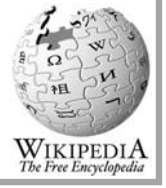
Fundamentals in Relation to
Object-Orientation

Prof. T.H. Tse
Department of Computer Science
Email: thtse@cs.hku.hk
Web: hku.hk/thtse .

Object-Oriented Approaches

- ☺ Object-oriented analysis
- ☺ Object-oriented design
- ☺ Object-oriented programming
- ☹ Object-oriented database ***??***

Allow OO programmers to develop the product, ***store*** them as objects, and ... ***modify*** existing objects ☹ .



Advantages of Object-Oriented Analysis and Design

(1) *User-friendly*

- Straightforward guidelines make it easy for novices to learn the trade
- Partition complex systems into manageable units
- Simple graphics
- User-friendly CASE tools .

Advantages of Object-Oriented Analysis and Design

(2) *Stand on shoulders of other giants*

- Direct extension from object-oriented programming
- Simple transition from structured analysis and entity modelling .

Advantages of Object-Oriented Analysis and Design

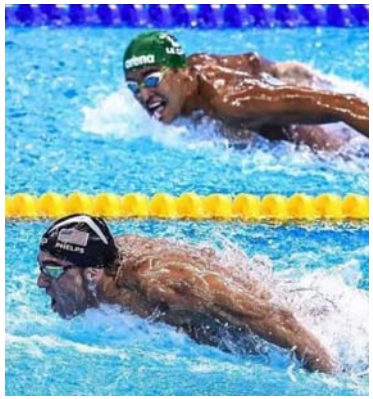
(3) *Object-oriented*

- Focus more on data structures than on target functions
- More stable base for development process
- Use objects as a single unifying software concept throughout development process
- All other concepts based on objects
  - ***Examples:*** attributes, relationships, methods, states, events .

Advantages of Object-Oriented Analysis and Design

(3) *Object-oriented*

- Encapsulated objects are protected from unexpected ripple effects
- Inheritance encourages re-use .



Advantages of Object-Oriented Analysis and Design

(4) *More comprehensive*

- Entity modelling concentrates on static relationships
- Structured analysis concentrates on behaviour
- Static relationship and dynamic behaviour in OO

(5) *Easier to plan*

- No need to visualize the system as a single entity
- No need to expect an "insight", as in a top-down approach .

Advantages of Object-Oriented Analysis and Design

(6) *More reusable*

- Previous methodologies assumed that we replace the current system (or a substantial portion) by a new system
- In structured design, reuse of the same module for different purposes was explicitly discouraged .

Advantages of Object-Oriented Analysis and Design

(7) *Seamless development process*

Advantages of Object-Oriented Analysis and Design

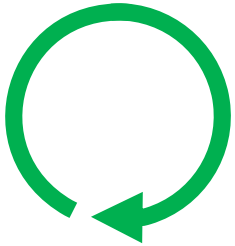
(7) *Seamless development process*

- *Objects* are continuously used and extended throughout development cycle
- Development phases are less distinct
- No discontinuities
  - The notation in one phase is ***not*** replaced by a different notation in another phase
- Reduce information loss in design and implementation .

Advantages of Object-Oriented Analysis and Design

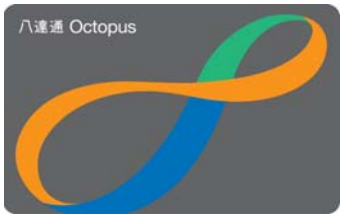
(8) *Iterational rather than sequential*

- Seamless object-oriented development refines product to finer details
  - Each iteration adds or clarifies features rather than modifying completed work
  - Reduces inconsistencies or errors .



Objects

- Is this an object?



Objects

- Is this an object?

Objects

- An ***object*** is an individual, identifiable item, unit, or entity, either physical or conceptual, with a well-defined role
  - ***Examples:*** A student
  - A bank account .

Objects

Warning

Some authors define objects this way:

- An ***object*** is something that is or is capable of being seen, touched, or otherwise sensed, and about which users store data and associate behaviour [Whitten et al.] ***??***
- 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; Bennett et al.] ***??*** .

Classes

A class is a set of objects that share

- A common ***structure*** — properties
- Similar ***attributes*** — properties
- Common ***behaviour*** — operations
- Similar ***relationships*** with other objects
- Common ***semantics*** — meanings

A single object is simply an instance of a class .

Classes and Objects

Examples

Class ***Objects***

Attributes

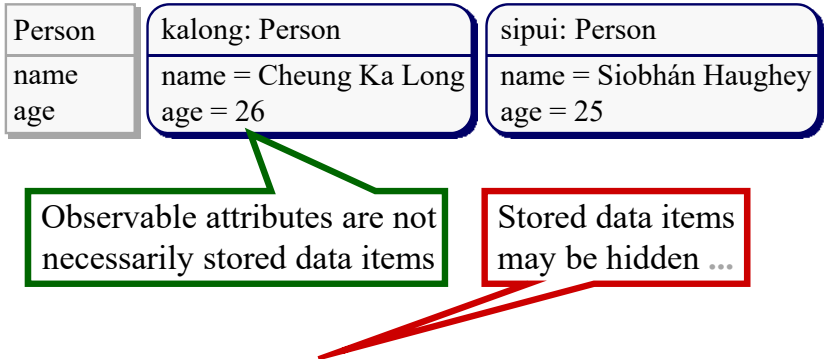
- An ***attribute*** is an ***observable*** property of objects in a class
- Each object in the class has a value for each attribute
- The value may or may not be unique for an individual object

- ***Attributes*** are the data that represent characteristics of interest about an object ?? .

Attributes and Values

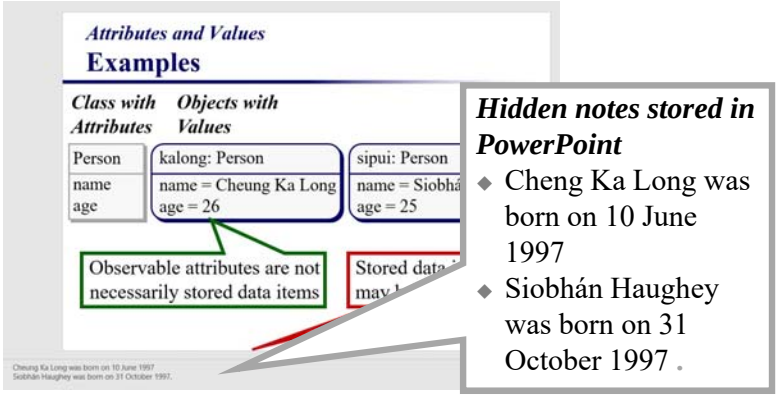
Examples

Class with Attributes ***Objects with Values***



Attributes and Values

Examples



Behaviour

- ***Behaviour*** refers to those things that the object can do
- In the object-oriented paradigm, behaviour is modelled by operations .

Operations

- An ***operation*** is a function or transformation that may be applied to or by an object in a class
  - *Examples:* debit, credit
- All objects in a class share the same operations
- Each operation has the current object as an implicit argument .

Operations

Method

- A ***method*** is an operation that may change the value of some attribute of an object

Query

- A ***query*** is an operation that only reports the value of some attribute of an object without changing any attribute .

Operations

Examples

- Is “debit” a method or query?
- Is “get balance” a method or query?
- Suppose the system computes the balance using “`for (int i = 0; i <= 10; i++)`”, and changes `i` from `0` to `10` after processing
  - Is “get balance” a method or query?
- Suppose the bank charges $5 when you ask for the balance
  - Is “get balance” a method or query? .

Operations

Examples

Account
balance
debit credit

Query "get balance" by default .

Customer
name age address
changeName changeAge changeAddress

More on Methods and Queries

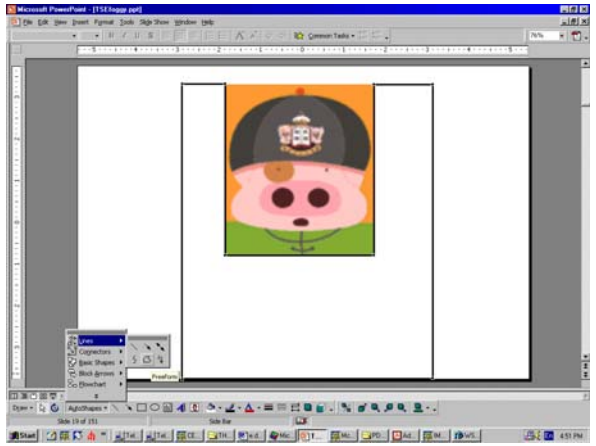
- Is it really necessary to distinguish between methods and queries?
- Methods describe the ***dynamic behaviour*** of objects
- Queries describe their ***static attributes***
- Is it really necessary to distinguish between dynamic behaviour and static attributes? .

Methods and Queries

Software Testing Example

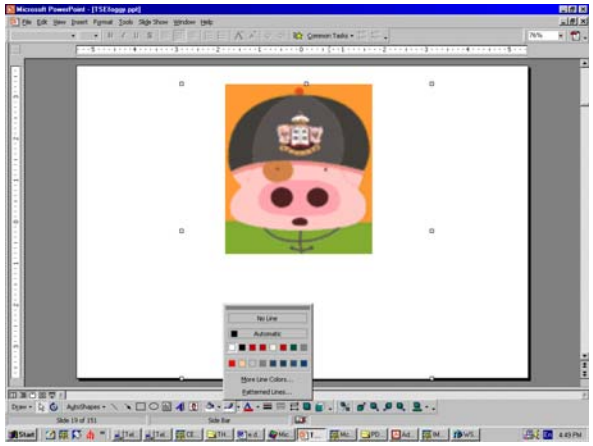
Methods and Queries

Software Testing Example



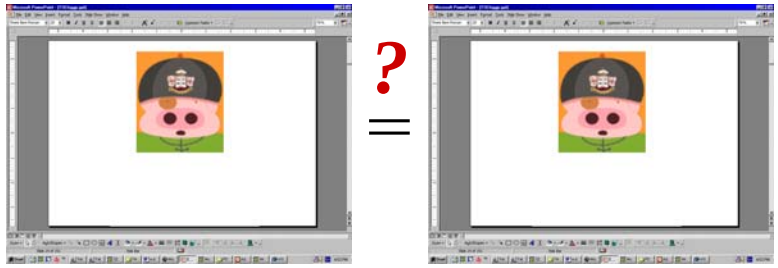
Methods and Queries

Software Testing Example



Software Testing Example

Expected Object = Actual Object ?



- What exactly is "="? .

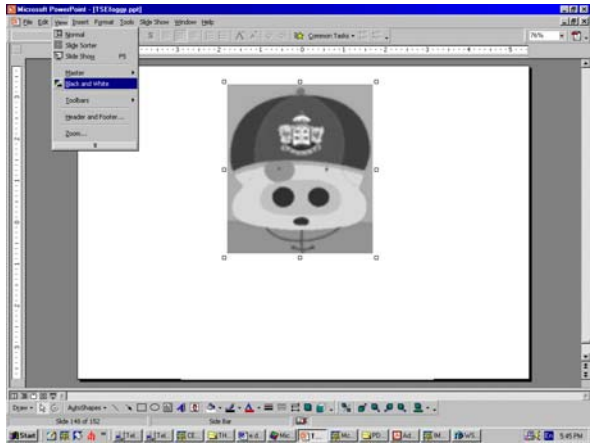
Software Testing Example

Attributive Equivalence

- Two objects will be ***attributively equivalent*** if they have the exactly the same ***static attributes***
- Simple to test
- ***But*** the definition is too weak to be useful
- Why? …

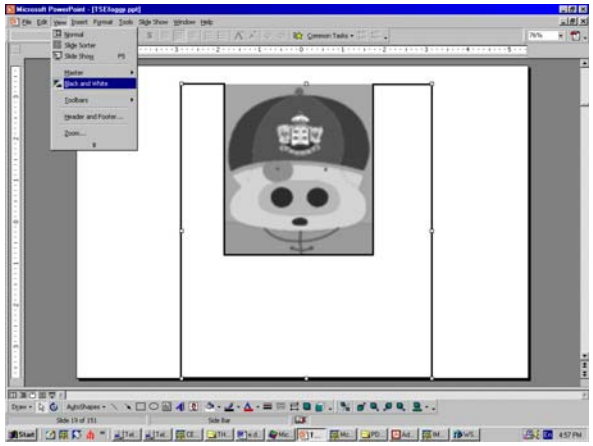
Software Testing Example

Behaviour of Expected Object



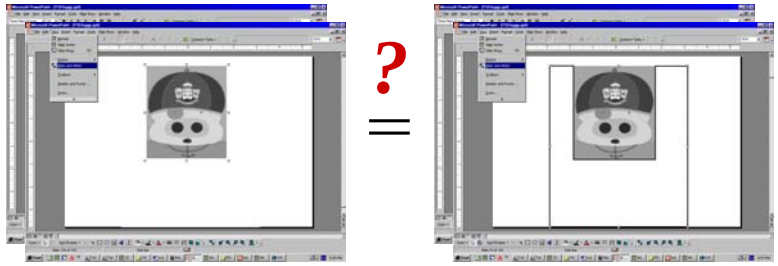
Software Testing Example

Behaviour of Actual Object



Software Testing Example

Expected Object = Actual Object ?



- Attributively equivalent
- Observationally non-equivalent .

Software Testing Example

Observational Equivalence

- Two objects will be ***observationally equivalent*** if they have the exactly the same ***static attributes and dynamic behaviour***
- Very useful object-oriented concept
- *But very very very very very very very very very very very very very very very* difficult to test
- Need to check an infinite combination of possible behaviour for every test case
- Solved by Chen and Tse (2001 and 2013) .

Object-Oriented Concepts

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Persistence and states .

Abstraction

- Models the most important aspects while suppressing or ignoring less important details
- Manage complexity by concentrating on essential characteristics
- Domain and perspective specific
  - What is important in one context may not be the case in another .

Encapsulation

"Just do it. Don't tell me how."

- Separate the external aspects of an object from the internal, implementation details
- Summarize both the ***structure*** and ***behaviour*** of an abstraction
- Define the ***contractual interface*** between abstraction and implementation

Also called ***information hiding*** .

Encapsulation Example

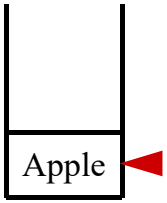
- push "Apple"
- push "Banana"
- pop

What is the resulting stack? .

Encapsulation Example

According to the specification:

- push "Apple"

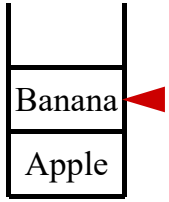


depth = 1
top = Apple

Encapsulation Example

According to the specification:

- push “Apple”
- push “Banana”

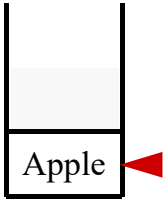


depth = 2
top = Banana

Encapsulation Example

According to the specification:

- push “Apple”
- push “Banana”
- pop .

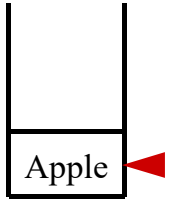


depth = 1
top = Apple

Encapsulation Example

According to the implementation:

- push “Apple”

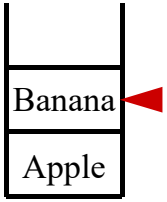


depth = 1
top = Apple

Encapsulation Example

According to the implementation:

- push “Apple”
- push “Banana”



depth = 2
top = Banana

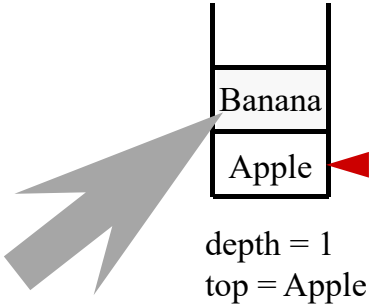
Object-Oriented Concepts

Encapsulation Example

According to the implementation:

- push “Apple”
- push “Banana”
- pop

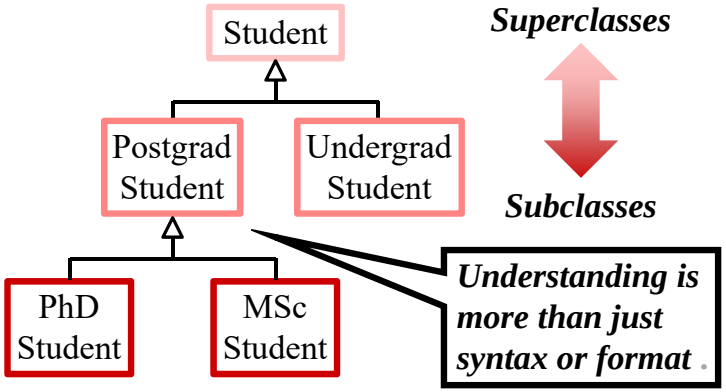
Is the implementation correct? .



depth = 1
top = Apple

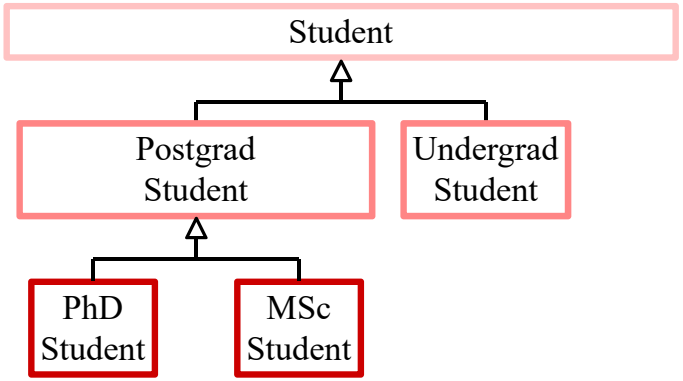
Object-Oriented Concepts

Inheritance Example



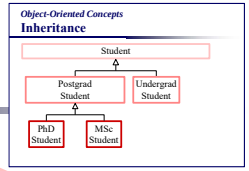
Object-Oriented Concepts

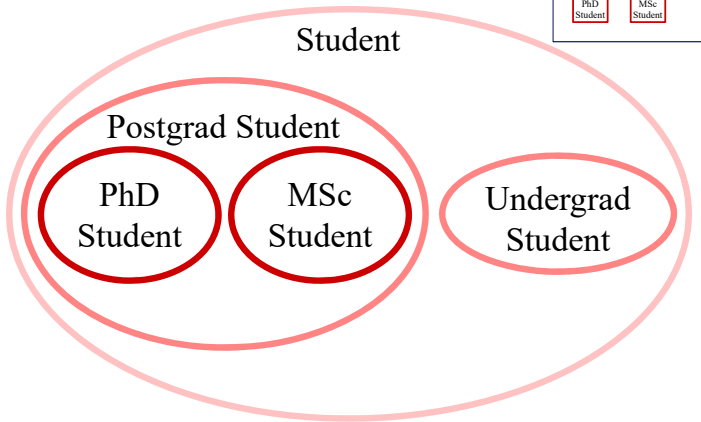
Inheritance Example



Object-Oriented Concepts

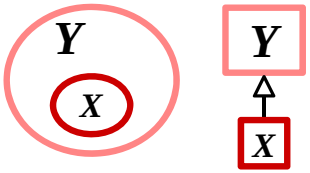
Inheritance Example





Object-Oriented Concepts

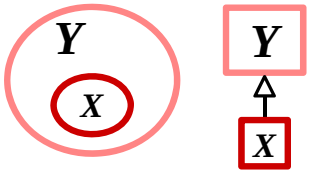
Inheritance



- A class ***X*** ***inherits*** from another class ***Y*** if *X* is a ***subclass*** (smaller class) of *Y* having special properties
- *Y* is called the *superclass* ...

Object-Oriented Concepts

Inheritance



- *X* has access to all operations defined on *Y* as well as special operations of its own
- Inheritance is also known as *generalization*-***specialization*** relationships .

Object-Oriented Concepts

Polymorphism

- Traditional *typed* languages, such as Pascal, assume that a parameter of a function or procedure has a unique type
- Said to be monomorphic
- A parameter in polymorphic languages may have more than one type .

Object-Oriented Concepts

Polymorphism

- 3 kinds of polymorphisms:
  - ***Subtype polymorphism*** will work correctly if passed an argument that is a subtype of the object it expects
    - *Example:* `sqrt(4.0)` and `sqrt(4)`
  - ***Parametric polymorphism*** is obtained when a function works uniformly on a range of types, which normally exhibit a common structure
    - *Example:* `gpa(undergrad)` and `gpa(postgrad)`
  - ***Ad-hoc polymorphism*** is obtained when a function works, or appears to work, on several different types
    - *Example:* `50 - 50` and `fifty-fifty` .

Polymorphism

- In terms of implementation:
  - A ***subtype*** or ***parametric polymorphic function*** will execute the same code for arguments of any admissible type
  - An ***ad-hoc polymorphic function*** may execute different code for each type of argument .

Polymorphism

- ***Overloading*** is another name for ad-hoc polymorphism
- ***Overriding*** supersedes subtype polymorphism
  - *Example:* `debit` in `Account` and `debit` in `Minimum Balance Account` .

Persistence

- Unlike transient data item, an object must be persistent and have a ***life history***
- Created at some point in time, undergoes changes in ***states***, and only destroyed at the request of the user or another object
- Must have an ***identity*** .

States

- A state is a collection of the attribute values and links of an object
  - ***Example:*** Airplane Mode is "on", which represents
    - Wi-Fi is "off" *and*
    - Bluetooth is "off"

- A state specifies the response of the object to input events
- Corresponds to time interval between 2 events received by an object .