer
Parinya

# Class and Objects (Instances)

- When we write an Object-Oriented Program, we design and construct a set of classes.
- When the program runs, objects are created from those classes.
- We need to create the right set of classes to accomplish what the program needs to do.

# Human analogy

- There are people of different names, ages, races, height, and weight.
- Despite their individual differences, all people are members of the same category.
- In OOP terms, all people are in same class: the class of humans.
- In OOP terminology, each individual person is said to be an instance of the Human class.
- A group of people may communicate with each other to finish a project.

# The big idea of OOP

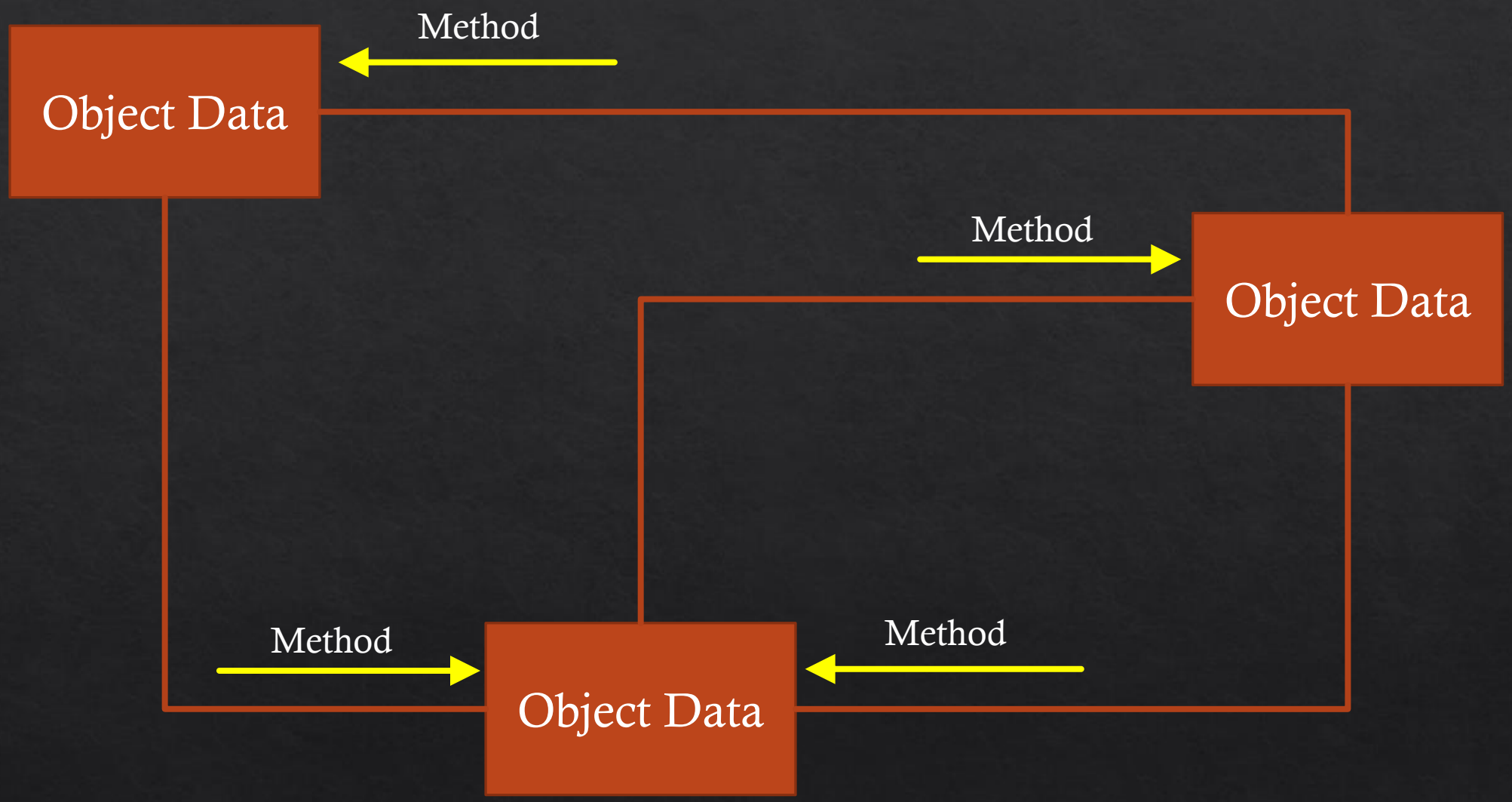
Alan Kay is considered one of the fathers of the idea of OOP. He once said:

*"I'm sorry that I long ago coined the term **objects** for this topic because it gets many people to focus on the lesser idea. The big idea is **messaging**."*

http://lists.squeakfoundation.org/pipermail/squeak-dev/1998-October/017019.html

# Object and Messaging

- Each object provides a service, or performs an action, that is used by other objects.
- Action is initiated in OOP by the transmission of a message to an object responsible for the action.
- The message encodes the request for an action and is accompanied by any additional information (arguments) needed to carry out the request.
- The receiver is the object to whom the message is sent. If the receiver accepts the message, it accepts the responsibility to carry out the indicated action.
- Fundamental concept in OOP is to describe behavior in terms of **responsibilities**.

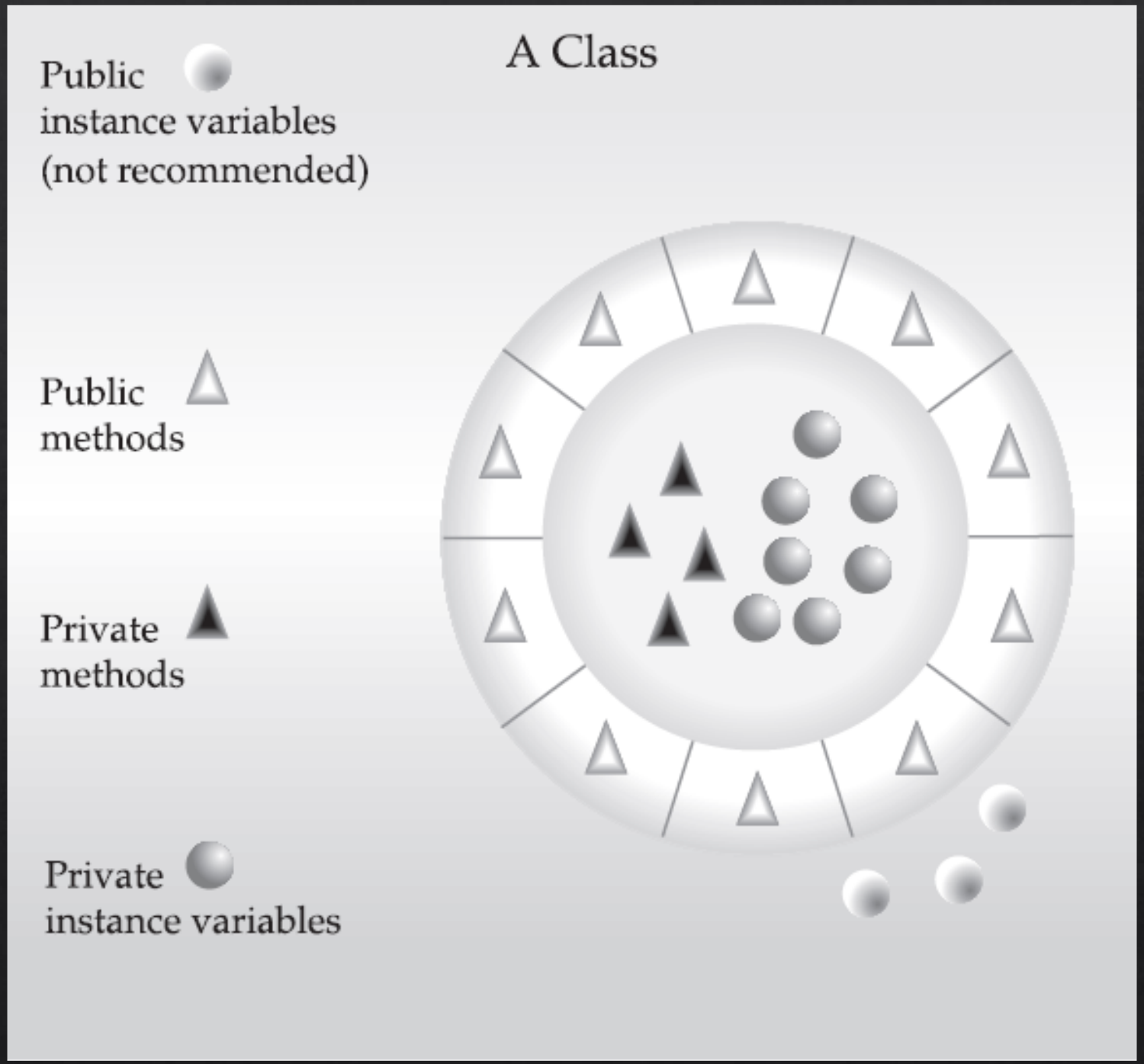

Method
Object Data
Method
Object Data
Method
Method
Object Data

# 4 Pillars of OOP

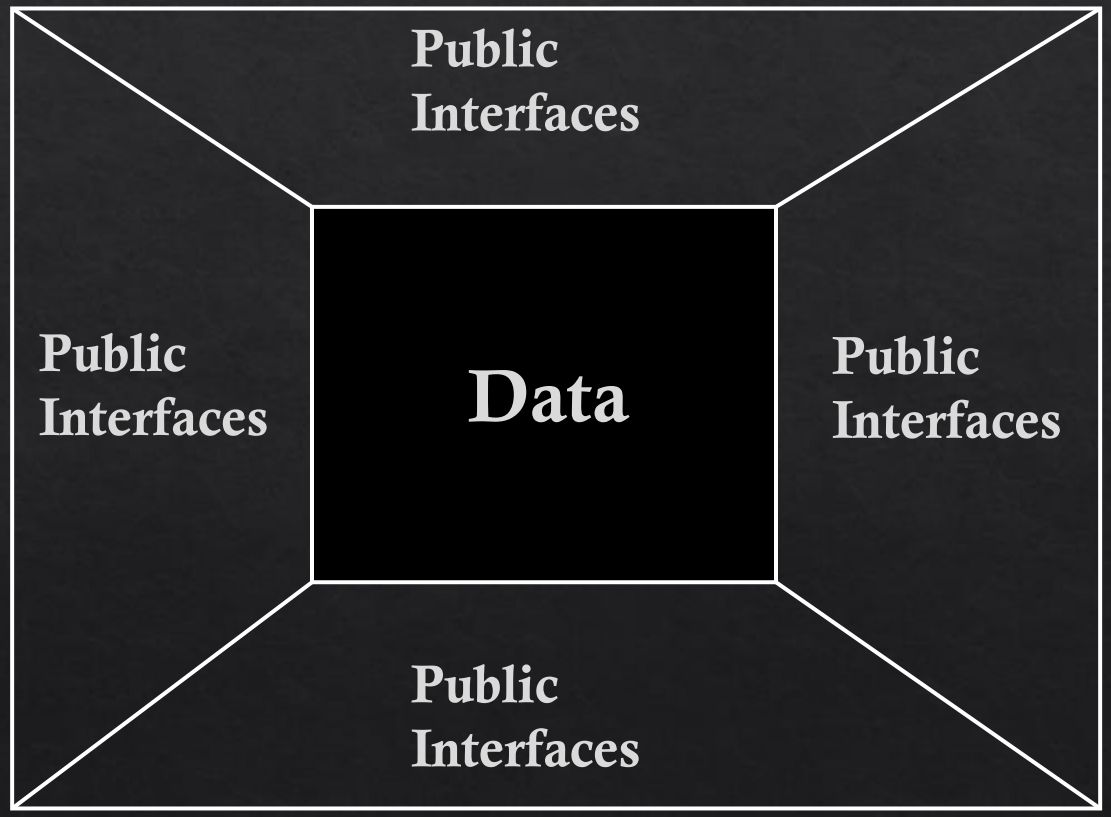
- Encapsulation
- Inheritance
- Abstraction
- Polymorphism

# Encapsulation

- Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
- One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper.


A Class
Public instance variables (not recommended)
Public methods
Private methods
Private instance variables

# Data Hiding

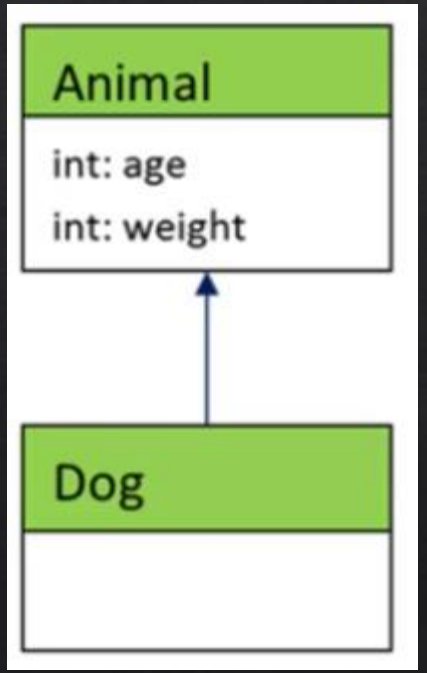

Public
Interfaces
Public
Interfaces
Data
Public
Interfaces
Public
Interfaces

# Inheritance

- Inheritance is the process by which one object acquires the properties of another class (class-based) or another object (prototype-based).
- A class that inherits from another class is a subclass or child.
- The class that gives the inheritance is the superclass or base class or parent.
- It supports the concept of hierarchical classification.
- Most knowledge is made manageable by hierarchical (that is, top-down) classifications.

# Example of Inheritance

- When using inheritance, make sure the 2 classes (parent and child) has a 'Is a' relationship.
- Example:
  - Dog is an animal.
  - Parent Class (Animal) and Child Class (Dog).
  - Dog is inheriting Animal attributes.

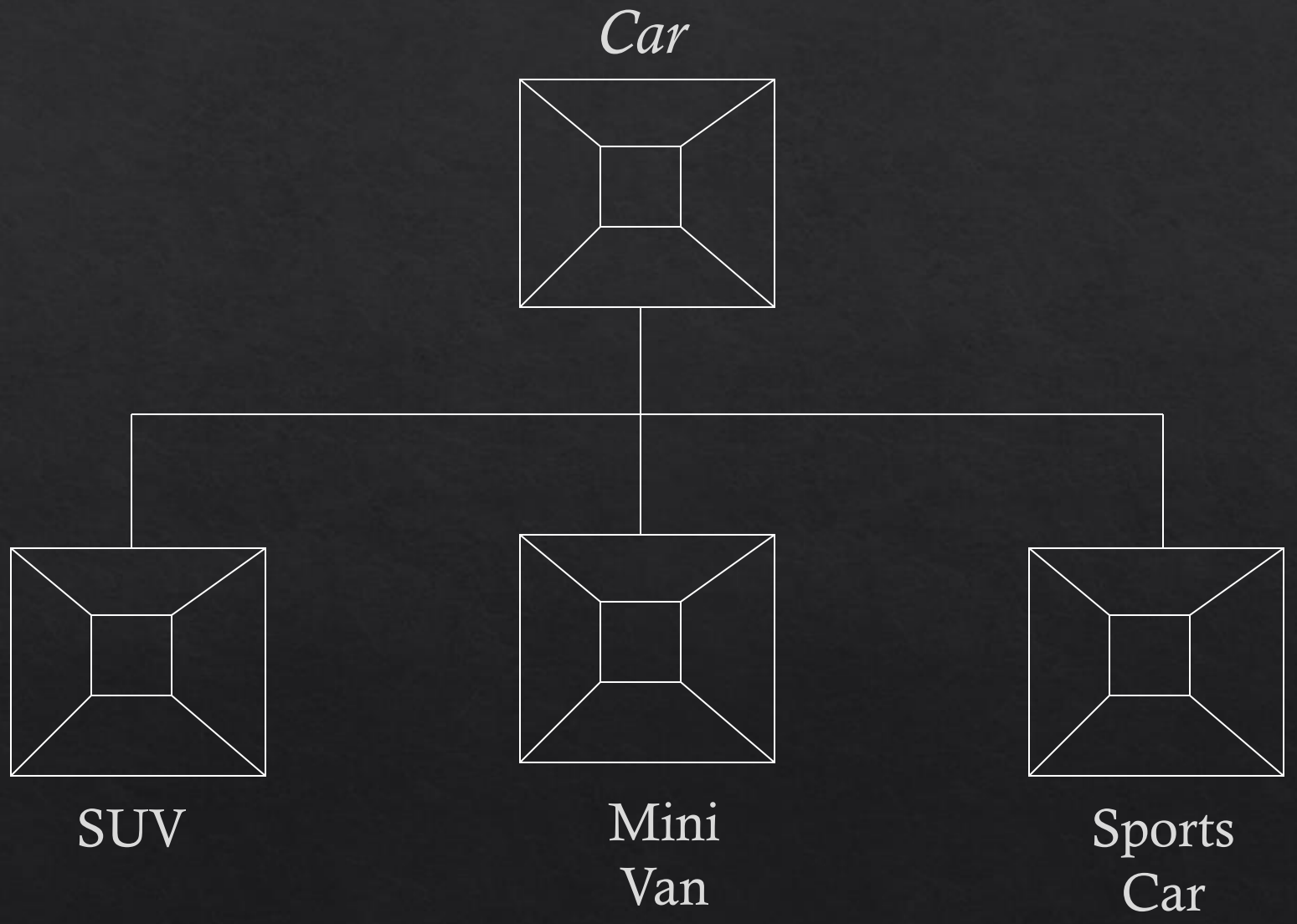


# Abstraction

- The data from a traditional process-oriented program can be transformed by abstraction into its component objects.
- A sequence of process steps can become a collection of messages between these objects.
- Thus, each of these objects describes its own unique behavior.
- One can treat these objects as concrete entities that respond to messages telling them to do something.
- This is the essence of object-oriented programming.

# Polymorphism

- Polymorphism (from Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.
- By inheriting its superclass’ features, an subclass is still a member of the superclass. It can execute all of it’s methods as well as those of the superclass.
- This ability to be both a member of subclass and a superclass is called polymorphism.
- In other words, a sports car is still a car.


*Car*
SUV
Mini Van
Sports Car

# Organizing Classes

- Determine if classes make up a class hierarchy with subclasses
- Functionality that is common to multiple classes can be put in super classes
- Changes to a superclass are automatically reflected in all their subclasses

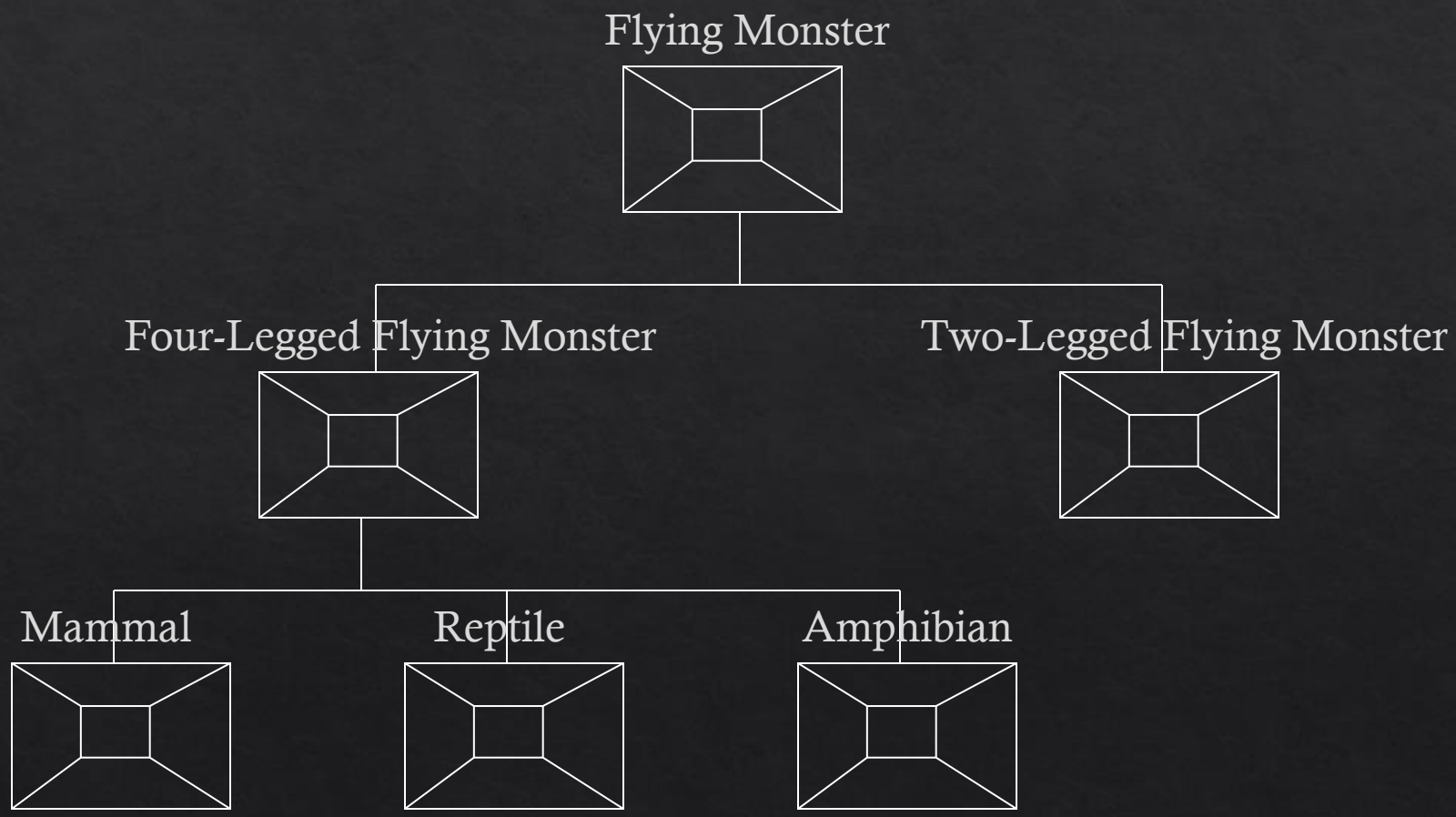
# Monster Class Hierarchy

Monster

Flying Monster

Swimming Monster

# Flying Monster Class Hierarchy



# Steps for Object-Oriented Programming Design

1. Break down objects to their smallest features
2. Look for commonality between the objects
3. Look for differences between the objects
4. Find the largest commonality between all objects
5. Put the remaining common objects together and repeat

# Recommended reading list

- The Object-Oriented Thought Process by Matt A. Weisfeld from chapter 1 to 9
- OOP - Learn Object Oriented Thinking and Programming by Rudolf Pecinovský
- Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D by Brett McLaughlin