COMP2009
Software Engineering

Object-Oriented Concepts

Overview

- UML, the Unified Modelling Language is Object-Oriented.
- This slide set looks at:
  - Models and modelling
  - Some history and background
  - Object-oriented ideas and concepts

- P.S. Object-*Oriented,* NOT Object-*Orientated*

What is a Model?

- A *selective* representation of a system.
  - Emphasises essential details.
  - Omits irrelevant details.
  - The purpose and level of detail of the model determines what is emphasised and what is omitted.
- An abstraction over reality.
- Example, consider a record of an employee in a payroll database. What information is needed?
  - Name, age, height, address, weight, shoe size, phone number, department, salary, savings, pension, favourite colour, ?????

- What is emphasised or omitted in these models?
  - Wind tunnel model on aeroplane.
  - Scale model of a building.
  - A map.
  - London Underground route planner.
  - Organisation chart.

Abstraction

- *Abstraction* is the process of identifying and representing *essential* detail.
  - Not representing, or eliminating, detail that is not essential.
  - Removing detail from a model.
  - Making a model more abstract
- *Reification* and *refinement* are the opposite of abstraction.
  - Adding detail to a model.
  - Making a model more concrete.

Modelling

- *Modelling* a system means *representing* its main characteristics, states and behaviour using a *notation*.

- Example: You can model a Library System using Java.
  - A low-level, detailed, monolithic model.

- Example: You can model a Library System using UML.
  - A comprehensive, higher-level model expressed in multiple views.

Purpose of a Model

- A *model* is a description from which detail has been removed in a systematic manner and for a particular purpose.
  - A simplification of reality intended to promote understanding.
  - Enable communication between all interested parties.
- Models are the most important engineering tool, allowing us to understand and analyse large and complex problems.
  - Visualisation
  - Verification
- Models are built in a *language* appropriate to the expression and analysis of properties of particular interest.

Model Building

- Building a system can be seen as a process of reification.
  - Moving from a very abstract statement of what is wanted to a concrete implementation.
- In doing this, you move through a sequence of intermediate descriptions which become more and more concrete.
  - These intermediate descriptions are models.
- The process of building a system can thus be seen as the process of building a series of progressively more detailed models.

Tools for Dealing with the Complexity of Software

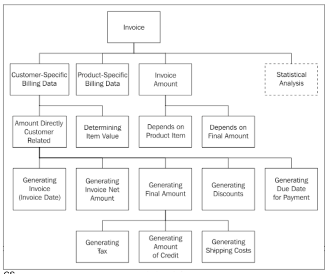
- **Modularity**
  - A well-defined collection of parts with well-delimited interactions.
- **Encapsulation**
  - Confines the impact of changes made to a module.
  - Clear separation of interface from implementation.
  - A client of the module knows no more than what is in the interface.
- **Abstraction**
  - Allows focus on essential details, ignoring non-essential details.
- **Information hiding**
  - A client of the module needs to know no more than what is in the interface.
- **Hierarchical Decomposition**
  - Decomposition of complex problem into smaller independently solvable pieces.
  - Separation of higher and lower levels of abstraction.

Decomposition

- Complex systems need to be *decomposed* into smaller, less complex parts to be manageable.
- The parts, or components, must join together correctly via well defined interfaces.
- The system *architecture* defines the overall structure and how the components are composed together.

The Old: Hierarchical Input Processing Output (HIPO)

- Hierarchy of functions and sub-functions.
- Input-Process-Output pattern for each element.

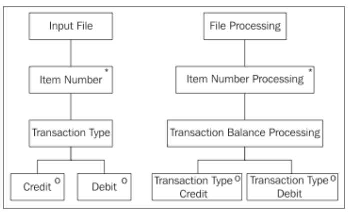


- Structured programming.
- Structure charts.
- Data defined separately.
- Data-Flow Diagrams (DFDs)
- COBOL

UML2 in Action, Grässle et al.

Data-Structure Oriented

- e.g., Jackson Method
- Jackson diagram shows:
  - structure of data set on left.
  - derived program structure on right

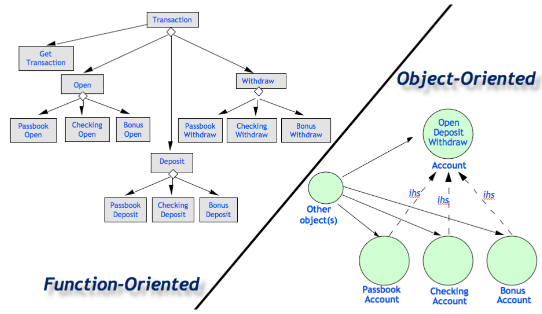


UML2 in Action, Grässle et al.

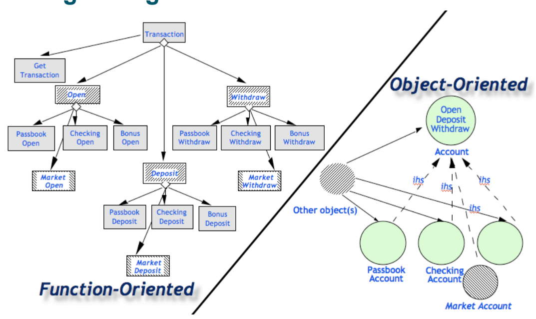
The New: Object-Oriented Methods

- Classes, objects.
  - Object combines data and function.
  - Class defines structure and behaviour of instance objects.
  - More in a later lecture.
- More direct representation of problem domain.
- Provides continuity of representation between analysis, design and implementation.
- Facilitates more effective reuse of analysis and design.
- Better able to cope with change compared to function-oriented approaches.

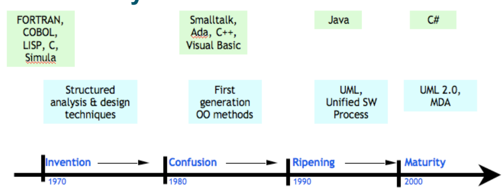
Contrasting Decomposition Strategies



Making Changes



A Brief History of OO



- *Class*, *inheritance* from Simula (1964), Simula-67.
- Term *object-oriented* first applied to Smalltalk (1970's), Smalltalk-80.
- *Structural feature, functional abstraction* from LISP.
- *Frames*, *actors* from artificial intelligence.
- Extension to *libraries, databases, GUIs, architecture, deployment*, …

Main OO Concepts

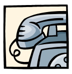
- Object
  - State
  - Identity
  - Behaviour
- Message
- Method
- Interface
- Class
- Inheritance
- Polymorphism

Objects

- "*An abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both.*" [Coad & Yourdon 1991]
- "*An entity able to save a state (information) & which offers a number of operations (behaviour) to either examine or affect this state.*" [Jacobson 1992]
- "*… a thing which has behavior, state & identity*" [Booch 1991]

Objects (2)

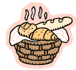
- Something that represents an *atomic entity*.
- Something you can uniquely identify (*identity*).
- Something that *encapsulates* data as its state.
- Something you can send messages to, causing it to *respond/behave* in some way.
- Something whose behaviour depends on its internal *state* (which may change).


object ***has*** state + identity + behaviour

Example Objects

- Passive objects (having no behaviour)
  - One loaf of bread
  - One packet of herbal tea
  - Invoice 63501 sent to A Farm, Malvern
- Active objects (having behaviour)
  - Lorry "M235 BCM"; Van "N683 CNM"
  - Fax machine in Richard Green's office
- Human agents
  - Richard Green; David Brown (Executive)
- Structure objects
  - Marketing Department




State

- All the data the object currently encapsulates.
- Data is defined in terms of named *attributes*.
- The values of some attributes are fixed (immutable).
- The values of some attributes can change (mutable).
- The values of the attributes are the state of the object.

student_id; registered_courses; weight; date_of_birth

Identity

- Attribute values may change, so they don't uniquely characterise an object over time.
- Identity makes it possible to distinguish any object in an unambiguous way, independent from its state.
- Characterises the object to give persistence.
- Like a primary key in a database, or a unique handle.

phone number student_id

Behaviour

- *Operations* enable an object to act and react.
- Objects can receive certain *messages* and act upon them.
- The set of messages the object can understand is generally fixed and defined in its *interface*.
- Reaction to messages often depend on current attribute values (i.e. state) and may even change the state.

on *flickSwitch* off

Messages

- *Messages* are sent to objects to trigger behaviour.
- Structure: Operation selector and optional arguments.
- Arguments are values being passed to the object.

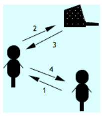
Identity: myClock
Attribute: currentTime
Message: resetTime(newTime)
Message: tellTime

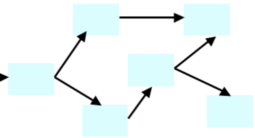
- An object may respond to a message by sending a message to another object.

DEPARTMENT OF COMPUTER SCIENCE — UCL

Message Passing

- Objects therefore interact by passing messages to one another (i.e. object interaction).
- These messages make requests to objects to perform operations or services.





DEPARTMENT OF COMPUTER SCIENCE — UCL

Interfaces

- An object can have *public* and *private* interfaces.
- An object's public interface defines the messages it will accept from other objects.
- An object's private interface can only be used by the object itself or certain privileged objects.
- Structure: Operation selector, required arguments and what will be returned.

Public: resetTime(newTime: Time)
Public: tellTime() : Time
Private: time : Time

DEPARTMENT OF COMPUTER SCIENCE — UCL

Method

- A description that determines how an object reacts to a message.
  - Example: a piece of code implementing an operation.
- Methods are called in response to messages sent to objects.
- The operation performed is determined by the the object's *class* and the value of the object's attributes.
  - So the method has access to the object's state.
- Matching a message name to a method is called *binding*.
  - Message names are *dynamically bound* to methods.

DEPARTMENT OF COMPUTER SCIENCE — UCL

Class

- Objects can have a lot in common, leading to much duplication of descriptions.
- A class describes a set of objects with equivalent roles in the system of interest, acting as a template or blueprint.
- Classes are abstractions representing groups of objects with the same behaviour and information structure.
- Every object belongs to a class (i.e. is an instance of the class), and a class may have many instance objects.

timepiece

student

DEPARTMENT OF COMPUTER SCIENCE — UCL

Example Classes

Product

Attributes: Price, Weight, ...
Relations: Order, Delivery, Supplier
Operations: Is Created, Is Sold, Is Delivered

Employee

Attributes: Age, Weight, Sex
Relations: Role
Operations: Joins Company, Leaves Company, Reviewed, Promoted, Demoted

Truck

Attributes: Registration No., Weight, Fuel
Relations: Driver, Trailer
Operations:
Deliveries Goods
  Repeat Until End
    Go To Destination
    Unload Goods
  EndRepeat
Refuel....

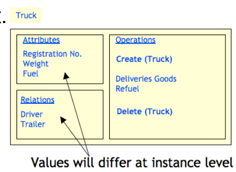
Department

Attributes: Personnel Number, Location, ...
Relations: Director, Marketing Team, ...
Operations: Hire new staff, Dismiss new staff

DEPARTMENT OF COMPUTER SCIENCE — UCL

Instance and Instantiation

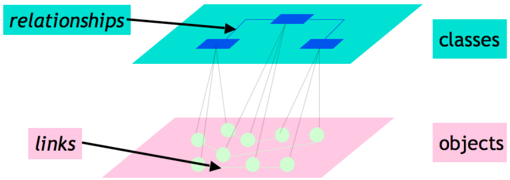
- **Instantiation**: The process of creating a new object belonging to a class.
- **Instance**: The resulting object.
- Note that this object is a new 'thing' with its own identity.



- The state of the new object must be initialised.
- The class of the object determines its interface.
  - It will behave consistently with all other objects of its class.

Classes v. Objects

- Classes *define* the structure and behaviour of a system.
- Objects represent the *actual* system.

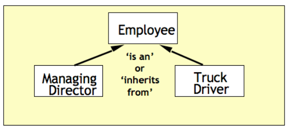


Relationships

- **Association**: A relationship between two classes.
  - Static: A Lecturer *teaches* a Student.
  - Dynamic: Zaphod *teaches* 20 Students (Smith, Nataraj, etc.)
- **Composition/Aggregation**: Stronger forms of associations, representing part/whole relationships.
  - Aggregation: weak ownership.
    - A Student is part of a Course.
    - But the Student can be a part of many Courses.
  - Composition: strong ownership.
    - A Tire is part of a Car.
    - And the Tire is part of exactly one Car.
- Inheritance: A relationship specifying that a class is an extension of another class (e.g. a car is a *kind-of* vehicle).

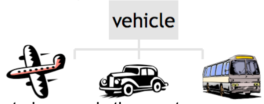
Inheritance

- Inheritance is a relationship between different classes that share common characteristics.
- '*If class B inherits class A, then both the operations and information structure in class A will become part of class B*' [Jacobsen 1992].
  - In general, there are *many* possible ways of doing this.
- Results in simpler classes at higher levels of abstraction.

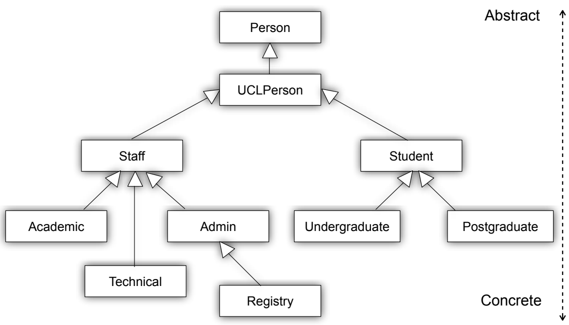


Generalisation, Specialisation and Polymorphism

- A *superclass* may be inherited by a subclass.
- The *subclass* gains all the properties of the superclass and can add more.
- The superclass is a *generalisation*.
- The subclass is a *specialisation*.
- Leads to polymorphism.
  - Object sending a message needs to know only the most general class of the receiving object relevant to the operation requested.
  - Example: *Re-fuel* a Vehicle.
    - Can re-fuel an airplane, a car, or a bus
  - Example: *Fly* an Airplane
    - Can fly a turboprop, a jet, or a glider
    - But can't fly *all* Vehicles.



Inheritance Hierarchy



Generalisation & Specialisation

- A superclass is a generalisation.
  - Shape defines the abstract properties of shapes in general.
  - Number defines the common behaviour of numbers.
  - Person defines common attributes (name, date of birth, etc.)
- A subclass is a specialisation.
  - Square represents a specific kind of concrete shape.
  - Integer, Double define specific kinds of number representation.
  - Undergraduate defines specific attributes (e.g., year, registered modules).

Abstract v. Concrete

- Abstract classes provide a partial or abstract description.
  - Not enough to create instance objects.
  - Define a common set of public methods that all subclass objects must have - common interface.
  - Define a common set of variables/methods can be shared via inheritance.
    - Do not need to be duplicated in all subclasses.
- Concrete classes provide a complete description.
  - Inherited + new attributes/methods.
  - Inherit shared interface.
  - Can be used to create instance objects.

Summary

- Object, Class
- State, Identity
- Message, Method, Binding
- Instance, Instantiation
- Inheritance, Polymorphism