CPS 506
Comparative Programming Languages
Object-Oriented Programming Language Paradigm

Topics

- Introduction
- Object-Oriented Programming
- Design Issues for Object-Oriented Languages
- Support for Object-Oriented Programming in
  - Smalltalk
  - C++
  - Java
  - C#
  - Ada 95
  - Ruby
- Implementation of Object-Oriented Constructs

Introduction

- Object Oriented
  - Collection of objects interact with each other
  - Building blocks
    - Object modeling
    - Classification
    - Inheritance
  - Examples
    - Java, Smalltalk, C++, C#

Introduction (con't)

- Many object-oriented programming (OOP) languages
  - Some support procedural and data-oriented programming (e.g., Ada 95 and C++)
  - Some support functional program (e.g., CLOS)
  - Newer languages do not support other paradigms but use their imperative structures (e.g., Java and C#)
  - Some are pure OOP language (e.g., Smalltalk & Ruby)

Object-Oriented Programming

- Abstract data types
- Inheritance
  - Inheritance is the central theme in OOP and languages that support it
- Polymorphism
  - having many forms
  - In OO languages polymorphism refers to the late binding of a call to one of several different implementations of a method in an inheritance hierarchy.

Inheritance

- Productivity increases can come from reuse
  - ADTs are difficult to reuse—always need changes
  - All ADTs are independent and at the same level
- Inheritance allows new classes defined in terms of existing ones, i.e., by allowing them to inherit common parts
- Inheritance addresses both of the above concerns
  - Reuse ADTs after minor changes
  - Define classes in a hierarchy

Object-Oriented Concepts

- ADTs are usually 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 Concepts (con't)

- 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
  - The destination object
- In the simplest case, a class inherits all of the entities of its parent

Object-Oriented Concepts (con't)

- 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 Concepts (con't)

- There are two kinds of variables in a class:
  - Class variables - one/class
  - Instance variables - one/object
- There are two kinds of methods in a class:
  - Class methods – accept messages to the class
  - Instance methods – accept messages to objects
- Single vs. Multiple Inheritance

Dynamic Binding

- 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 will be dynamic
- Allows software systems to be more easily extended during both development and maintenance (example?)

Dynamic Binding Concepts

- 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

Design Issues for OOP Languages

- The Exclusivity of Objects
  - Pure vs. impure OO languages
- Are Subclasses Subtypes?
  - With some conditions
- Type Checking and Polymorphism
  - Static vs. delay type checking
- Single and Multiple Inheritance
  - Complexity
    - Diamond inheritance
  - Efficiency
    - Diamond inheritance

Design Issues for OOP Languages

- Object Allocation and Deallocation
  - Uniform behavior
  - Explicit or implicit
- Dynamic and Static Binding
- Nested Classes
  - Information hiding
  - Visibility from both sides
    - Nesting class
    - Nested class
- Initialization of Objects
  - Manually or implicit
  - Initialization of the super-class (implicit or explicit)

The Exclusivity of Objects

- Everything is an object
  - Advantage - elegance and purity
  - Disadvantage - slow operations on simple objects
- Add objects to a complete typing system
  - Advantage - fast operations on simple objects
  - Disadvantage - results in a confusing type system (two kinds of entities)
- 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

Are Subclasses Subtypes?

- Does an "is-a" relationship hold between a parent class object and an object of the subclass?
  - If a derived class is-a parent class, then objects of the derived class must behave the same as the parent class object
- A derived class is a subtype if it has an is-a relationship with its parent class
  - Subclass can only add variables and methods and override inherited methods in "compatible" ways

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

Single and Multiple Inheritance

- Multiple inheritance allows a new class to inherit from two or more classes
- 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 quite convenient and valuable

Allocation and De-Allocation of Objects

- From where are objects allocated?
  - If they behave line the ADTs, they can be allocated from anywhere
    - Allocated from the run-time stack
    - Explicitly create on the heap (via new)
  - If they are all heap-dynamic, references can be uniform thru a pointer or reference variable
    - Simplifies assignment - dereferencing can be implicit
  - If objects are stack dynamic, there is a problem with regard to subtypes
- Is deallocation explicit or implicit?

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
- Allow the user to specify

Nested Classes

- If a new class is needed by only one class, there is no reason to define so it can be seen by other classes
  - Can the new class be nested inside the class that uses it?
  - In some cases, the new class is nested inside a subprogram rather than directly in another class
- Other issues:
  - Which facilities of the nesting class should be visible to the nested class and vice versa

Initialization of Objects

- Are objects initialized to values when they are created?
  - Implicit or explicit initialization

- How are parent class members initialized when a subclass object is created?

Support for OOP in Smalltalk

- Smalltalk is a pure OOP language
  - Everything is an object
  - All computation is through objects sending messages to objects
  - None of the appearances of imperative languages
  - All objected are allocated from the heap
  - All deallocation is implicit

Support for OOP in Smalltalk (con't)

- Type Checking and Polymorphism
  - All binding of messages to methods is dynamic
    - The process is to search the object to which message is sent for the method; if not found, search the superclass, etc. up to the system class which has no superclass
  - The only type checking in Smalltalk is dynamic
  - Type error occurs when a message is sent to an object that has no matching method

Support for OOP in Smalltalk (con't)

- Inheritance
  - A Smalltalk subclass inherits all of the instance variables, instance methods, and class methods of its superclass
  - All subclasses are subtypes (nothing can be hidden)
  - No multiple inheritance

Support for OOP in Smalltalk (con't)

- 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
  - Introduced the graphical user interface
  - Greatest impact: advancement of OOP

Support for OOP in C++

- General Characteristics:
  - Evolved from C and SIMULA 67
  - Among the most widely used OOP languages
  - Mixed typing system
    - Procedural and OO
  - Constructors and destructors
  - Elaborate access controls to class entities

Support for OOP in C++ (con't)

- Inheritance
  - A class need not be the subclass of any class
  - Access controls for members are
    - Private (visible only in the class and friends) (disallows subclasses from being subtypes)
    - Public (visible in subclasses and clients)
    - Protected (visible in the class and in subclasses, but not clients)

Support for OOP in C++ (con't)

- In addition, the subclassing process can be declared with access controls (private or public), which define potential changes in access by subclasses
  - Private derivation - inherited public and protected members are private in the subclasses
  - Public derivation - public and protected members are also public and protected in subclasses

Inheritance Example in C++

```
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
```

Re-exportation in C++

- 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;
        …
}
```

Re-exportation (con't)

- 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++ (con't)

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

Support for OOP in C++ (con't)

- 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++ (con't)

- Evaluation
  - C++ provides extensive access controls (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++, focus is 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
  - A finalize method is implicitly called when the garbage collector is about to reclaim the storage occupied by the object

Support for OOP in Java (con't)

- Inheritance
  - Single inheritance supported 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 interface Comparable <T> {
        public int comparedTo (T b);
}
```

  - Methods can be final (cannot be overriden)

Support for OOP in Java (con't)

- Dynamic Binding
  - In Java, all messages are dynamically bound to methods, unless the method is final (i.e., it cannot be overriden, therefore dynamic binding serves no purpose)
  - Static binding is also used if the method is static or private both of which disallow overriding

Support for OOP in Java (con't)

- Several varieties of nested classes
- All are hidden from all classes in their package, except for the nesting class
- Nonstatic classes nested directly are called innerclasses
  - An innerclass can access members of its nesting class
  - A static nested class cannot access members of its nesting class
- Nested classes can be anonymous

Support for OOP in Java (con't)

- Evaluation
  - Design decisions to support OOP are similar to C++
  - No support for procedural programming
  - No parentless classes
  - Dynamic binding is used as "normal" way to bind method calls to method definitions
  - Uses interfaces to provide a simple form of support for multiple inheritance

Reflection

- Reflection is a mechanism whereby a program can discover and use the methods of any of its objects and classes.

- Reflection is essential for programming tools that allow plugins (such as Eclipse -- www.eclipse.org) and for JavaBeans components.

Reflection (con't)

- In Java the `Class` class provides the following information about an object:

- The superclass or parent class.
- The names and types of all fields.
- The names and signatures of all methods.
- The signatures of all constructors.
- The interfaces that the class implements.

```
Class class = obj.getClass( );
Constructor[ ] cons = class.getDeclaredConstructors( );
for (int i=0; i < cons.length; i++) {
    System.out.print(class.getName( ) + "(" );
    Class[ ] param = cons[i].getParameterTypes( );
    for (int j=0; j < param.length; j++) {
        if (j > 0) System.out.print(", ");
            System.out.print(param[j].getName( );
    }
    System.out.println( ")" );
}
```

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 (e.g., no inheritance)

Support for OOP in C# (con't)

- 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`:

```
base.Draw()
```

Support for OOP in C# (con't)

- 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# (con't)

- Nested Classes
  - A C# class that is directly nested in a nesting class behaves like a Java static nested class
  - C# does not support nested classes that behave like the non-static classes of Java

Support for OOP in C# (con't)

- 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 (con't)

- Inheritance
  - Subclasses can be derived from tagged types
  - New entities are added to the inherited entities by placing them in a record definition
  - All subclasses are subtypes
  - No support for multiple inheritance
    - A comparable effect can be achieved using generic classes

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 (con't)

- Dynamic Binding
  - Dynamic binding is done using polymorphic variables called classwide types
    - For the tagged type `Person`, the classwide type is `Person'` class
  - Other bindings are static
  - Any method may be dynamically bound
  - Purely abstract base types can be defined in Ada 95 by including the reserved word `abstract`

Support for OOP in Ada 95 (con't)

- Evaluation
  - Ada offers complete support for OOP
  - C++ offers better form of inheritance than Ada
  - Ada includes no initialization of objects (e.g., constructors)
  - Dynamic binding in C-based OOP languages is restricted to pointers and/or references to objects; Ada has no such restriction and is thus more orthogonal

Support for OOP in Ruby

- General Characteristics
  - Everything is an object
  - All computation is through message passing
  - Class definitions are executable, allowing secondary definitions to add members to existing definitions
  - Method definitions are also executable
  - All variables are type-less references to objects
  - Access control is different for data and methods
    - It is private for all data and cannot be changed
    - Methods can be either public, private, or protected
    - Method access is checked at runtime
  - Getters and setters can be defined by shortcuts

Support for OOP in Ruby (con't)

- Inheritance
  - Access control to inherited methods can be different than in the parent class
    - Subclasses are not necessarily subtypes
  - Mixins can be created with modules, providing a kind of multiple inheritance
- Dynamic Binding
  - All variables are typeless and polymorphic
- Evaluation
  - Does not support abstract classes
  - Does not fully support multiple inheritance
  - Access controls are weaker than those of other languages that support OOP

Implementing OO Constructs

- Two interesting and challenging parts
  - Storage structures for instance variables
  - Dynamic binding of messages to methods

Instance Data Storage

- Class instance records (CIRs) store the state of an object
  - Static (built at compile time)
- If a class has a parent, the subclass instance variables are added to the parent CIR
- Because CIR is static, access to all instance variables is done as it is in records
  - Efficient

Dynamic Binding of Methods Calls

- Methods in a class that are statically bound need not be involved in the CIR; methods that will be dynamically bound must have entries in the CIR
  - Calls to dynamically bound methods can be connected to the corresponding code thru a pointer in the CIR
  - The storage structure is sometimes called virtual method tables (vtable)
  - Method calls can be represented as offsets from the beginning of the vtable

Summary

- OO programming involves three fundamental concepts: ADTs, inheritance, dynamic binding
- Major design issues: exclusivity of objects, subclasses and subtypes, type checking and polymorphism, single and multiple inheritance, dynamic binding, explicit and implicit de-allocation of objects, and nested classes
- Smalltalk is a pure OOL
- C++ has two distinct type system (hybrid)
- Java is not a hybrid language like C++; it supports only OO programming
- C# is based on C++ and Java
- Ruby is a new pure OOP language; provides some new ideas in support for OOP
- JavaScript is not an OOP language but provides interesting variations
- Implementing OOP involves some new data structures