Programming Paradigms
Data Abstraction and Object-Orientation (Part 4)
Overview

- Encapsulation and Information Hiding
- Inheritance
- Initialization and Finalization
- Dynamic Method Binding
- Mix-in and Multiple Inheritance
Static vs. Dynamic Method Binding

- Given: Subclass that defines a method already defined in the superclass
- How to decide which method gets called?
  - Based on type of variable
  - Based on type of object the variable refers to
Example

class person { ... }
class student : public person { ... }
class professor : public person { ... }

void person::print_mailing_label() { ... }
void student::print_mailing_label() { ... }
void professor::print_mailing_label() { ... }

student s;
professor p;

person *x = &s;
person *y = &p;

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

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

```cpp
class person { ... }
class student : public person { ... }
class professor : public person { ... }

void person :: print_mailing_label () { ... }
void student :: print_mailing_label () { ... }
void professor :: print_mailing_label () { ... }

student s;
professor p;

person *x = &s;
person *y = &p;

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

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

Subclasses also define method print mailing label
### Example

```cpp
class person { ... }
class student : public person { ... }
class professor : public person { ... }

void person::print_mailing_label() { ... }
void student::print_mailing_label() { ... }
void professor::print_mailing_label() { ... }

student s;
professor p;

person *x = &s;
person *y = &p;

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

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

**Variables of subtypes**

**Variables of supertype**
Example

class person { ... }
class student : public person { ... }
class professor : public person { ... }

void person::print_mailing_label() { ... }
void student::print_mailing_label() { ... }
void professor::print_mailing_label() { ... }

student s;
professor p;

person *x = &s;
person *y = &p;

s.print_mailing_label();
p.print_mailing_label();
x->print_mailing_label();
y->print_mailing_label();
Example

class person { ... }
class student : public person { ... }
class professor : public person { ... }

void person::print_mailing_label() { ... }
void student::print_mailing_label() { ... }
void professor::print_mailing_label() { ... }

student s;
professor p;

person *x = &s;
person *y = &p;

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

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

Which methods to call here?
Answer 1: Bind methods based on type of variable

- Can be statically resolved (i.e., at compile time)
- Will call `print_mailing_label of person` because `x` and `y` are pointers to `person`
Dynamic Method Binding

- **Answer 2:** Bind methods based on type of object the variable refers to

  - In general, cannot be *resolved* compile time, but only *at runtime*

  - Will call `print_mailing_label` of student for x because x points to a student project (and likewise for y and professor)
Pros and Cons

Static method binding
- No performance penalty because resolved at compile-time
- But: Subclass cannot control its own state

Dynamic method binding
- Subclass can control its state
- But: Performance penalty of runtime method dispatch
Example (C++)

class text_file {
    char *name;
    // file pointer
    long position;
    public:
    void seek(long offset) {
        // (...)
    }
};

class read_ahead_text_file : public text_file {
    char *upcoming_chars;
    public:
    void seek(long offset) {
        // redefinition
    }
};
class text_file {
    char *name;
    // file pointer
    long position;
public:
    void seek(long offset) {
        // (...)
    }
};

class read_ahead_text_file : public text_file {
    char *upcoming_chars;
public:
    void seek(long offset) {
        // redefinition
    }
};

Subclass needs to change upcoming_chars in seek

But with static method binding, cannot guarantee that it gets called
Support in Popular PLs

Static method binding

Dynamic method binding
Support in Popular PLs

Static method binding

Dynamic method binding

Dynamic binding for all methods: Smalltalk, Python, Ruby
Support in Popular PLs

Static method binding

Dynamic method binding by default, but method or class can be marked as not overridable: Java, Eiffel
Support in Popular PLs

Static method binding

Dynamic method binding

Static binding by default, but programmer can specify dynamic binding
Java, Eiffel: Final/frozen Methods

- Mark individual **methods** (or classes) as non-overridable
  - Java: *final* keyword for methods and classes
  - Eiffel: *frozen* keyword for individual methods
C++, C#: Overriding vs. Redefining

Override method: Dynamic binding
Redefine methods with same name: Static binding

- C++: Superclass must mark method as `virtual` to allow overriding
- C#: Subclass must mark method with `override` to override the superclass method
Demo

Virtual.cpp
Abstract Methods and Classes

- **Abstract method**
  - Implementation omitted
  - Subclass must provide it

- **Abstract class**: Class with at least one abstract method
Demo

Abstract.java

Abstract.cpp
Quiz: Method Binding

# Pseudo code

class A:
    void foo():
        ...
    void bar():
        ...

class B extends A:
    void bar():
        ...

A x = new B()
B y = x
x.bar() # call 1
y.bar() # call 2

What is called when

a) PL always uses dynamic method binding
b) PL always uses static method binding
c) PL always uses static method binding unless overriding method marked with override
Quiz: Method Binding

# Pseudo code
```python
class A:
    void foo():
        ...
    void bar():
        ...

class B extends A:
    void bar():
        ...
A x = new B()
B y = x
x.bar() # call 1
y.bar() # call 2
```

What is called when
a) PL always uses dynamic method binding
b) PL always uses static method binding
c) PL always uses static method binding unless overriding method marked with `override`

Please vote via Ilias.
Quiz: Method Binding

# Pseudo code

class A:
    void foo():
        ...
    void bar():
        ...

class B extends A:
    void bar():
        ...

A x = new B()
B y = x
x.bar() # call 1
y.bar() # call 2

What is called when

a) PL always uses dynamic method binding
b) PL always uses static method binding
c) PL always uses static method binding unless overriding method marked with override

Please vote via Ilias.
Quiz: Method Binding

# Pseudo code

class A:
    void foo():
        ...
    void bar():
        ...

class B extends A:
    void bar():
        ...

A x = new B()
B y = x
x.bar() # call 1
y.bar() # call 2

What is called when
a) PL always uses dynamic method binding
b) PL always uses static method binding
c) PL always uses static method binding unless overriding method marked with override

Please vote via Ilias.
Method Lookup

With dynamic method binding, how does the program find the right method to call?

- Most common implementation:
  Virtual method table ("vtable")

- Every object points to table with its methods

- Table is shared among all instances of a class
class foo {
    int a;
    double d;
    char c;
    public:
    virtual void k() { ... }
    virtual int l() { ... }
    virtual void m() { ... }
    virtual double n() { ... }
}

F

foo's vtable

code pointers

Compiler-generated code for dynamic method binding for F.m() :

r1 := F
r2 := *r1 // vtable address
r2 := *(r2 + (3-1)*4) // assuming size of 1 address is 4 bytes

r2 := *r2

F;
Implementation of Inheritance

- Representation of **subclass instance**, including its vtable: Fully compatible with superclass
  - Can use subclass instance like a superclass instance without additional code
class bar : public foo {
    int w;

    public:
    void m() { .. }    
    virtual double c() { .. }    
    virtual char* t() { .. }    

} B;
Implementation of Inheritance (2)

class foo { ... }
class bar : public foo { ... }

foo F;
bar B;

foo *q;
bar *s;

q = &B;
s = &F;
class foo { ... }
class bar : public foo { ... }

foo F;
bar B;

foo *q;
bar *s;

q = &B;
s = &F;

Okay: References through q will use prefixes of B’s data space and vtable
Implementation of Inheritance (2)

class foo { ... }
class bar : public foo { ... }

foo F;
bar B;

foo *q;
bar *s;

q = &B;
s = &F;

Static semantic error: F lacks the additional data and vtable entries of a bar object