Object Oriented Design

ENCE464 Embedded Software and Advanced Computing

Course coordinator: Steve Weddell (steve.weddell@canterbury.ac.nz)

Lecturer: Le Yang (le.yang@canterbury.ac.nz)

Department of Electrical and Computer Engineering

Where we’re going today

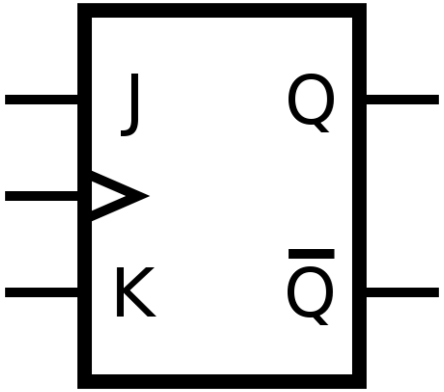
- **Object class**
- Inheritance and polymorphism
- Containment vs. sharing
- Uniform access principle

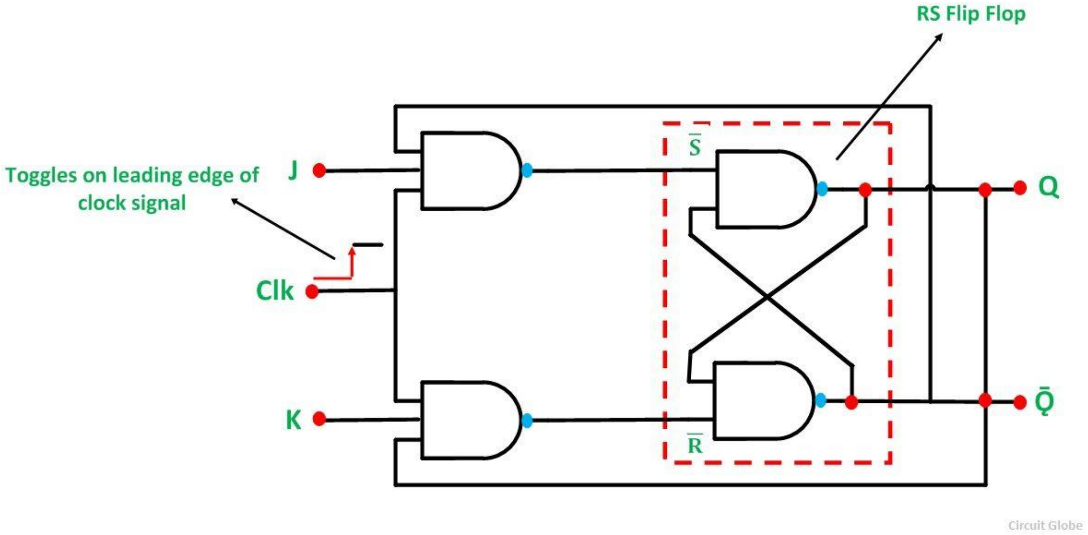
Approaches of Object Oriented Design

- Encapsulation
  - Bundle data and methods that work on the data in one unit
- Abstraction
  - Hide unnecessary details and allow users to realize more complex operations based on provided abstraction without knowing them
- Interface
  - Programming structure offered by an object that allows enforcing certain operations
- Hierarchy
  - Seek to employ natural hierarchy

Object Class (1)

- Example: implement a JK flip flop
  - By hardware or software …
  - With encapsulation, abstraction and interface, users only need to know
    - Number of inputs (input data)
    - Operations on input data
      - Set/Reset input bits
      - Truth table (output)



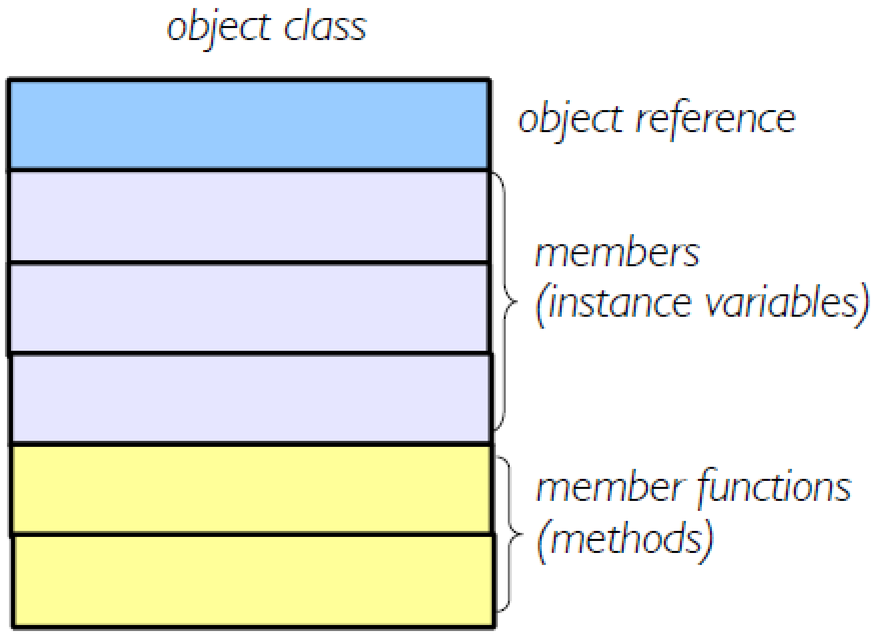


Object Class (2)

- Use types to represent fundamental concepts
  - According to Booch, *an object has* *state*, *behavior* *and* *identity*

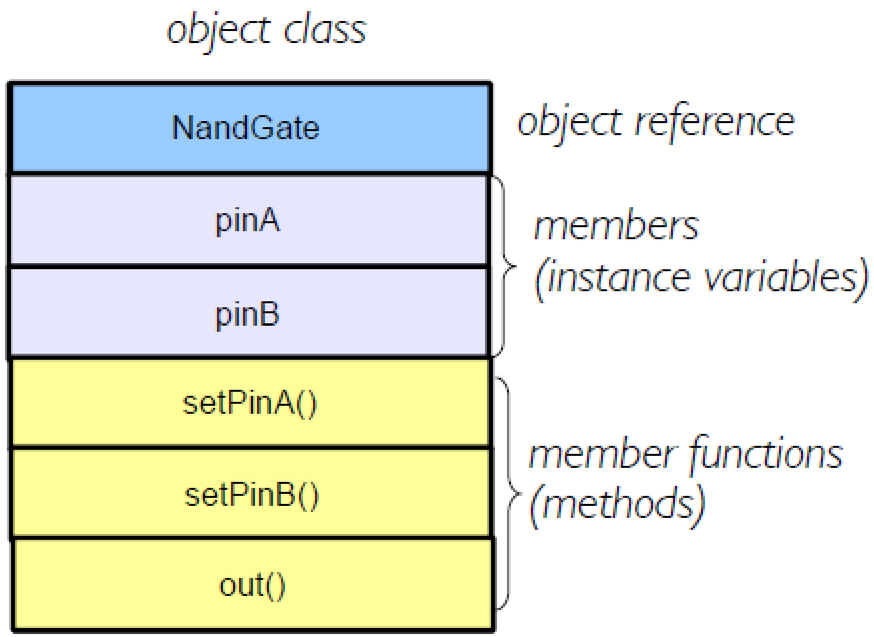
- Object = a set of (member) values + operations on those values
  - Specified as a class

- UML class diagram



Object Class (3)

- Example: NAND gate



object class

NandGate

pinA

pinB

setPinA()

setPinB()

out()

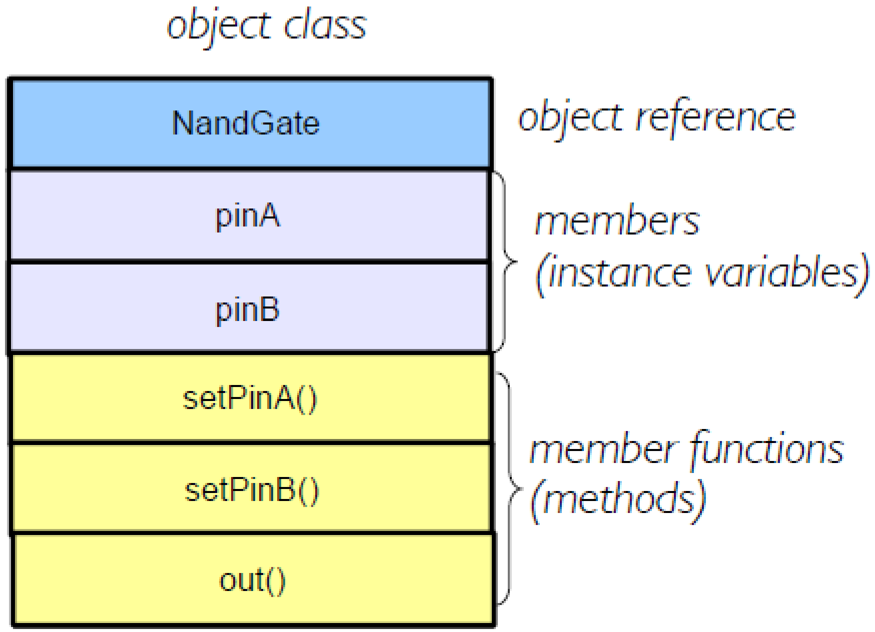
object reference

members
(instance variables)

member functions
(methods)

Object Class (4)

- Example: NOR gate



- What did we observe?

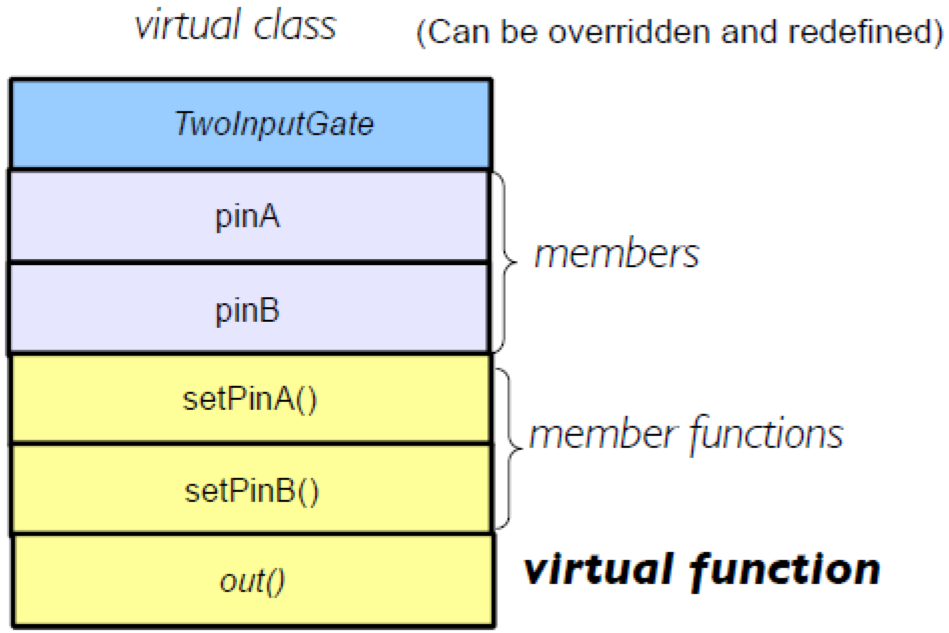
Where we’re going today

- Object class
- **Inheritance and polymorphism**
- Containment vs. sharing
- Uniform access principle

Inheritance

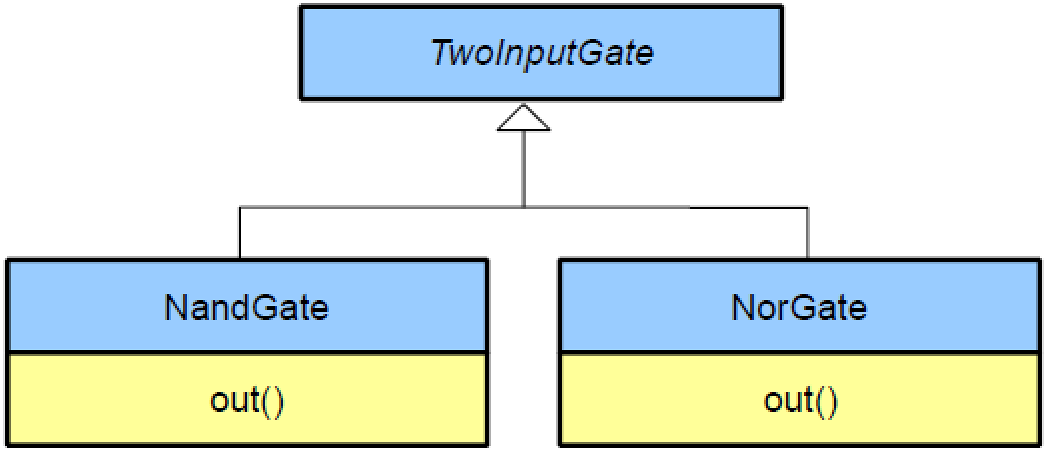
- Inheritance is a way of defining a hierarchy of object classes and exhibiting similarity between classes

- Example: hierarchy of two-input gates
  - Virtual function is a member function that can be redefined in a derived class



Polymorphism

- Polymorphism is a way to handle multiple types using a single interface



- With inheritance, both *NandGate* and *NorGate* inherit from virtual class *TwoInputGate*
- With polymorphism, they have different implementations of *out()* member function

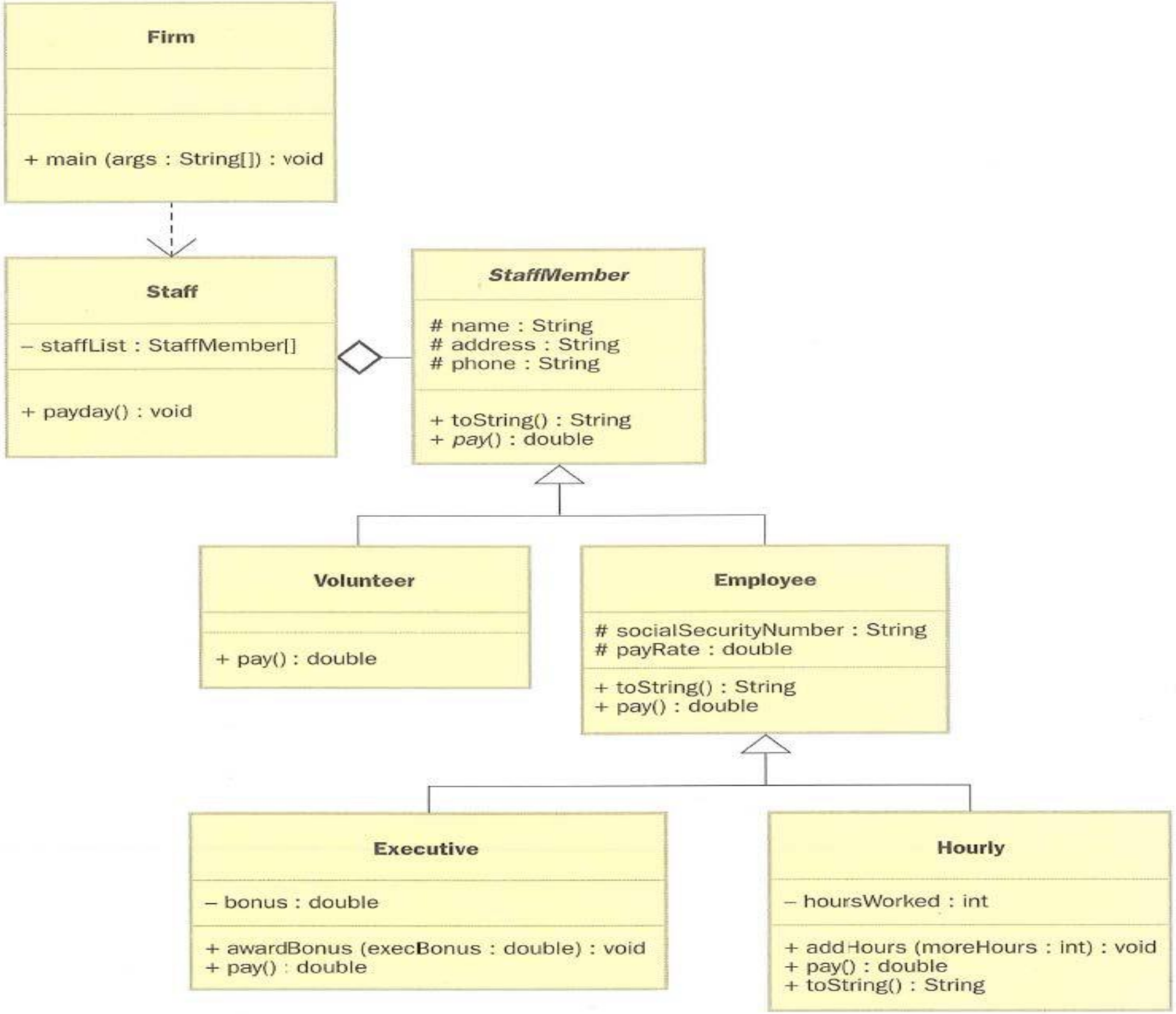


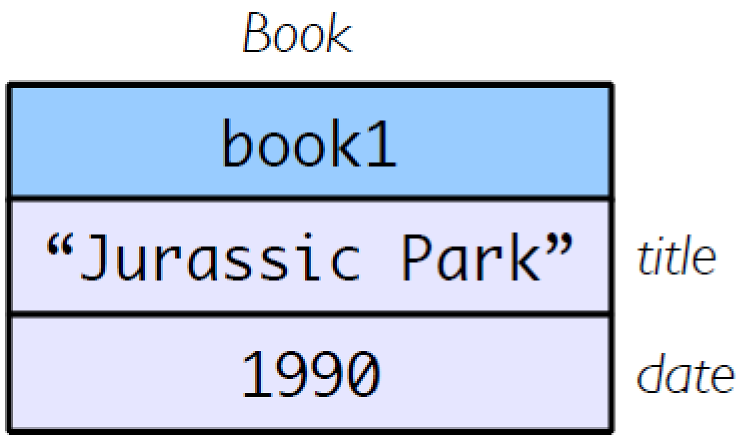
FIGURE 9.1 A class hierarchy of employees

Where we’re going today

- Object class
- Inheritance and polymorphism
- **Containment vs. sharing**
- Uniform access principle

Illustrative Example (1)

- Book object



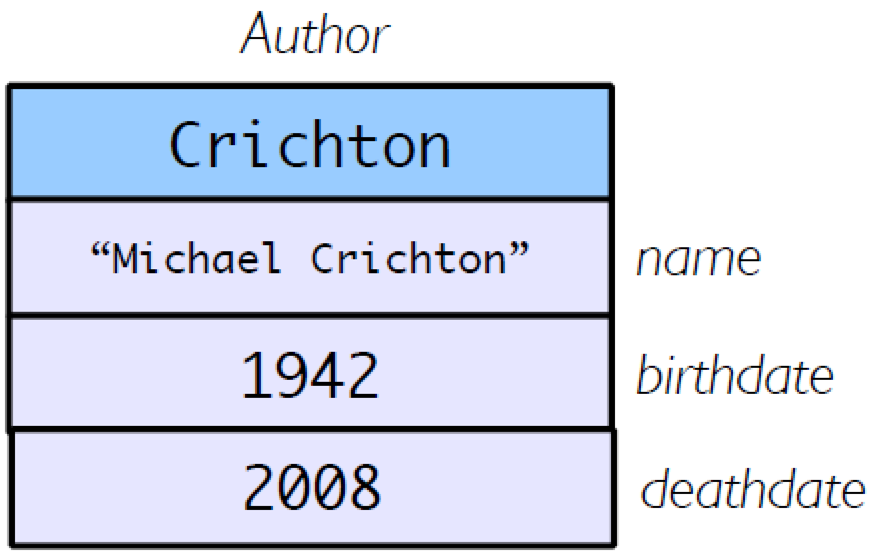
Book

book1

“Jurassic Park” title

1990 date

- Author object



Author

Crichton

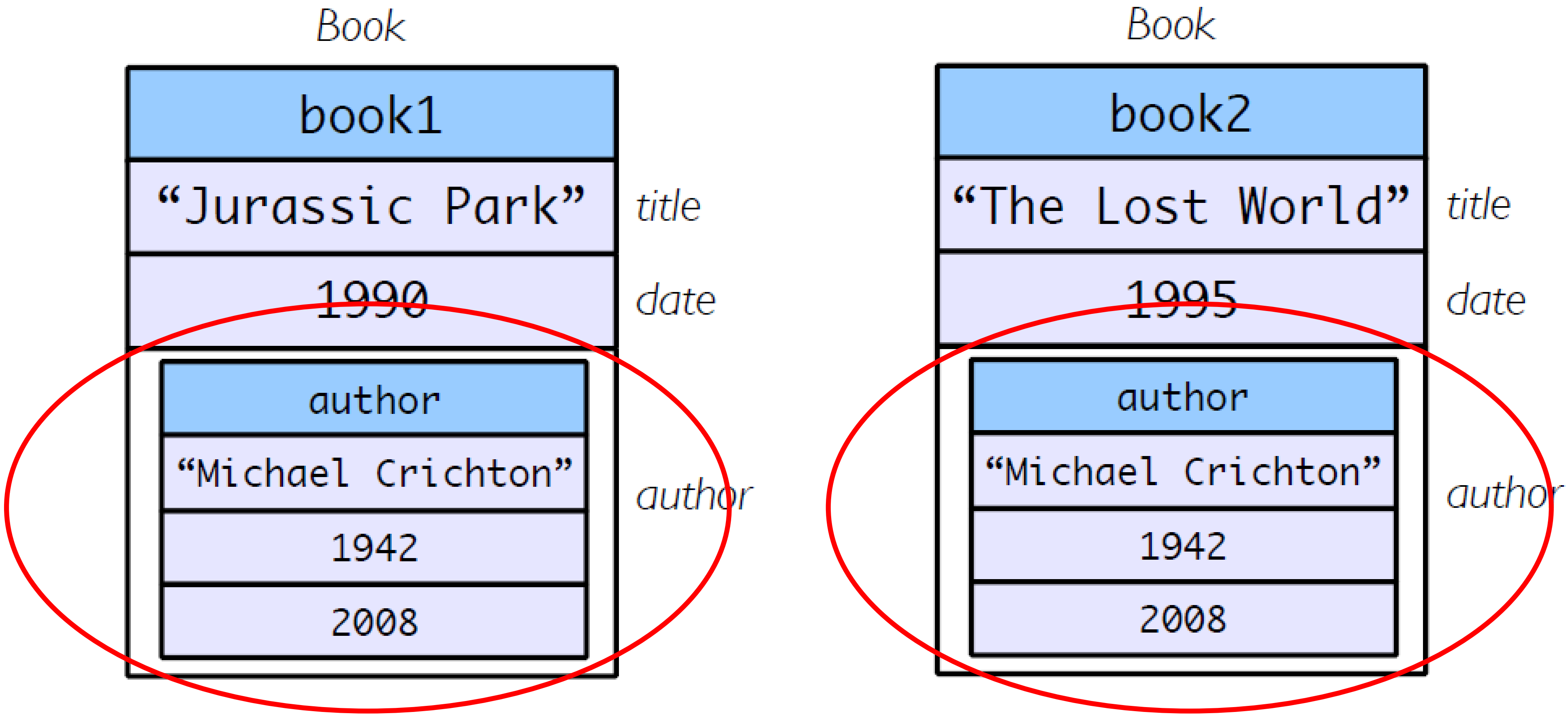
“Michael Crichton” name

1942 birthdate

2008 deathdate

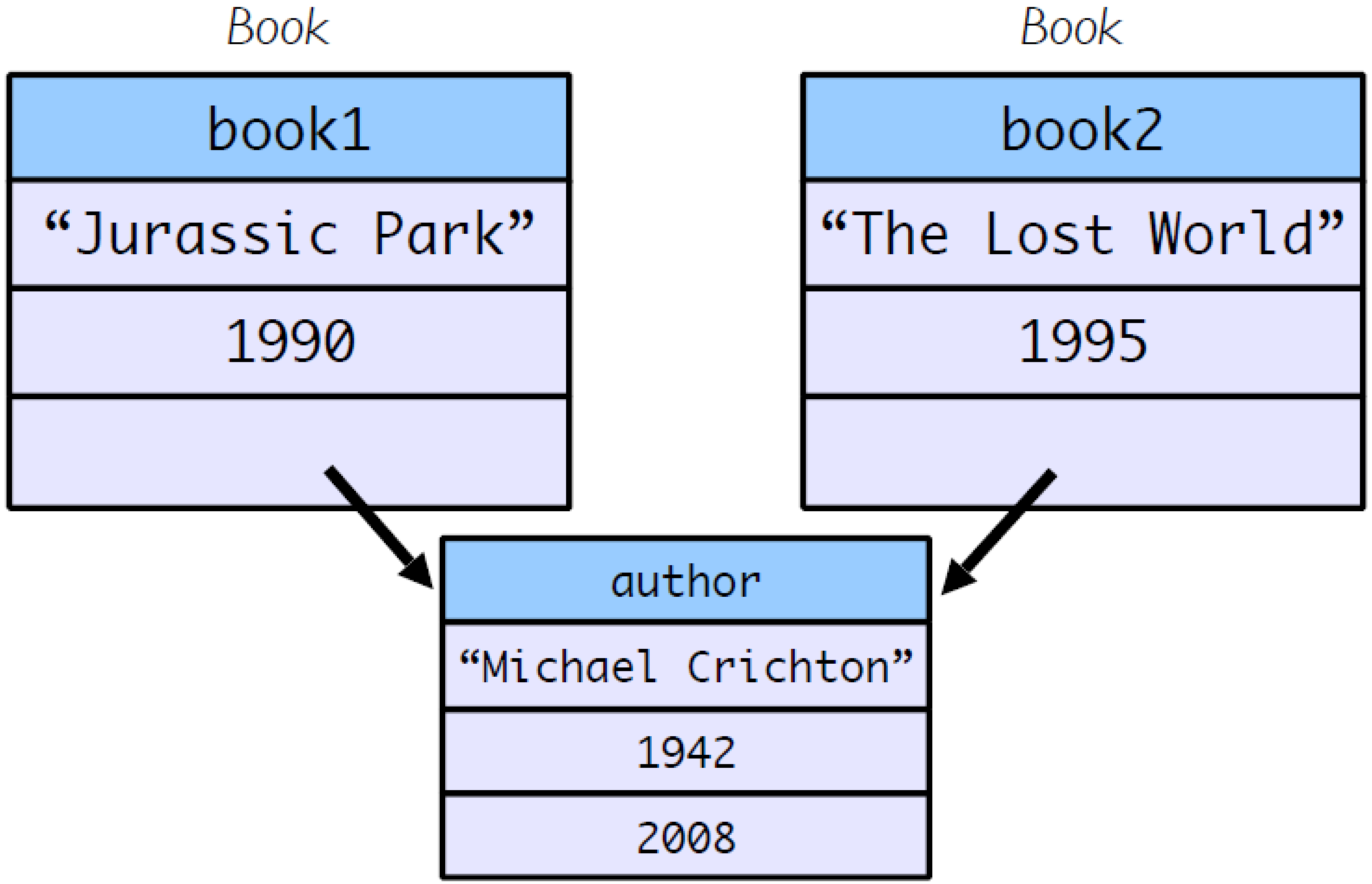
Illustrative Example (2)

- Book objects containing the same author object



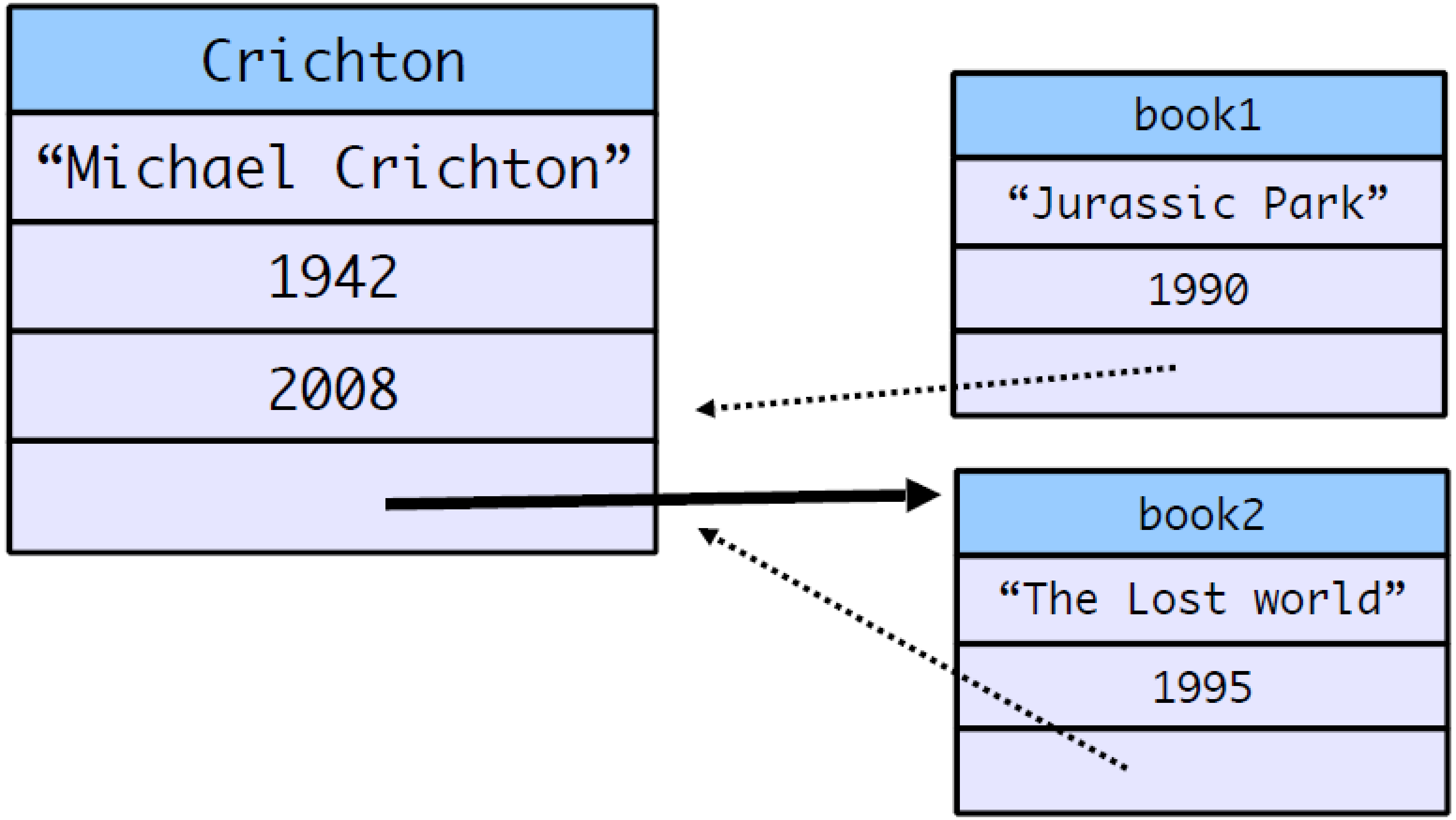
Illustrative Example (3)

- Two different book objects sharing the same author object



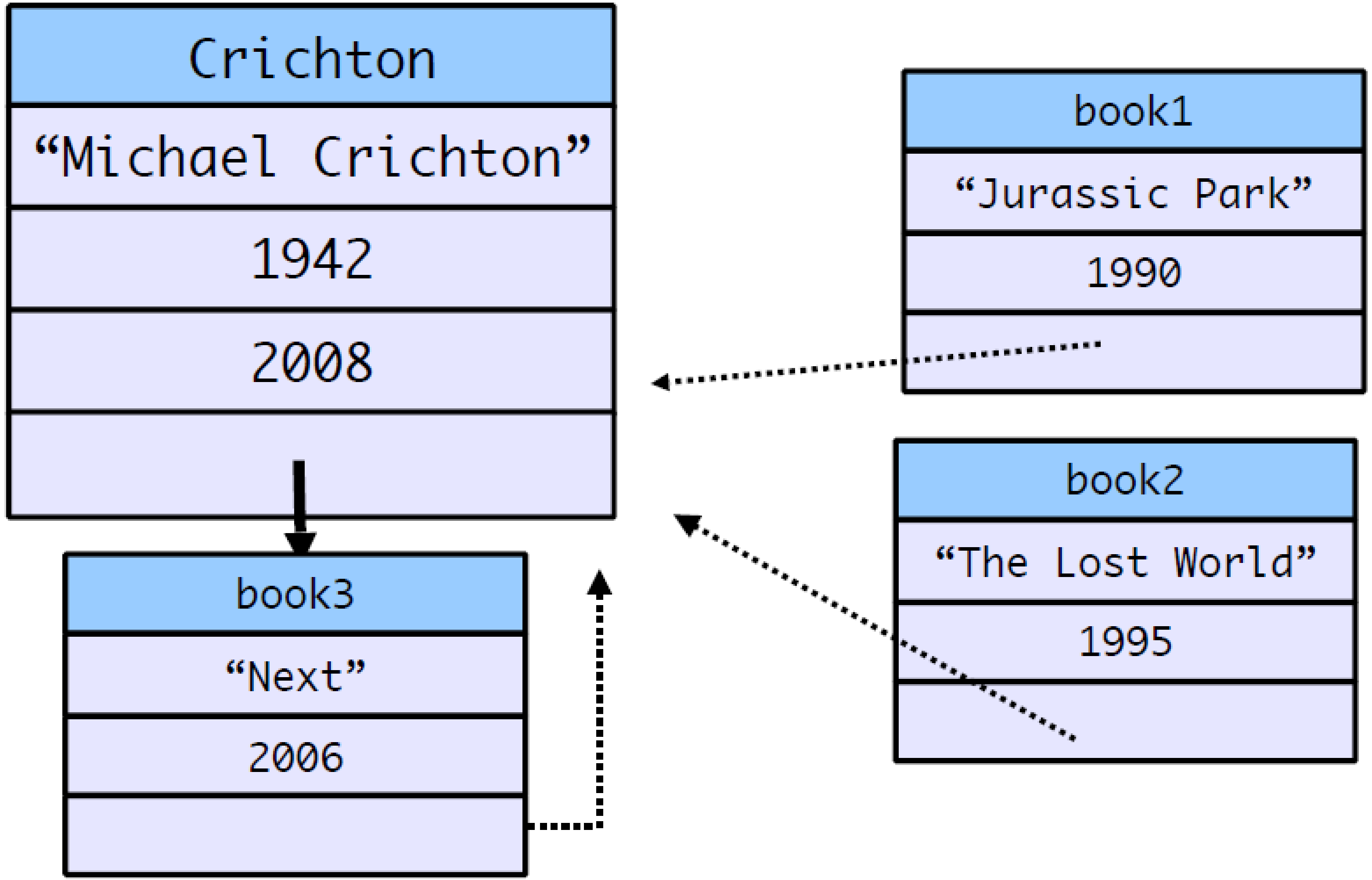
Illustrative Example (4)

- Author object ‘lists’ all the books the author published



Illustrative Example (5)

- Author object ‘lists’ all the books the author published



Where we’re going today

- Object class
- Inheritance and polymorphism
- Containment vs. sharing
- **Uniform access principle**

Uniform Access Principle (1)

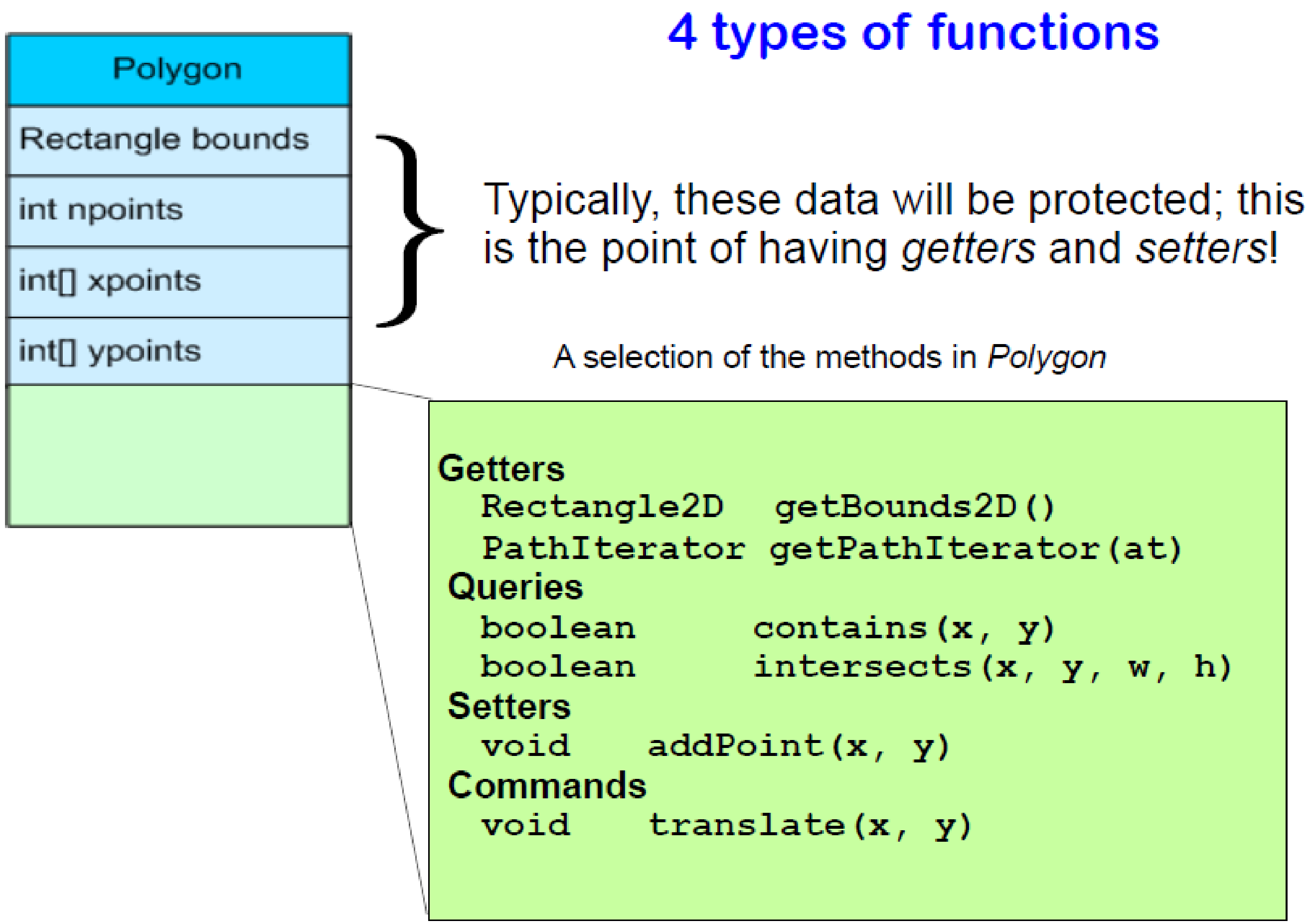
- From Bertrand Meyer
  - *All services offered by a module should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation*

- Maintenance of large software projects or software libraries
  - Example: change an object such that a simple member access is transformed into a method call
    - obj -> variable to obj -> variable ()
    - This might require changing source code in many different locations

  - What if we would like to change obj -> variable () to obj -> variable

Uniform Access Principle (2)

- A simple solution: make all members private and access them via ‘getter’ and ‘setter’ functions



4 types of functions

Polygon

- Rectangle bounds
- int npoints
- int[] xpoints
- int[] ypoints

Typically, these data will be protected; this is the point of having *getters* and *setters*!

A selection of the methods in *Polygon*

```
Getters
  Rectangle2D   getBounds2D()
  PathIterator getPathIterator(at)
Queries
  boolean      contains(x, y)
  boolean      intersects(x, y, w, h)
Setters
  void    addPoint(x, y)
Commands
  void    translate(x, y)
```

Uniform Access Principle (3)

- Getters
  - Return the value of a member in an object
- Setters
  - Set the value of a member according to the function argument
- Queries
  - Return information on the state of the object not directly encoded as a member value
- Commands
  - Send signal to the object to perform certain operations (e.g., change the state)
- Command-Query Separation
  - Queries can be used in any order and anywhere with confidence
  - More care should be taken with commands

Summary

- Encapsulation, Abstraction, Interface
  - Seek to hide details in both implementation and design levels
- Objects have state (member) and behavior (member functions)

- Inheritance allows exploring similarities among classes and building hierarchy
- Polymorphism is a way to handle multiple types using a single interface

- Sharing may be better than containment
- Uniform access principle and command-query separation