Programming Languages

Tevfik Koşar

Lecture - XXIV
April 20th, 2006

Roadmap

- Object Oriented Programming
- Key Features
  - Encapsulation
  - Inheritance
  - Data Abstraction
  - Polymorphism
- Initialization & Finalization
- Static vs Dynamic Method Binding
Object Oriented Programming

- **Object**: any object in real world or an instance of a class in a program
- **Object oriented**: languages and programming techniques based on objects (classes) instead of procedures or functions
- **Objects Capable of**:
  - receiving messages
  - processing data
  - sending messages
    - via object specific functions called “methods”
- **Each object can be viewed as**:
  - an independent little machine with a distinct role or responsibility

Object Oriented Programming

**Goals:**
- **Reduce conceptual load** (minimize amount of detail programmer must think at one time)
- **Provide fault containment** (prevent programmer from using a program component in appropriate ways)
- **Provide independence between program components** (modify internal implementation without changing external code or vice versa)
Object Oriented Programming

Key Features:
1. Encapsulation
2. Inheritance
3. Data Abstraction
4. Polymorphism

1. Encapsulation

- type of privacy applied to some data and methods in a class
- hides irrelevant details from the user
- ensures object can be changed only through established channels
  - eg. the class’s public methods - interface
- clients of the interface perform operations purely through the interface, so if the implementation changes, the clients do not have to change
  - eg. queue implementation, sorting
2. Inheritance

- Mechanism for creating subclasses
- Provides a way to define a subclass as a
  - specialization
  - subtype
  - extension to a more general class
    - eg. human → man, animal → dog, fruit → apple
- Subclasses
  - acquire all of data and methods of the its superclass
  - it can add or change data or methods
  - is-a relationship
    - eg. a man is a human, a dog is an animal, apple is a fruit
    - fruit is a generalization of apple, and apple is an instantiation of fruit
- Intended to help reuse of existing code with little or no modification
  - eg. man and woman objects can share most of the required code

3. Data Abstraction

- Mechanism to reduce details so that one can focus on a few concepts at a time
- Separation between the abstract properties of a data type and the concrete details of its implementation
  - eg. list data type
4. Polymorphism

- Two or more classes reacting differently to the same message
- The programmer does not need to know the exact type of the object in advance, so this behavior can be implemented at run time (dynamic binding).
- The different objects involved need to present a compatible interface to the clients (the calling routines). That is, there must be public methods with the same name and the same parameter sets in all the objects.

4. Polymorphism (cont.)

- Polymorphism allows client programs to be written based only on the abstract interfaces of the objects
- The original client program does not even need to be recompiled (only relinked) in order to make use of new types exhibiting new (but interface-conformant) behavior
Object Initialization

• **Constructors**: initialize an object automatically at the beginning of its lifetime

• In C++, compiler ensures that an appropriate constructor is called for every elaborated object:
  1) `foo b;`       // calls foo:foo()
  2) `foo b(10, 'x');` // calls foo:foo(int, char)
  3) `foo a;
     bar b;
     ...
     foo c(a);   // calls foo:foo(foo&)
     foo d(b);   // calls foo:foo(bar&)

  // single argument constructors are called “copy constructor”

4) `foo a;
   bar b;
   ...
   foo c = a;   // calls foo:foo(foo&)
   foo d = b;   // calls foo:foo(bar&)`
Object Finalization

• When an object is destroyed, the destructor of that class is called → Garbage collection

• In case of a derived class
  - 1. call destructor of the derived class
  - 2. call destructors of base classes (in reverse order of derivation)

Static vs Dynamic Binding

• class person {…}
• class student : public person {…}
• class professor : public person {…}

• student s;
• professor p;

• void person::print_label();   //polymorphic
---
s.print_label();              //calls person::print_label(s)
p.print_label();              //calls person::print_label(p)
Static vs Dynamic Binding

Suppose:
• student::print_label();
• professor::print_label();
---
s.print_label();          // calls student::print_label(s)
p.print_label();          // calls professor::print_label(p)

---

Static vs Dynamic Binding

Suppose:
• person *x = &s;
• person *y = &p;
---
x->print_label();        // ??
y->print_label();        // ??

Does the choice depend on the type of x & y, or on the type of object they refer?
Static vs Dynamic Binding

- First option (use type of the object making the call): static method binding
- Second option (use type of the object referred): dynamic method binding
- Dynamic method binding is central to object-oriented programming!
- C++ uses static binding by default. You need to use "virtual" keyword to use dynamic binding