LECTURE 5

OBJECT-ORIENTED THINKING

Ian Stavness, CMPT 270, 2017-18 Term 1

Objects in the world

- Properties
  - Physical
  - State
    - eg. car running, car in gear
- Behaviour
  - eg. car drive down road, turn corner, defrost
  - Note that behaviour is dependent upon the state
  - Key : interface
    - need to know what it can do and how to control it
    - need NOT know how the behaviour is implemented

Object-Oriented programming

- Incorporate the concept of an object into programming
- For large-scale systems:
  - Maintainability
  - Extensibility
- Handle complexity by decomposition
  - Divide and conquer

Evolution of Software Systems and Modeling

- 1950s and 60s
  - Focus on algorithms
    - design, efficiency
  - Decomposition based on control flow
- 1970s and 80s
  - Focus on data
  - Data-oriented models
  - Decomposition based on data organization and flow

Evolution of Software Systems and Modeling

- 1990s, 2000s
  - Focus on objects
  - Object-oriented models
  - Decomposition based on objects/classes and subsystems and relationships among them
- 2010s
  - Focus on mobile / cloud
  - Speed, memory footprint
  - Data issues

Software

- A main objective of software
  - To assist in solving problems of the real world
- Two components of software
  - A model of the real world
  - Algorithms to manipulate the model

Software Models

- A model is an abstraction
  - Precise
  - Simplifies
    - ignores irrelevant aspects
    - retains relevant concepts and detail

    Note that what is relevant is dependent upon the application

  Key: The model must accurately reflect the real world from the perspective of the application.

Software Models

- Results from the model are interpreted in the real world to solve problems.
- Object-oriented programming languages are convenient to model the world, partly because objects in the programming language can be used to model objects in the real world.

Real world object

- property
- already exists (manufactured or natural creation)
- behaviour

Software object

- field/attribute
  - stores values and entities
  - like fields of a struct in C/C++
- constructor
  - special type of routine to return an instance of the current type

Software object

- methods/operations (called functions in C/C++)
  - observers/accessors/functions
    - access or calculate values based on the information within the object or accessible from the object
    - don’t change the state
  - mutators/modifiers/procedures
    - change the state of the object or the state of something
    - don’t return anything
  - hybrid
    - both access and mutate
    - extensively used in Java and C
    - discouraged by modern software approaches

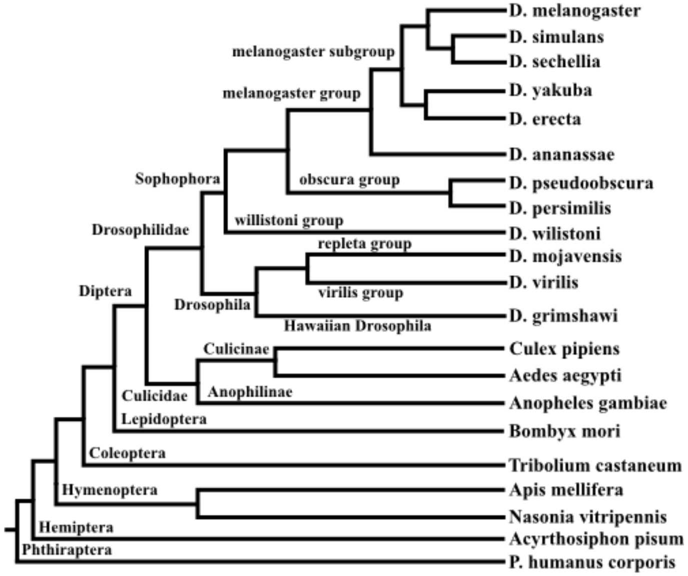
Software object

- An object has
  - Data
  - Operations
- At execution time, program/system basically consists of
  - a collection of objects
  - a driver routine to get it started
- Where should data be placed?
  - with the object to which it pertains ("follow the model")
- Where should operations be placed?
  - with the object that has the behaviour corresponding to the operation
  - with the data that they use

Ex: Tree

Tree classification

- ❑ Family
- ❑ Genus
- ❑ Species
- ❑ Variety



- ❑ Type of apple

Ian Stavness, CMPT 270, 2017-18 Term 1

Tree composition

- Root [ ]
- Trunk
- Branch [ ]
  - Leaf [ ]
  - Apple [ ]

- Trunk properties: diameter, height, bark color

Tree state

- Age (sapling, new, old, dead)
- Season (bud, bloom, fruit, bare)
- Location (yard, park, forest)

Object classifications

- Entity
- Container
- Interface
- Control

Entity Object

- Entity objects that model entities of the external world, for example
  - Concrete objects
    - eg. people, buildings, equipment, things
  - Conceptual objects
    - eg. organizations, agreements, abstractions (maps, plan, blueprint)
  - Event and state-change objects
    - anything with more than one value that needs to be noted or recorded
    - eg. purchase, sale, withdrawal, birth, death, termination

Container Object

- Contain a number of entities
- Implemented via data structures
- Key properties:
  - what type of entities are stored
  - what container operations are needed

    In particular, how are the entities in the container accessed

Interface Objects

- Handle communication between the system and external entities
- May need to be changed when the external entities change
- e.g. GUI interfaces, printer interfaces

Control objects

- Control the sequence of execution of the program
  - eg. driver, main program
  - we will see many others

Types

- Objects are organized into categories (types)
  - Modeling: objects with similar enough characteristics are given the same type
    - How similar they must be to be given the same type depends upon the application.
  - Programming:
    - Objects to have the same type are created to have the same **class**
    - The class defines the fields and operations
    - Hence, all objects of the same type have the same fields and operations.

Ex: Student registration system

- What entity types/classes are needed?
  - fields
  - accessors
  - mutators
  - constructors
- What containers are needed?
- What does each one store?

Ex: Student registration system

- How are the items of the container accessed?
  - By their value? By key? By index? Sequentially one by one? In what order, or does the order matter
- Think in terms of the needs of the application, don’t think data structures yet.
  - accessors
  - mutators

E.x. (for Assign. 1) Pet Kennel

- Consider a pet kennel where people can leave their dog or cat for a few days while the people are away.
- The kennel has a fixed number of pens, where each animal is kept in a distinct pen.
- The objective is to keep track of which pet is in each pen, and owner for each pet.
- Also, information is kept for each owner, for example their name, address and pets that they have.
- The information for an owner is retained even if the owner does not presently have a pet in the kennel.

- Pet
  - name
  - type (dog or cat)
  - size
  - breed
  - colour
  - pen number
  - Owner
  - accessor methods:
    - IsCat
    - IsDog
  - mutator methods:
    - Set pen number
    - Clear pen number

- Dog
  - name
  - breed
  - colour
  - size
  - pen number
  - Owner
  - mutator methods:
    - Set pen number
    - Clear pen number

- Cat
  - name
  - breed
  - colour
  - pen number
  - Owner
  - mutator methods:
    - Set pen number
    - Clear pen number

- Owner
  - name
  - address
  - telephone number
  - container of pets
    - access first to last
  - mutator methods:
    - Add Pet

- Pen
  - number
  - size
  - Pet
  - mutator methods:
    - Add Pet
    - Remove Pet

- Kennel
  - name
  - address
  - telephone number
  - container of pens
    - access by index (pen number)
    - access first to last
  - mutator methods:
    - Add Pen

- Container of all owners
  - access by name
  - mutator methods:
    - Add Owner