ICOM 4036 Lecture 6 Object Oriented Programming

Introduction

- Categories of languages that support OOP:
  1. OOP support is added to an existing language
  - C++ (also supports procedural and data-oriented programming)
  - Ada 95 (also supports procedural and data-oriented programming)
  - CLOS (also supports functional programming)
  - Scheme (also supports functional programming)

Introduction

2. Support OOP, but have the same appearance and use the basic structure of earlier imperative languages
   - Eiffel (not based directly on any previous language)
   - Java (based on C++)
3. Pure OOP languages
   - Smalltalk

Object-Oriented Programming

- Paradigm Evolution
  1. Procedural - 1950s-1970s (procedural abstraction)
  2. Data-Oriented - early 1980s (data abstraction)
  3. OOP - late 1980s (inheritance and dynamic binding)

Object-Oriented Programming

- Origins of Inheritance
  - Observations of the mid-late 1980s :
    - Productivity increases can come from reuse
    - Unfortunately,
      - ADTs are difficult to reuse--never quite right
      - All ADTs are independent and at the same level
    - Inheritance solves both--reuse ADTs after minor changes and define classes in a hierarchy to achieve polymorphism

Object-Oriented Programming

- OOP Definitions:
  - ADTs are called classes
  - Class instances are called objects
  - A class that inherits is a derived class or a subclass
  - The class from which another class inherits is a parent class or superclass
  - Subprograms that define operations on objects are called methods

Object-Oriented Programming

- OOP Definitions (continued):
  - Calls to methods are called messages
  - The entire collection of methods of an object is called its message protocol or message interface
  - Messages have two parts--a method name and the destination object
  - In the simplest case, a class inherits all of the entities of its parent

Object-Oriented Programming

- Inheritance can be complicated by access controls to encapsulated entities
  - A class can hide entities from its subclasses
  - A class can hide entities from its clients
  - A class can also hide entities for its clients while allowing its subclasses to see them
- Besides inheriting methods as is, a class can modify an inherited method
  - The new one overrides the inherited one
  - The method in the parent is overriden

Object-Oriented Programming

- There are two kinds of variables in a class:
  1. Class variables - one/class
  2. Instance variables - one/object
- There are two kinds of methods in a class:
  1. Class methods – accept messages to the class
  2. Instance methods – accept messages to objects
- Single vs. Multiple Inheritance
- One disadvantage of inheritance for reuse:
  – Creates interdependencies among classes that complicate maintenance

Object-Oriented Programming

- Polymorphism in OOPLs
  - A polymorphic variable can be defined in a class that is able to reference (or point to) objects of the class and objects of any of its descendants
  - When a class hierarchy includes classes that override methods and such methods are called through a polymorphic variable, the binding to the correct method MUST be dynamic
  - This polymorphism simplifies the addition of new methods

Object-Oriented Programming

- An abstract method is one that does not include a definition (it only defines a protocol)
- An abstract class is one that includes at least one virtual method
- An abstract class cannot be instantiated

Object-Oriented Programming

- Design Issues for OOPLs
  1. The Exclusivity of Objects
     a. Everything is an object
     - Advantage - elegance and purity
     - Disadvantage - slow operations on simple objects (e.g., float)

     b. Add objects to a complete typing system
     - Advantage - fast operations on simple objects
     - Disadvantage - results in a confusing type system (two kinds of entities)

Object-Oriented Programming

1. The Exclusivity of Objects (continued)
   - c. Include an imperative-style typing system for primitives but make everything else objects
   - Advantage - fast operations on simple objects and a relatively small typing system
   - Disadvantage - still some confusion because of the two type systems

Object-Oriented Programming

2. Are Subclasses Subtypes?
   - Does an “is-a” relationship hold between a parent class object and an object of the subclass?

Object-Oriented Programming

3. Implementation and Interface Inheritance
   - If only the interface of the parent class is visible to the subclass, it is interface inheritance
     - Disadvantage - can result in inefficiencies
   - If both the interface and the implementation of the parent class is visible to the subclass, it is implementation inheritance
     - Disadvantage - changes to the parent class require recompilation of subclasses, and sometimes even modification of subclasses

Object-Oriented Programming

4. Type Checking and Polymorphism
   - Polymorphism may require dynamic type checking of parameters and the return value
     - Dynamic type checking is costly and delays error detection
   - If overriding methods are restricted to having the same parameter types and return type, the checking can be static

Object-Oriented Programming

5. Single and Multiple Inheritance
   - Disadvantages of multiple inheritance:
     - Language and implementation complexity (in part due to name collisions)
     - Potential inefficiency - dynamic binding costs more with multiple inheritance (but not much)
   - Advantage:
     - Sometimes it is extremely convenient and valuable

Object-Oriented Programming

6. Allocation and Deallocation of Objects
    - From where are objects allocated?
        - If they all live in the heap, references to them are uniform
        - Simplifies assignment - dereferencing can be implicit
    - Is deallocation explicit or implicit?

Object-Oriented Programming

7. Dynamic and Static Binding
   - Should ALL binding of messages to methods be dynamic?
     - If none are, you lose the advantages of dynamic binding
     - If all are, it is inefficient

Support for OOP in Smalltalk

- Smalltalk is a pure OOP language
  - Everything is an object
  - All computation is through objects sending messages to objects
  - It adopts none of the appearance of imperative languages
- The Smalltalk Environment
  - The first complete GUI system
  - A complete system for software development
  - All of the system source code is available to the user, who can modify it if he/she wants

Support for OOP in Smalltalk

- Type Checking and Polymorphism
  - All binding of messages to methods is dynamic
  - The process is to search the object to which the message is sent for the method; if not found, search the superclass, etc.
  - Because all variables are typeless, methods are all polymorphic

Support for OOP in Smalltalk

- Inheritance
  - All subclasses are subtypes (nothing can be hidden)
  - All inheritance is implementation inheritance
  - No multiple inheritance
  - Methods can be redefined, but the two are not related

Support for OOP in Smalltalk

- Evaluation of Smalltalk
  - The syntax of the language is simple and regular
  - Good example of power provided by a small language
  - Slow compared with conventional compiled imperative languages
  - Dynamic binding allows type errors to go undetected until run time
  - Greatest impact: advancement of OOP

Support for OOP in C++

- General Characteristics:
  - Mixed typing system
  - Constructors and destructors
  - Elaborate access controls to class entities

Support for OOP in C++

- Inheritance
  - A class need not be the subclass of any class
  - Access controls for members are

  1. Private (visible only in the class and friends) (disallows subclasses from being subtypes)
  2. Public (visible in subclasses and clients)
  3. Protected (visible in the class and in subclasses, but not clients)

Support for OOP in C++

- Inheritance (continued)
  - In addition, the subclassing process can be declared with access controls (private or public), which define potential changes in access by subclasses

  a. Private derivation - inherited public and protected members are private in the subclasses

  b. Public derivation public and protected members are also public and protected in subclasses

Example

```
class base_class {
  private:
    int a;
    float x;
  protected:
    int b;
    float y;
  public:
    int c;
    float z;
};

class subclass_1 : public base_class { … };
//  - In this one, b and y are protected and
//     c and z are public

class subclass_2 : private base_class { … };
// - In this one, b, y, c, and z are private,
//    and no derived class has access to any
//    member of base_class
```

Support for OOP in C++

- Reexportation
  - A member that is not accessible in a subclass (because of private derivation) can be declared to be visible there using the scope resolution operator (: :), e.g.,

```
class subclass_3 : private base_class {
        base_class :: c;
        …
      }
```

Support for OOP in C++

- Reexportation (continued)
  - One motivation for using private derivation:
    - A class provides members that must be visible, so they are defined to be public members; a derived class adds some new members, but does not want its clients to see the members of the parent class, even though they had to be public in the parent class definition

Support for OOP in C++

- Multiple inheritance is supported
  - If there are two inherited members with the same name, they can both be reference using the scope resolution operator

Support for OOP in C++

- Dynamic Binding
  - A method can be defined to be **`virtual`**, which means that they can be called through polymorphic variables and dynamically bound to messages
  - A pure virtual function has no definition at all
  - A class that has at least one pure virtual function is an abstract class

Support for OOP in C++

- Evaluation
  - C++ provides extensive access control (unlike Smalltalk)
  - C++ provides multiple inheritance
  - In C++, the programmer must decide at design time which methods will be statically bound and which must be dynamically bound
    - Static binding is faster!
  - Smalltalk type checking is dynamic (flexible, but somewhat unsafe)
  - Because of interpretation and dynamic binding, Smalltalk is ~10 times slower than C++

Support for OOP in Java

- Because of its close relationship to C++, we focus on the differences from that language
- General Characteristics
  - All data are objects except the primitive types
  - All primitive types have wrapper classes that store one data value
  - All objects are heap-dynamic, are referenced through reference variables, and most are allocated with **`new`**

Support for OOP in Java

- Inheritance
  - Single inheritance only, but there is an abstract class category that provides some of the benefits of multiple inheritance (**`interface`**)
  - An interface can include only method declarations and named constants, e.g.,

```
public class Clock extends Applet
   implements Runnable
```

  - Methods can be **`final`** (cannot be overriden)

Support for OOP in Java

- Dynamic Binding
  - In Java, all messages are dynamically bound to methods, unless the method is **`final`** (means it cannot be overriden; therefore, dynamic binding serves no purpose)

Support for OOP in Java

- Encapsulation
  - Two constructs, classes and packages
  - Packages provide a container for classes that are related (can be named or unamed)
  - Entities defined without a scope (access) modifier have package scope, which makes them visible throughout the package in which they are defined - they go in the unnamed package
    - Every class in a package is a friend to the package scope entities elsewhere in the package
    - So, package scope is an alternative to the friends of C++

Support for OOP in C#

- General characteristics
  - Support for OOP similar to Java
  - Includes both classes and structs
  - Classes are similar to Java’s classes
  - Structs are less powerful stack-dynamic constructs

Support for OOP in C#

- Inheritance
  - Uses the syntax of C++ for defining classes
  - A method inherited from parent class can be replaced in the derived class by marking its definition with **`new`**
  - The parent class version can still be called explicitly with the prefix **`base`**

Support for OOP in C#

- Dynamic binding
  - To allow dynamic binding of method calls to methods:
    - The base class method is marked **`virtual`**
    - The corresponding methods in derived classes are marked **`override`**
  - Abstract methods are marked **`abstract`** and must be implemented in all subclasses
  - All C# classes are ultimately derived from a single root class, **`Object`**

Support for OOP in C#

- Evaluation
  - C# is the most recently designed C-based OO language
  - The differences between C#’s and Java’s support for OOP are relatively minor

Support for OOP in Ada 95

- General Characteristics
  - OOP was one of the most important extensions to Ada 83
  - Encapsulation container is a package that defines a tagged type
  - A tagged type is one in which every object includes a tag to indicate during execution its type (the tags are internal)
  - Tagged types can be either private types or records
  - No constructors or destructors are implicitly called

Support for OOP in Ada 95

- Inheritance
  - Subclasses can be derived from tagged types
  - New entities in a subclass are added in a record

Example of a Tagged Type

```
Package PERSON_PKG is
  type PERSON is tagged private;
  procedure DISPLAY(P : in out PERSON);
  private
    type PERSON is tagged
      record
        NAME : STRING(1..30);
        ADDRESS : STRING(1..30);
        AGE : INTEGER;
      end record;
end PERSON_PKG;
with PERSON_PKG; use PERSON_PKG;
 package STUDENT_PKG is
   type STUDENT is new PERSON with
     record
       GRADE_POINT_AVERAGE : FLOAT;
       GRADE_LEVEL : INTEGER;
     end record;
   procedure DISPLAY (ST: in STUDENT);
 end STUDENT_PKG;
```

Note: **`DISPLAY`** is being overridden from **`person_PKG`**

Support for OOP in Ada 95

- Inheritance (continued)
  - All subclasses are subtypes
  - Single inheritance only, except through generics
- Dynamic Binding
  - Dynamic binding is done using polymorphic variables called classwide types
  - e.g., for the tagged type **`PERSON`**, the classwide type is **`PERSON'class`**
  - Other bindings are static
  - Any method may be dynamically bound

The Object Model of JavaScript

- General Characteristics of JavaScript
  - Has little in common with Java
  - Dynamic typing
  - No classes or inheritance or polymorphism
  - Variables can reference objects or can directly access primitive values

The Object Model of JavaScript

- JavaScript Objects
  - An object has a collection of properties, which are either data properties or method properties
  - Appear as hashes, both internally and externally
  - A list of property/value pairs
  - Properties can be added or deleted dynamically
  - A bare object can be created with new and a call to the constructor for Object

```
var my_object = new Object();
```

  - References to properties are with dot notation

Implementing OO Constructs

- Class instance records (CIRs) store the state of an object
- If a class has a parent, the subclass instance variables are added to the parent CIR
- Virtual Method Tables (VMTs) are used for dynamic binding