204362 – Object-Oriented Design

Object-Oriented Concept

Adapted for 204362
by Areerat Trongratsameethong

Learning Objectives

- The fundamental concepts of object-orientation, including:
  - Objects and classes
  - Generalization, specialization and inheritance
  - Information hiding and message passing
  - Polymorphism
- Why OOD?

Object-Oriented Concept

An object is:

"An abstraction of something in a problem domain, reflecting the capabilities of the system to keep information about it, interact with it, or both."
Coad and Yourdon (1990)

สะท้อนถึงขีดความสามารถของระบบในการจัดเก็บข้อมูลสารสนเทศของวัตถุ การปฏิสัมพันธ์ระหว่างวัตถุ

Abstraction ในที่นี้หมายถึง ไม่ได้รวมรายละเอียดทุกอย่าง สนใจเฉพาะสิ่งที่สำคัญ

Object คือ ภาพรวมของสิ่งต่างๆที่สำคัญ หรือสิ่งที่สนใจ ที่อยู่ในขอบเขตของปัญหา (ไม่ต้องเอาทุกรายละเอียดของอ็อบเจ็ค เอามาเฉพาะสิ่งที่อยู่ในขอบเขตของปัญหา)

Object-Oriented Concept [2]

Abstraction

- การแสดงถึงวิธีแก้ปัญหา โดยแสดงเฉพาะในขอบเขตที่สนใจ
- ช่วยลดความซับซ้อนและช่วยให้เข้าใจปัญหาได้ง่ายขึ้น
- การแก้ปัญหาในเชิงวิศวกรรมซอฟต์แวร์ สามารถนำเสนอแนวคิดในการแก้ปัญหาผ่านไดอะแกรม (Diagram) ต่างๆ
- ความสัมพันธ์ของสิ่งต่างๆ สามารถนำเสนอในรูปแบบของ แบบจำลองของคลาส (Class Modeling) การสืบทอดคุณสมบัติ (Inheritance) ความสัมพันธ์ระหว่างคลาส (Association) และส่วนประกอบของคลาส (Aggregation/Composition)

http://www.pages.drexel.edu/~ap62/pages/ooconcepts_1.html

Object-Oriented Concept [3]

"Objects have state, behaviour and identity."

Booch (1994)

- ***State***: สถานะของ object ณ เวลาใดเวลาหนึ่ง ซึ่งมีผลต่อพฤติกรรมของอ็อบเจ็ค
- ***Behavior***: พฤติกรรมที่ object สามารถทำได้ สามารถตอบสนองต่อเหตุการณ์ต่างๆ หรือสามารถตอบสนองต่อสิ่งที่มากระตุ้น

  Identity: แต่ละ object มีเอกลักษณ์ หรือลักษณะเฉพาะตัว

Examples of Objects

Object	Identity	Behaviour	State
A person.	'Hussain Pervez.'	Speak, walk, read.	Studying, resting, qualified.
A shirt.	My favourite button white denim shirt.	Shrink, stain, rip.	Pressed, dirty, worn.
A sale.	Sale no #0015, 18/05/05.	Earn loyalty points.	Invoiced, cancelled.
A bottle of ketchup.	*This* bottle of ketchup.	Spill in transit.	Unsold, opened, empty.

Class and Instance

- All **objects** are ***instances* of** some ***class***
- A **Class** is a **description of a set of objects** with similar:
  - features (attributes, operations, links);
  - semantics;
  - constraints (e.g. when and whether an object can be instantiated).

OMG (2009)

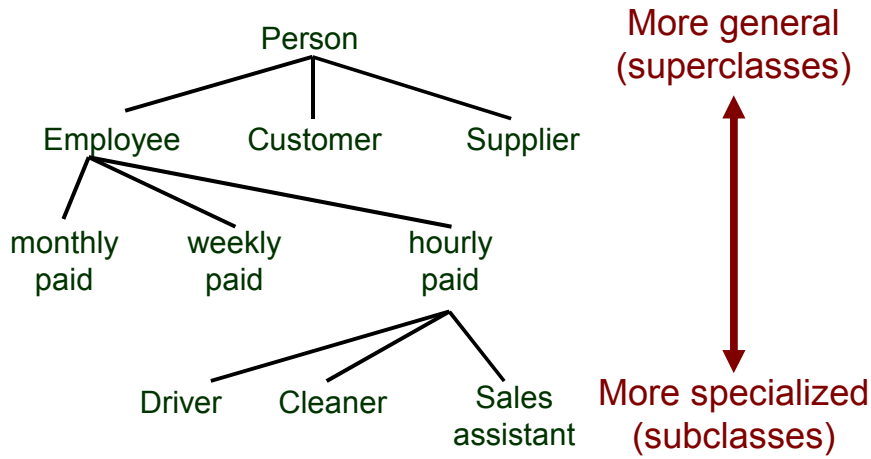
Class and Instance

- An object is an instance of some class
- So, instance = object
  - but also carries connotations (อธิบาย, ความหมายแฝง) of the class to which the object belongs (อ็อบเจ็คเป็นของคลาสไหน)
- Instances of a class are similar in their:
  - ***Structure***: what they *know*, what information they hold, what links they have to other objects
  - ***Behaviour***: what they *can do*

Generalization and Specialization

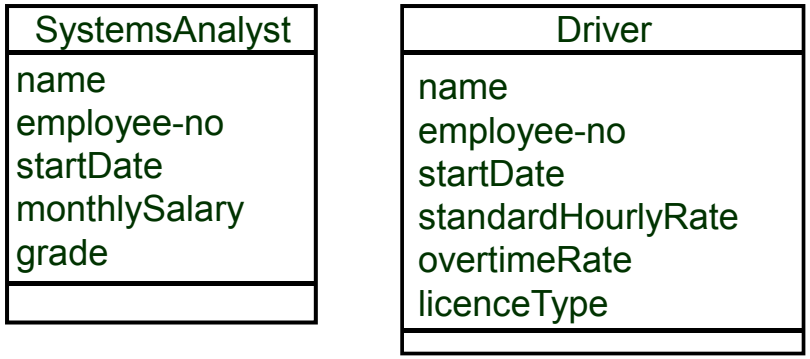
- Classification is hierarchic in nature
- For example, a person may be an employee, a customer, a supplier of a service
- An employee may be paid monthly, weekly or hourly
- An hourly paid employee may be a driver, a cleaner, a sales assistant

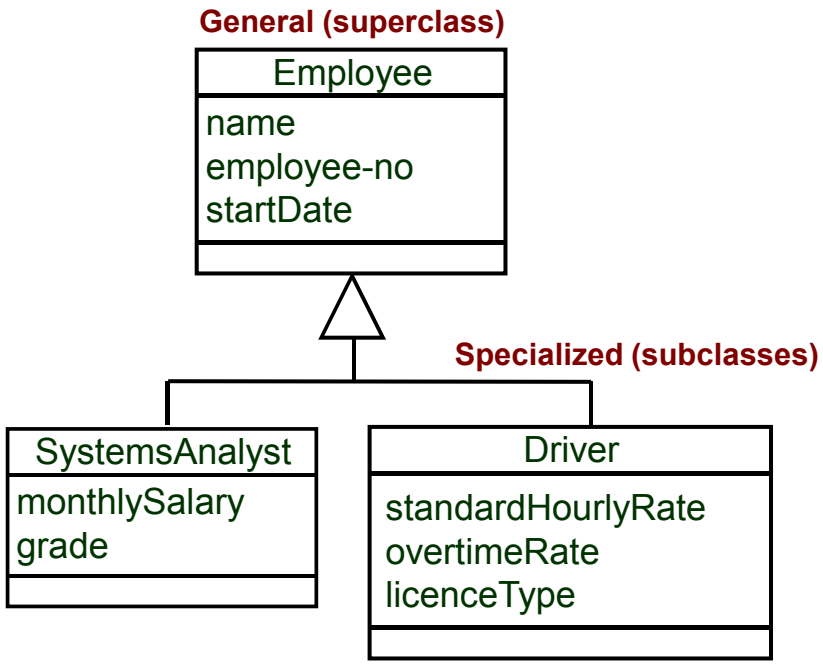
Specialization Hierarchy



Generalization and Specialization

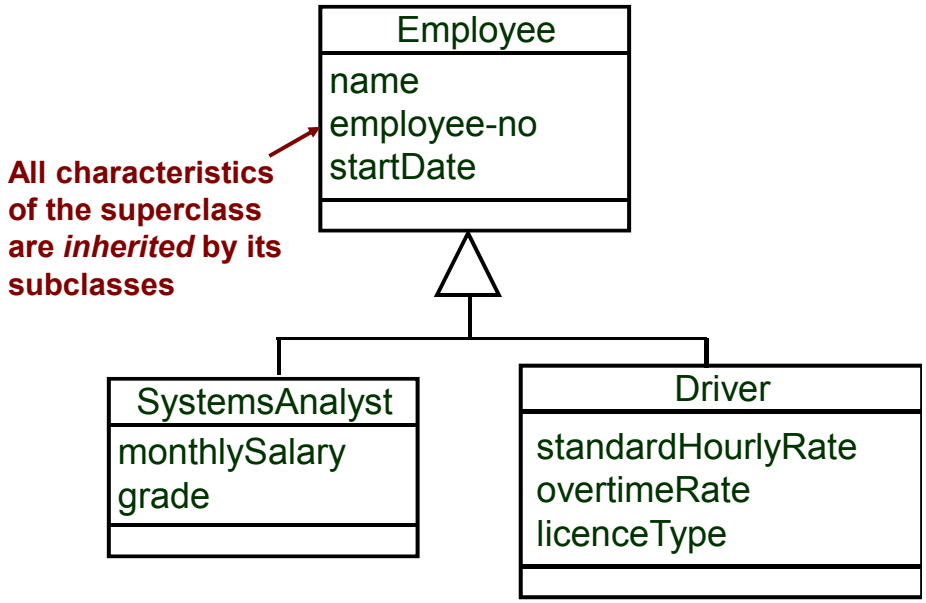
- More general bits of description are *abstracted out* from specialized classes:

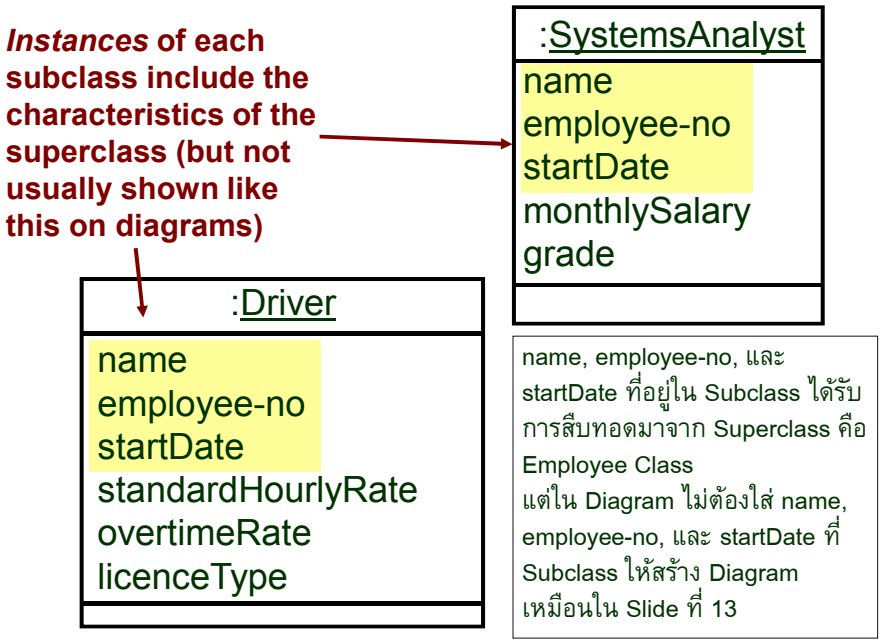




Inheritance

- The *whole* description of a superclass applies to *all* its subclasses, including:
  - Information structure (including associations)
  - Behaviour
- (But actually inheritance is how an O-O programming language *implements* generalization / specialization)





Inheritance [2]

Inheritance

- Another basic principle of OO. There are notions of subclass and superclass. A subclass is derived from a superclass. (An Employee is a Person.)
- The subclass inherits the attributes and behavior of the superclass.
- The subclass can override the behavior of the superclass.
- Inheritance creates a generalization-specialization hierarchy (type hierarchy).
- Inheritance promotes re-use.
- Generalization Rules
  - The ISA Rule:
    - If all members of class A are members of class B and
    - class A inherits all the properties (attributes, relationships, operations and constrains) of class B then
    - class B is a superclass of class A and it is said that A ISA B
  - The ROLE Rule:
    - If class A and class B have different roles of the same class C and
    - class A and class B have at least one different property from class C then
    - class A and class B are subclasses of class C

http://www.pages.drexel.edu/~ap62/pages/ooconcepts_1.html

Message-passing

Encapsulation

- The grouping of related items into one unit.
- One of the basic concepts of OO.
- Attributes and behaviors are encapsulated to create objects.
- OO modeling is close to how we perceive the world. (แบบจำลองเชิงวัตถุ เป็นแบบจำลองที่มีการจำลองในลักษณะเหมือนกับโลกแห่งความเป็นจริง)
- Implementation details are hidden from the outside world. We all know how to use a phone, few of us care how it works.
- The packaging of operations and attributes representing state into an object type so that state is accessible or modifiable only through the objects' interface (การเข้าถึงข้อมูลของอ็อบเจ็คทำผ่าน method())
- Encapsulation lets builders of objects reuse already-existing objects, and if those objects have already been well-tested, much larger and more complex systems can be created.

http://www.pages.drexel.edu/~ap62/pages/ooconcepts_1.html

Message-passing [2]

- Several objects may collaborate (ทำงานร่วมกัน) to fulfil each system action
- "Record CD sale" could involve:
  - A CD stock item object
  - A sales transaction object
  - A sales assistant object
- These objects communicate by sending each other messages

Message-passing [3]

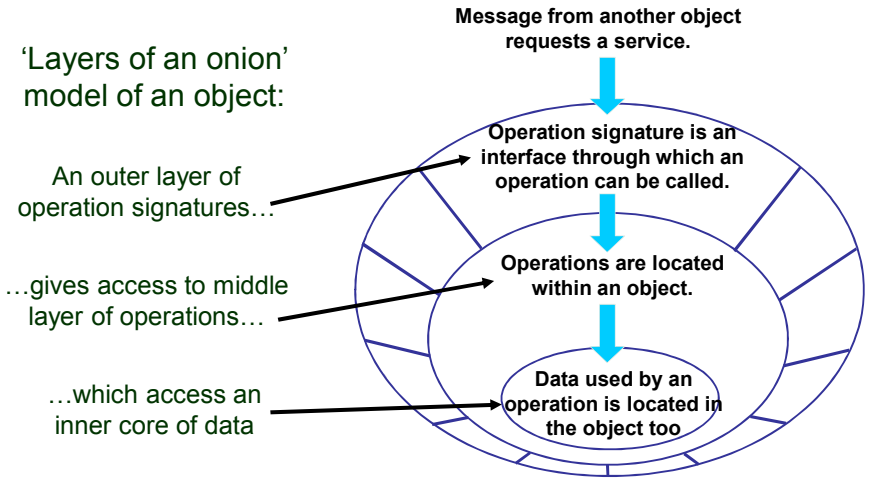
Message Passing

- Objects communicate by sending messages.
- Messages convey (ส่งผ่าน) some form of information.
- An object requests another object to carry out an activity by sending it a message.
- Most messages pass arguments back and forth.
- Meilir Page-Jones defines three types of messages:
  - Informative - send information for the object to update itself.
  - Interrogative - ask an object to reveal some information about itself.
  - Imperative - take some action on itself, or another object
- Grady Booch defines four types of messages:
  - Synchronous - receiving object starts only when it receives a message from a sender, and it is ready.
  - Balking - sending object gives up on the message if the receiving object is not ready to accept it.
  - Timeout - sending object waits only for a certain time period for the receiving object to be ready to accept the message.
  - Asynchronous - sender can send a message to a receiver regardless of whether the receiver is ready to receive it.

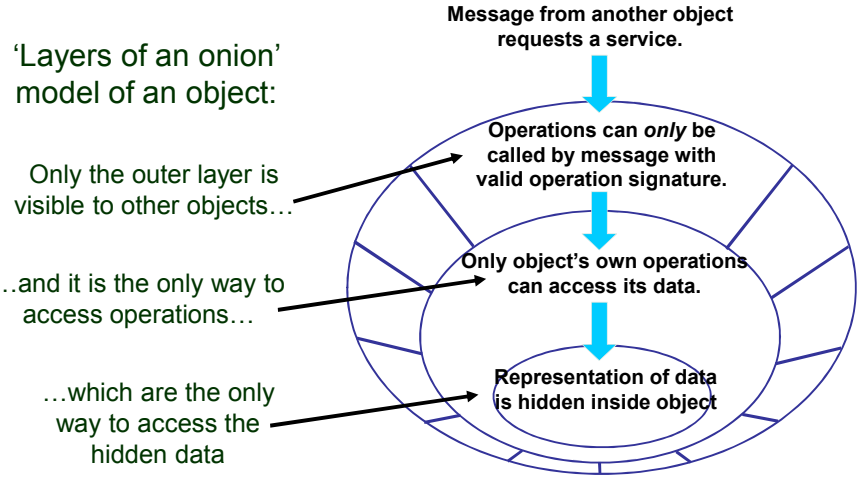
http://www.pages.drexel.edu/~ap62/pages/ooconcepts_1.html

Message-passing and Encapsulation

'Layers of an onion' model of an object:



Information Hiding: a strong design principle



Polymorphism

- Polymorphism allows one message to be sent to objects of different classes
- Sending object need not know what kind of object will receive the message
- Each receiving object knows how to respond appropriately

Example of Polymorphism

```
program Adhoc;

function Add( x, y : Integer ) : Integer;
begin
   Add := x + y
end;

function Add( s, t : String ) : String;
begin
   Add := ( s, t )
end;

begin
   (Add(1, 2));                 // Prints 3
   (Add('Hello, ', 'World!')); // Prints "Hello, World!"
end.
```


Polymorphism [2]

Polymorphism

- Literally means "**many forms**".
- A method can have many different forms of behavior.
- Commonly used between a set of classes that have a common superclass.
- The sender of a message does not have to know the type/class of the receiver.
- A single operation or attribute may be defined upon more than one class and may take on different implementations in each of those classes.
- An attribute may point to different objects at different times.
- Mechanism to allow dynamic substitution of objects within hyrarchy.
- Mechanism to allow sharing services in inheritance hierarchy.

http://www.pages.drexel.edu/~ap62/pages/ooconcepts_1.html

Why OOD?

OOD/OOP is good for:

- Analyzing user requirements: ง่ายต่อการวิเคราะห์ความต้องการของผู้ใช้
  - เราสามารถใช้ Use Case Diagram และ User Interface Mockup (UI Mockup) สำหรับสื่อสารและยืนยันความต้องการของผู้ใช้ได้
- Designing software: การออกแบบซอฟต์แวร์โดยใช้การออกแบบเชิงวัตถุ จะได้ผลลัพธ์คือ Class Diagram พร้อมใช้สำหรับการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)
- Constructing software:
  - Reusability (reusable components): การนำส่วนประกอบของซอฟต์แวร์กลับมาใช้
  - Reliability: มีความน่าเชื่อถือ
  - Robustness: มีความทนทาน
  - Extensibility: ง่ายต่อการต่อยอด
  - Maintainability: ง่ายต่อการบำรุงรักษา

Why OOD? [2]

OOD/OOP is good for:

- According to the Gartner Institute: สถาบัน การ์ตเนอร์ เป็นสถาบันวิจัยระดับโลกที่ให้คำปรึกษาด้านไอที การเงิน และอื่นๆ ได้กล่าวไว้ว่า
  - 74% ของโปรเจ็คไม่ประสบความสำเร็จ เนื่องจากงบประมาณบานปลาย หรือโปรเจ็คเสร็จไม่ตรงตามแผนการดำเนินงาน
  - 28% ที่ล้มเหลวทั้งทางด้านงบประมาณบานปลาย และโปรเจ็คเสร็จไม่ตรงตามแผนการดำเนินงาน
  - ในทุกๆปี จะมีไอทีโปรเจ็คมูลค่า เจ็ดหมื่นห้าพันล้านเหรียญสหรัฐ ที่ไม่ประสบความสำเร็จ

Advantages of O-O

- Can save effort
  - Reuse of generalized components cuts work, cost and time
- Can improve software quality
  - Encapsulation increases modularity
  - Sub-systems less coupled to each other
  - Better translations between analysis and design models and working code

Summary

In this lecture you have learned about:

- The fundamental concepts of O-O
  - Object, class, instance
  - Generalization and specialization
  - Message-passing and polymorphism
- Some of the advantages and justifications of O-O

