

# Recap on Object Oriented Concepts

Parinya Ekparinya

[Parinya.Ek@kmitl.ac.th](mailto: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

C

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

# Procedural Programming

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

C

# Procedural Programming

និង code ដែលអាចរាយការណ៍ "procedures" or function or subroutines

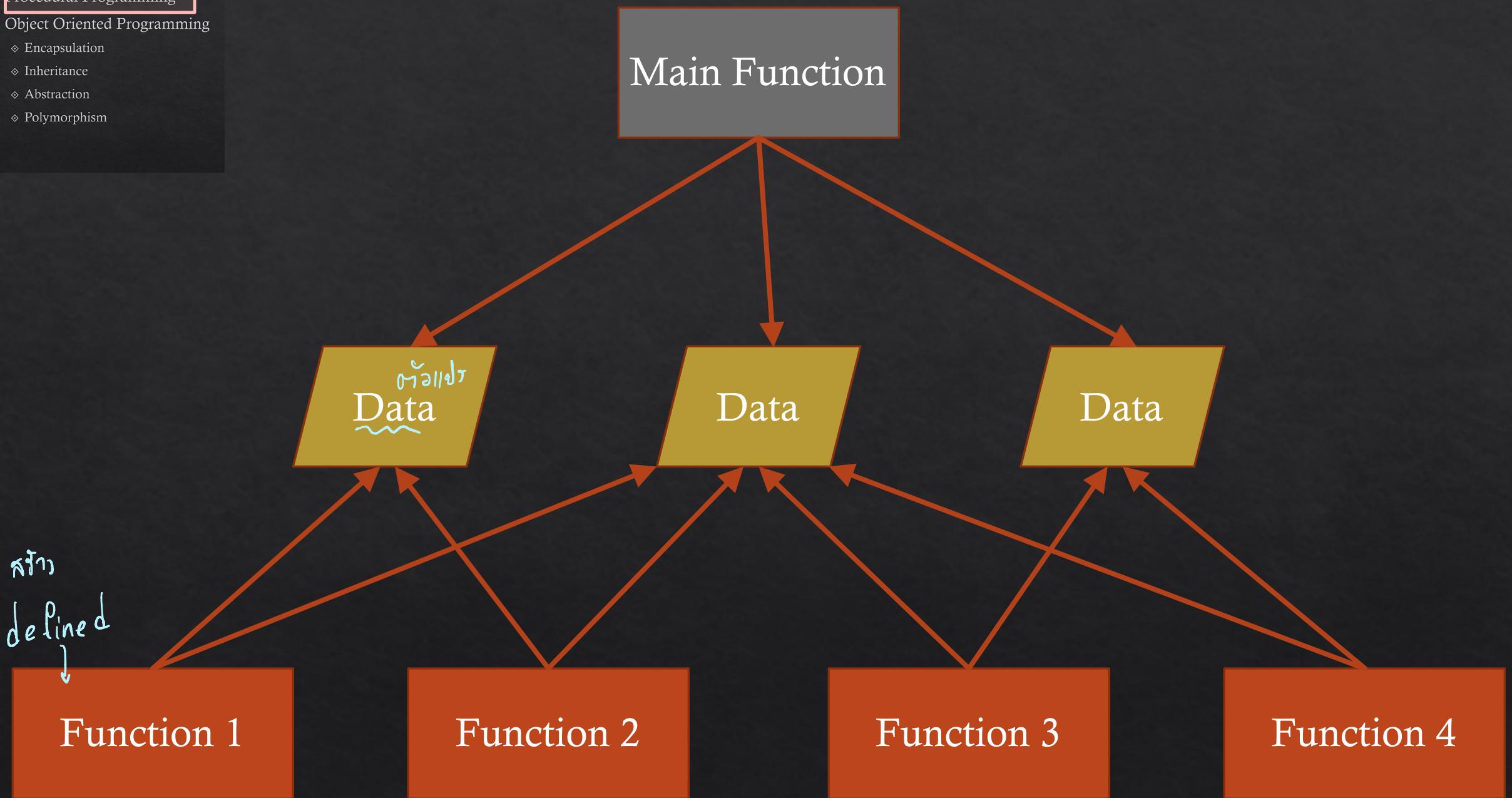
- ❖ The code of a program is typically divided into groups of smaller parts of code called **procedures/functions/subroutines**.  
កំសងម៉ែងក្នាំនានាជាម៉ោះទេរ៉ាំង
- ❖ Each procedure/function/subroutine typically consists of a series of computational steps/statements to perform a specific task.  
នាយក: ឯកសារនូវការបង្កើត និងសម្រេច និងសម្រេច និងសម្រេច
- ❖ Each procedure/function/subroutine may take input(s) and may return output(s).  
ឯកសារនូវការបង្កើត និងសម្រេច និងសម្រេច និងសម្រេច
- ❖ 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.  
ក្នាំនានាជាម៉ោះទេរ៉ាំង និងសម្រេច និងសម្រេច
- ❖ Software is designed by organizing groups of functions and/or passing data among them.  
ក្នាំនានាជាម៉ោះទេរ៉ាំង និងសម្រេច និងសម្រេច
- ❖ 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.  
Code កំណត់បន្ថែម →  
*behavior state*      *ធម្មការពិនិត្យ* →  
*គម្រោង*
- 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.  
Data គេហទ័រ →  
*ការងារ*      *Data នៃការងារ* →  
*function*
- 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.  
Model គែលការ →  
*គេលការ*      *គេលការដែលមិនត្រូវយក*

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



# Object Oriented Programming

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

# Object Oriented Programming

ແນວ Code ຍົມ ແລ້ວ object

- ❖ The code of one program is divided into smaller parts of code called **objects**.  
OOP ① + ② ກົດປຶກສຳມາດນີ້
- ❖ Each object typically contains both its data (state) and operations (behavior).  
ຝາກໆ ປຸ່ນ ຄວບຄຸມໄດຍ ຕັ້ງ object ໂດຍ
- ❖ The access to members of one object can be controlled by the definition of an object itself.  
ກົດປຶກສຳ object ສັນໜິກ ເລື່ອ object ສ່ວນລົງທະບຽນ
- ❖ Software is designed by using objects that interact with one another.  
ກາຫຼັກສຳໃນແກຣດີ OOP
- ❖ Many programming languages support OOP. Example: C++, Python, PHP, Java, C#

# State and Behavior

Object	State <i>(data)</i>	Behavior <i>(operation)</i>
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.

<i>object</i>
<u>:Duck</u>
name = “Donald”
birthday = 1931-03-13
weight = 1.2
+ quack()
+ walk()
+ fly()
+ swim()

<i>object</i>	<i>class</i>
<u>Parinya:Person</u>	
firstName = “Parinya”	
height = 1.7	
weight = 65	
+ walk()	
+ jump()	
- think()	

# Class

กํ

6

1

2



if access data from → ຂໍວາງກວນນຸ່ມເຊົາ

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.      න්‍යා සේට තො උස්ස තුන් මෙහෙයුව

# 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

នៃជាន់ object និង

object ក្នុង

ជាន់ object និង

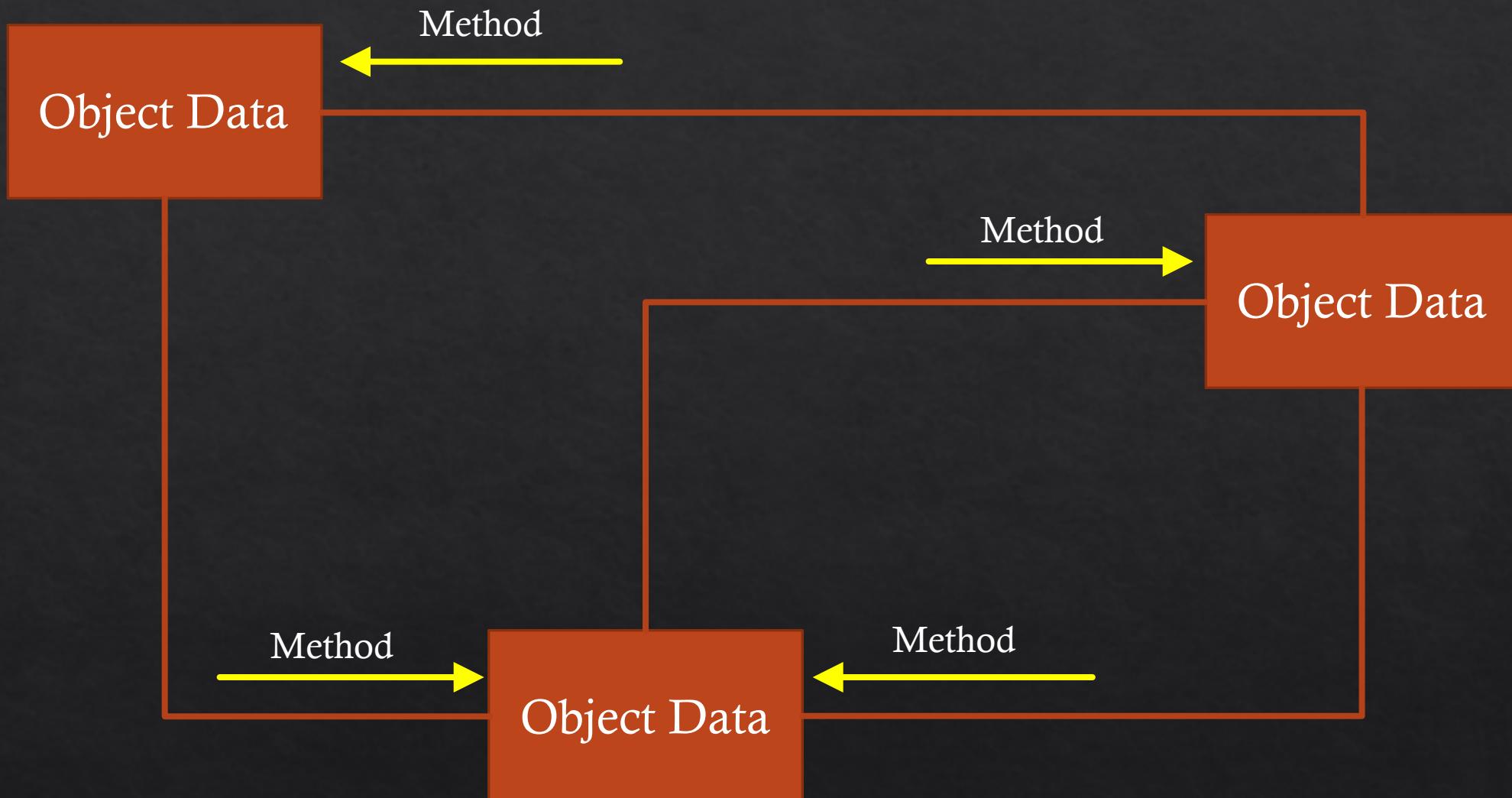
- ◆ Each object provides a service, or performs an action, that is used by other objects.  
នៃ message ទូទៅ object ដោយ, នឹងមាន action របស់
- ◆ Action is initiated in OOP by the transmission of a message to an object responsible for the action  
ចំណាំ នូវការបង្កើត / មានជូននូវការ action នេះ
- ◆ The message encodes the request for an action and is accompanied by any additional information (arguments) needed to carry out the request.  
នៃ message
- ◆ 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.  
object និង
- ◆ Fundamental concept in OOP is to describe behavior in terms of responsibilities.

object និង ការបង្កើត / នូវការ

គ្រប់គ្រង bound context រាយការ

និង ការបង្កើត

# ឧបករណ៍ក្រសួង OOP



# 4 Pillars of OOP

- ❖ Encapsulation
- ❖ Inheritance
- ❖ Abstraction
- ❖ Polymorphism

+ public  
- private

ສຶກສາ ໂດຍ ດັບ ດຳ ເພີ້ມ

## Encapsulation ແກ້ໄຂ ດຳ ປະໂຫຍດການນຸ່ມ

- ❖ 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.

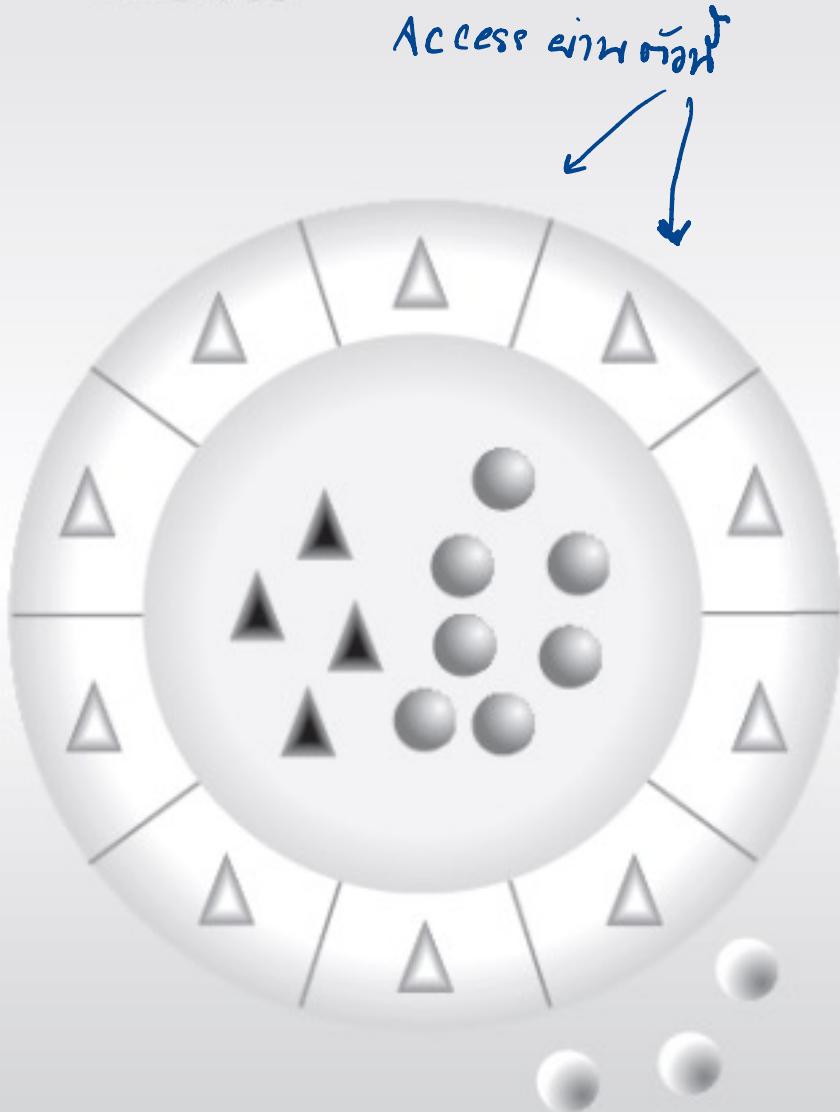
Public  
instance variables  
(not recommended)

Public  
methods

Private  
methods

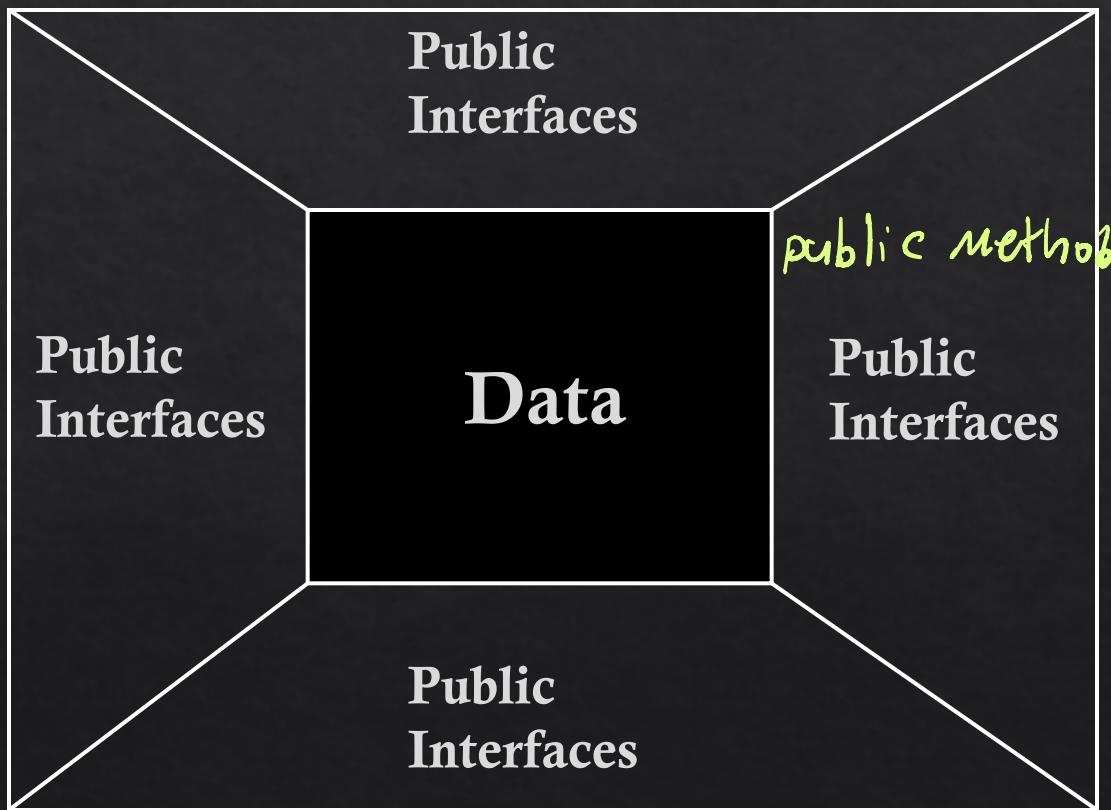
Private  
instance variables

## A Class



# Data Hiding

no data



នៅ នំបុរាណា can សេចក្តីជា  
ម្នាស់

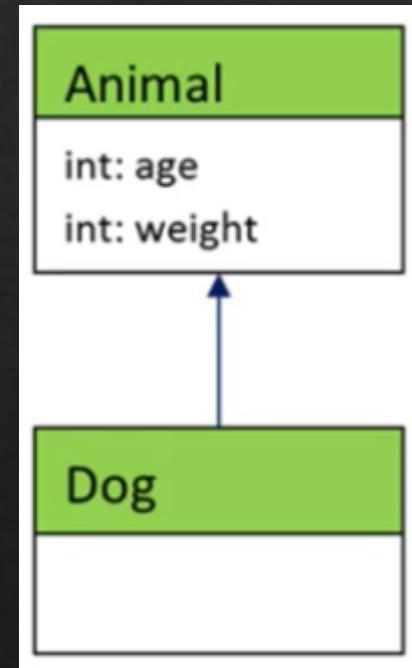
## Inheritance => reuse code ងារពិនិត្យបានប្រើប្រាស់

ក្រុមការពីទីបាននិងមិនទីបាន class នឹង ធ្វើត្រូវក្នុងការរាយទីនា ដើម្បីប្រើប្រាស់

- ◆ Inheritance is the process by which one object acquires the properties of another class (class-based) or another object (prototype-based).  
*class នៃសេចក្តី គាន់ class ណា តែង ឬ " "*
- ◆ A class that inherits from another class is a subclass or child.  
*ឯកជាមួយ ផ្ទាល់ពី ក្រុមការ*
- ◆ The class that gives the inheritance is the superclass or base class or parent.  
*ឯកជាមួយ ផ្ទាល់ពី ក្រុមការ*
- ◆ It supports the concept of hierarchical classification.  
*ឱ្យអាជីវកម្មភាព*
- ◆ 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.



ສຳເນົາ Class ກໍ່ມາແກດຕາຈົບການຂາວ່າງ

ອີງ Inheritance ດະສົມຜົນ ເພື່ອດີເລີ້ມການຂາວ່າງ

ສຳຄັນການການຕັ້ງຕິໂນໂລມ (No Go code ຍົດ)

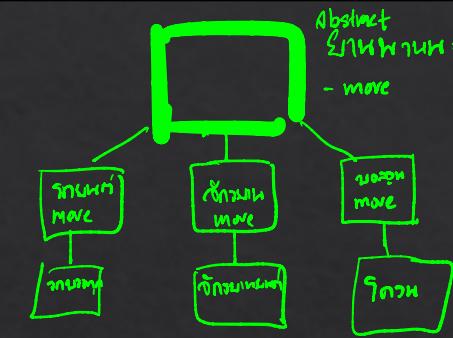
ໄກເກົດກາ high level ຍົດ

# 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.  
*ດັບຕັ້ງກົດ*
- ❖ 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.

ਜ਼ਿਆਦਾ abstraction  
ਜ਼ਿਆਦਾ

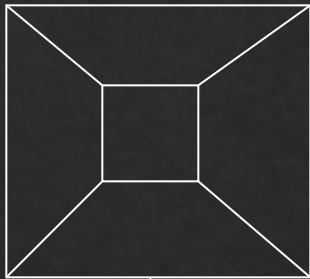
# Polymorphism



- ❖ 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.

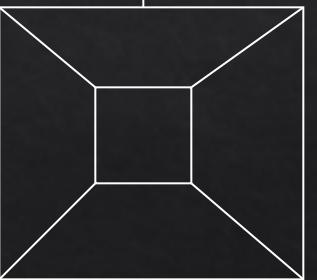
from abstraction

*Car*

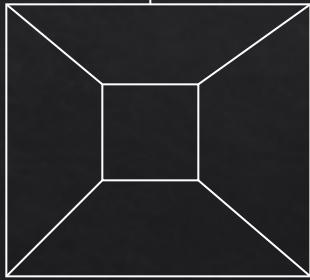


behavior from object

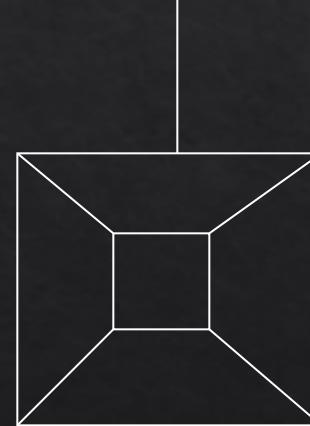
to



SUV



Mini  
Van



Sports  
Car

ମାତ୍ରାବ୍ୟବ୍ୟବ class

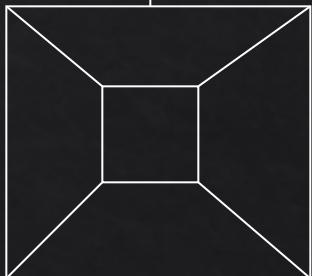
# Organizing Classes

- ❖ 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

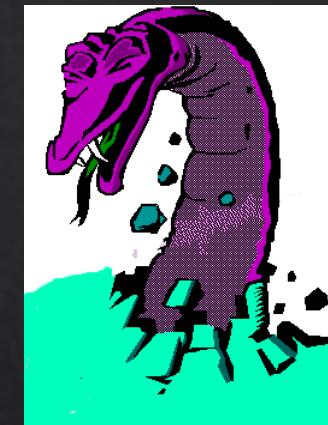
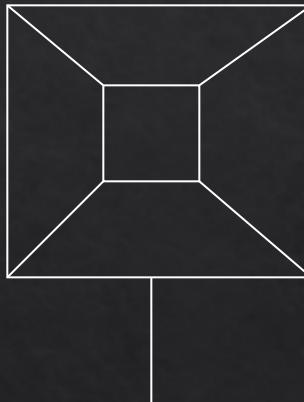
# Monster Class Hierarchy



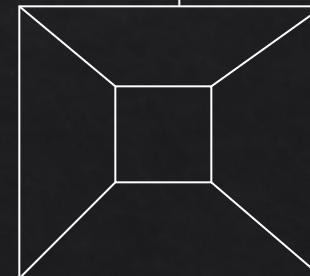
Flying Monster



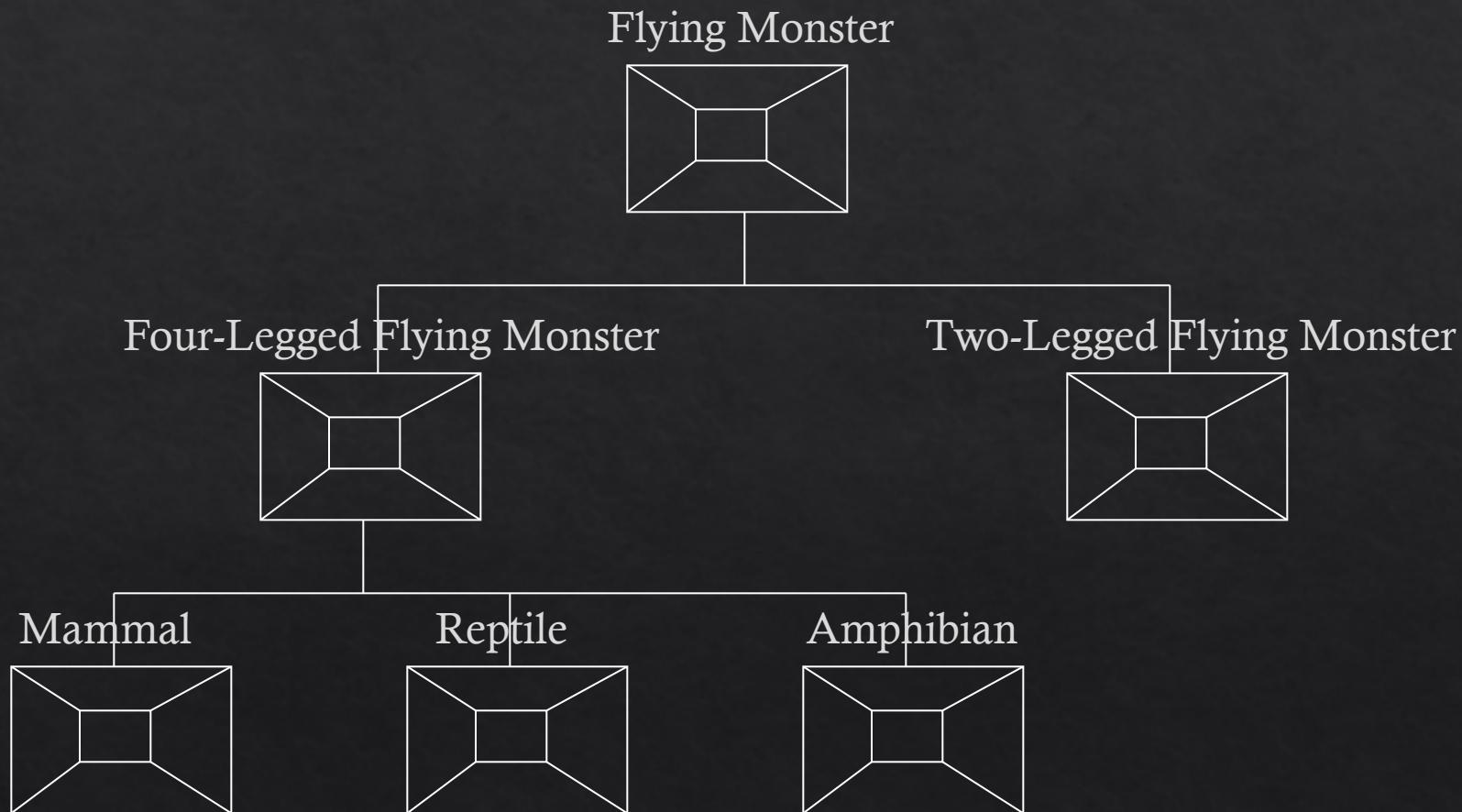
Monster



Swimming Monster



# Flying Monster Class Hierarchy



ପ୍ରକାଶିତ

# Steps for Object-Oriented Programming Design

ମୋଟାଗୁଡ଼ିକ

1. Break down objects to their smallest features
2. Look for commonality between the objects ନାହିଁ
3. Look for differences between the objects
4. Find the largest commonality between all objects common ମୋଟାଗୁଡ଼ିକ ଫାରମ
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