Chapter 11

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)

***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

Chapter 11

Paradigm Evolution

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

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

Chapter 11

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***
- **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**

Chapter 11

- 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

- 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

- 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 - messages to the class
    2. Instance methods - messages to objects

- Single vs. Multiple Inheritance

Chapter 11

- One disadvantage of inheritance for reuse:

    - Creates interdependencies among classes that complicate maintenance

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

- A *virtual method* is one that does not include a definition (it only defines a protocol)

- A *virtual class* is one that includes at least one virtual method

- A virtual class cannot be instantiated

Chapter 11

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

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

2. Are Subclasses Subtypes?

- Does an is-a relationship hold between a parent class object and an object of the subclass?

Chapter 11

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

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

Chapter 11

5. Single and Multiple Inheritance

- Disadvantages of multiple inheritance:
  - Language and implementation complexity
  - Potential inefficiency - dynamic binding costs more with multiple inheritance (but not much)

- Advantage:
  - Sometimes it is extremely convenient and valuable

6. Allocation and Deallocation of Objects

- From where are objects allocated?
  - If they all live in the heap, references to them are uniform

- Is deallocation explicit or implicit?

Chapter 11

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

Chapter 11

Overview of 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

Introduction to Smalltalk

- *Expressions*
  - Four kinds:
    1. Literals (numbers, strings, and keywords)

    2. Variable names (all variables are references)

    3. Message expressions (see below)

    4. Block expressions (see below)

Chapter 11

- *Message expressions*

- Two parts: the receiver object and the message itself
- The message part specifies the method and possibly some parameters
- Replies to messages are objects

- *Messages can be of three forms:*

1. *Unary* (no parameters)
e.g., `myAngle sin`
(sends a message to the `sin` **method of the** `myAngle` **object)**

2. *Binary* (one parameter, an object)
e.g., `12 + 17`
(sends the message `+ 17` **to the object** `12`**; the object parameter is** `17` **and the method is** `+`**)**

3. *Keyword* (use keywords to organize the parameters)
e.g., `myArray at: 1 put: 5`
(sends the objects `1` **and** `5` **to the** `at:put:` **method of the object** `myArray`**)**

- Multiple messages to the same object can be strung together, separated by semicolons

Chapter 11

Methods

- General form:
  message_pattern [| temps |] statements

  - A message pattern is like the formal parameters of a subprogram
    - For a unary message, it is just the name
    - For others, it lists keywords and formal names
  - temps are just names--Smalltalk is typeless!

Assignments

- Simplest Form:
  name1 `<-` name2

- It is simply a pointer assignment

- RHS can be a message expression
  e.g., `index <- index + 1`

Blocks

- A sequence of statements, separated by periods, delimited by brackets
  e.g.,
  `[index <- index + 1. sum <- sum + index]`

Chapter 11

Blocks (continued)

- **A block specifies something, but doesn t do it**
- **To request the execution of a block, send it the unary message,** `value`
  e.g., `[ ] value`

- **If a block is assigned to a variable, it is evaluated by sending** `value` **to that variable**
  e.g.,

```
addIndex <- [sum <- sum + index]

addIndex value
```

- **Blocks can have parameters, as in**
  `[:x :y | statements]`

- **If a block contains a relational expression, it returns a Boolean object,** `true` **or** `false`

Iteration

- **The objects** `true` **and** `false` **have methods for building control constructs**

- **The method** `WhileTrue:` **from** `Block` **is used for pretest logical loops. It is defined for all blocks that return Boolean objects.**

Chapter 11

Iteration (continued)

e.g.,

```
[count <= 20]
   whileTrue [sum <- sum + count.
              count <- count + 1]
```

- `timesRepeat:` **is defined for integers and can be used to build counting loops**

  e.g.,

```
xCube <- 1.
   3 timesRepeat: [xCube <- xCube * x]
```

Selection

- **The Boolean objects have the method** `ifTrue:ifFalse:` **, which can be used to build selection**

  e.g.,

```
total = 0
   ifTrue: [ ]
   ifFalse: [ ]
```

Chapter 11

Large-Scale Features of Smalltalk

- ***Type Checking and Polymorphism***
  - **All bindings 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**
- ***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**

Chapter 11

C++

- *General Characteristics*:

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

- *Inheritance*

  - A class need not be subclasses 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)

    - 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

Chapter 11

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

Example (book, p. 468)

```
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
```


Chapter 11

- 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;

}
```

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

- Multiple inheritance is supported

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

Chapter 11

- 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

- 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)

Chapter 11

Java

- 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`

- 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)**

Chapter 11

- ***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)**

- ***Encapsulation***

  - **Two constructs, classes and packages**

  - **Packages provide a container for classes that are related (can be named or unamed)**

  - **Entities defined without an 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++**

Chapter 11

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

- Tagged types can be either private types or records

- No constructors or destructors are implicitly called

- Inheritance

- Subclasses are derived from tagged types

- New entities in a subclass are added in a record

Chapter 11

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;
```

- `DISPLAY` **is being overriden from** `PERSON_PKG`

Chapter 11

- ***Inheritance*** **(more)**

  - **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**

Chapter 11

Eiffel

- General Characteristics

  - Has primitive types and objects

  - All objects get three operations, `copy`, `clone`, and `equal`

  - Methods are called *routines*

  - Instance variables are called *attributes*

  - The routines and attributes of a class are together called its *features*

  - Object creation is done with an operator (`!!`)

  - Constructors are defined in a `creation` clause, and are explicitly called in the statement in which an object is created

- *Inheritance*

  - The parent of a class is specified with the `inherit` clause

Chapter 11

- ***Access control***

  - `feature` **clauses specify access control to the entities defined in them**

    - **Without a modifier, the entities in a** `feature` **clause are visible to both subclasses and clients**

    - **With the name of the class as a modifier, entities are hidden from clients but are visible to subclasses**

    - **With the** `none` **modifier, entities are hidden from both clients and subclasses**

  - **Inherited features can be hidden from subclasses with** `undefine`

  - **Abstract classes can be defined by including the** `deferred` **modifier on the class definition**

- ***Dynamic Binding***

  - **Nearly all message binding is dynamic**

  - **An overriding method must have parameters that are assignment compatible with those of the overriden method**

Chapter 11

- ***Dynamic Binding* (continued)**

  - **All overriding features must be defined in a `redefine` clause**

  - **Access to overriden features is possible by putting their names in a `rename` clause**

- ***Evaluation***

  - **Similar to Java in that procedural programming is not supported and nearly all message binding is dynamic**

  - **Elegant and clean design of support for OOP**

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**