POWER PRESENTATION

Fundamental OO Concepts

- Encapsulation
- Inheritance
- Dynamic Method Binding

Encapsulation

- **Encapsulation**
  - Encapsulation allows the programmer to **group** data and the subroutines that operate on them together **in one place**, and to **hide irrelevant details** from the user.

- **Information Hiding**
  - Making objects and algorithms invisible to portions of the system that do not need them.

Modules

- If a module M exports a type T, the rest of the program can only pass T to subroutines exported from M.
  - T is said to be an opaque type.

```
var Database : module
  exports (tuple with (:=, name))
  ...
  type tuple = record
    var name : packed array 1..80 of char
        ...
  end tuple
   . . .
```

- What can the code outside the Database module do?

Module Changing

- Body is Changed

- Private Part of Header is Changed

- Public Part of Header is Changed

Classes can limit visibility

- Private
- Protected
- Public
- Package (in some languages, e.g. Java)

Derived class can restrict visibility

- Private
  - Protected and public members of base class are private in derived class.
- Protected
  - Protected and public members of base class are protected in derived class.
- Public
  - Protected and public members of base class are protected and public in derived class.
- Private members of base class aren’t visible in derived class.

Initialization and Finalization

Four Important Issues

- Choosing a Constructor
- References and Values
- Execution Order
- Garbage Collection
  - We’ve seen that already

Choosing a Constructor

- Object-Oriented Languages allow classes to have zero, one or more different constructors.
- Two ways to distinguish between constructors
  - Different Names
  - Different Number and Types of Arguements

Constructors

- Eiffel code:

-
  ```
  class COMPLEX
  creation
          new_cartesian, new_polar
          …
          new_cartesian(x_val, y_va; : REAL) is
                  …
          new_polar(rho, theta : REAL) is
                  …
  ```

-
  ```
  class mydata {
  public:
          mydata(string data);
          mydata(int data);
          mydata();
  };
  ```

References and Values

- C++ vs. Java
  - Java uses reference, C++ you can specify
- Reference
  - Every object is created explicitly so it is easy to make sure the correct constructor is called.
  - More elegant, but requires allocation from heap and extra indirections on every access of the object.
- Value
  - More efficient but harder to control initialization

Execution Order

- If class B is derived from class A, A constructor is called before B constructor
  - To get arguments to the A constructor, you must use an intializer list

```
class foo : bar {
        ...
}
foo::foo (foo_params) :  bar(bar_params) {
        …
```

  - The part after the colon is a call to bar’s constructor

Destructors and Garbage Collection

- When an object is destroyed, the destructor is called for the derived class first, then the destructors of the base classes are called.
  - Reverse order of derivation
- Destructors purpose is to return allocated space back to the heap
- Many languages provide automatic garbage collection
  - Java, Smalltalk, Eiffel, etc.

Dynamic Method Binding

Polymorphism

- A derived class (D) has all the members of its base class (C)
  - Class D can be used anytime class C is expected.
  - If class D does not hide any publicly visible members of C then D is a subtype of C.
- If class D is used in place of class C, this is a form of polymorphism.

Polymorphism Example

```
class person { …
class student : public person { …
class professor : public person { …

student s;
professor p;
…
person *x = &s;
person *y = &p;
```

Dynamic vs. Static binding

- **Static method binding** uses the type of the reference:

```
s.print_mailing_label();
p.print_mailing_label();
```

- **Dynamic method binding** uses the class of the object that is referred/pointed to:

```
x->print_mailing_label();
y->print_mailing_label();
```

Dynamic method binding

- Dynamic method binding: calls to virtual methods are dispatched to the appropriate implementation at run time based on the class of the object
  - **Simula:** virtual methods listed at beginning of class declaration

```
CLASS Person;
    VIRTUAL: PROCEDURE PrintMailingLabel;
BEGIN
    …
END Person;
```

Dynamic method binding

- **C++:** keyword “virtual” prefixes function declaration

```
class person {
public:
    virtual void print_mailing_label ();
    …
}
```

- This requires keeping a virtual method table along with each object
  - More on this in a bit…

Abstract Methods

- Bodyless virtual methods

In C++: called pure virtual method, created by following a procedure declaration with an assignment to zero.

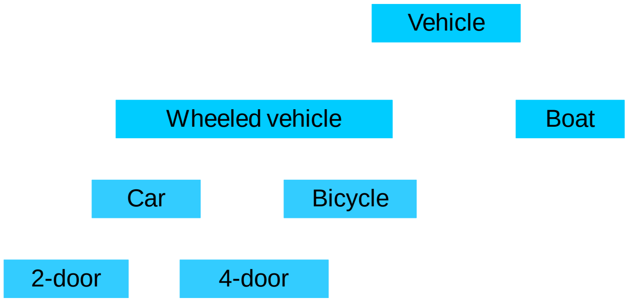
```
class person {
    …
public:
    virtual void print_mailing_label() = 0;
```

Abstract Classes

- Class that contains one or more abstract methods
  - Java: called an interface (which has only abstract methods)
- Generally not possible to declare object of an abstract class b/c it would be missing at least one member
  - But you can do so in C++
- Serves as a base for concrete classes.
  - Concrete class must provide a definition for every abstract method it inherits
- Application to dynamic method binding: allows code that calls methods of objects of a base class, assuming that the concrete methods will be invoked at run time.

Arrange concepts into an inheritance hierarchy

- Concepts at higher levels are more general
- Concepts at lower levels are more specific (inherit properties of concepts at higher levels)



Vehicle

Wheeled vehicle

Boat

Car

Bicycle

2-door

4-door

C++ and inheritance

- The language mechanism by which one class acquires the properties (data and operations) of another class
- Base Class (or superclass): the class being inherited from
- Derived Class (or subclass): the class that inherits

Advantages of inheritance

- When a class inherits from another class, there are **three** benefits:
- (1) You can *reuse* the methods and data of the existing class

  (2) You can *extend* the existing class by adding new data and new methods

  (3) You can *modify* the existing class by overloading its methods with your own implementations

Inheritance and accessibility

- A class inherits the *behavior* of another class and enhances it in some way
- Inheritance *does not* mean inheriting access to another class' private members

Rules for building a class hierarchy

- Derived classes are **special cases** of base classes
- A derived class **can also serve** as a base class for new classes.
- There is no limit on the **depth of inheritance** allowed in C++ (as far as it is within the limits of your compiler)
- It is possible for a class to be a base class for **more than one** derived class

Static vs. dynamic binding

- Static Binding: the determination of which method to call at compile time
- Dynamic Binding: the determination of which method to call at run time

Virtual Functions

- C++ uses *virtual functions* to implement run-time binding.
- To force the compiler to generate code that guarantees dynamic binding, the word *virtual* should appear before the function declaration *in the definition of the base class.*

REFRENCES

- WWW.CS.VIRGINIA.EDU
- WWW.CSE.UNR.EDU