Introduction to Object Oriented Programming 2E Timothy A. Budd

Chapter 14

Polymorphism

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Definition

Polymorphous: Having, or assuming, various forms, characters, or styles

(from *Webster's Collegiate Dictionary, Fifth Edition*).

Term used in several different ways in computer science

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Polymorphism in Programming Languages

- A variable hold a value drawn from a group of types
- A name associated with several different function bodies
- A single function with polymorphic variables as arguments

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Polymorphism in Untyped Languages

Polymorphism is a trivial issue in untyped languages.

All variables are potentially polymorphic.

```
silly: x
    " a silly smalltalk polymorphic method "
    (x isKindOf: Integer) ifTrue: [ ^ x + 1 ].
    (x isKindOf: Fraction) ifTrue: [ ^ x reciprocal ].
    (x isKindOf: String) ifTrue: [ ^ x reversed ].
    ^ nil
```

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Forms of Polymorphism

- polymorphic variables
- overloading (ad hoc polymorphism)
- overriding
- deferred methods
- pure polymorphism (functions with polymorphic parameters)
- generics or templates

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Polymorphic Variables

Variable declared as one class can hold *value*s from subclass

- Dynamic (run-time) type must be subclass of static (declared) type
- In C++ only works with pointers and references
- Polymorphic variables (or values) basis for power of OOP

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Overloading

A single function *name* denoting two or more function *bodies* is *overloaded*

- Common example: + means both integer and floating addition

  (In Java, + also used for string concatenation)

- Determination of which function to execute made by examining arguments

- Depending upon language, either at compile-time or at run-time

- Should not be confused with overriding or refinement – functions need not be related by classes

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Overloading Example: Florist

```
Procedure Spouse.sendFlowersTo(anAddress : address);
begin
   go to florist;
   give florist message sendFlowersTo(anAddress);
end;

Procedure Florist.sendFlowersTo(anAddress : address);
begin
   if address is nearby then
      make up flower arrangement
      tell delivery person sendFlowersTo(anAddress);
   else
      look up florist near anAddress
      phone florist
      give florist message sendFlowersTo(anAddress)
end;
```

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Type Signatures

- Selection of which procedure to execute based on *type signature*
  - number, types, and order of parameters
- C++, Java allow functions in same scope to have same name if signatures differ
  - C++ allows extensive overloading of operator symbols

```
stream ≪ integer;
stream ≪ char;
stream ≪ char *;
stream ≪ double;
stream ≪ complex;
```

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

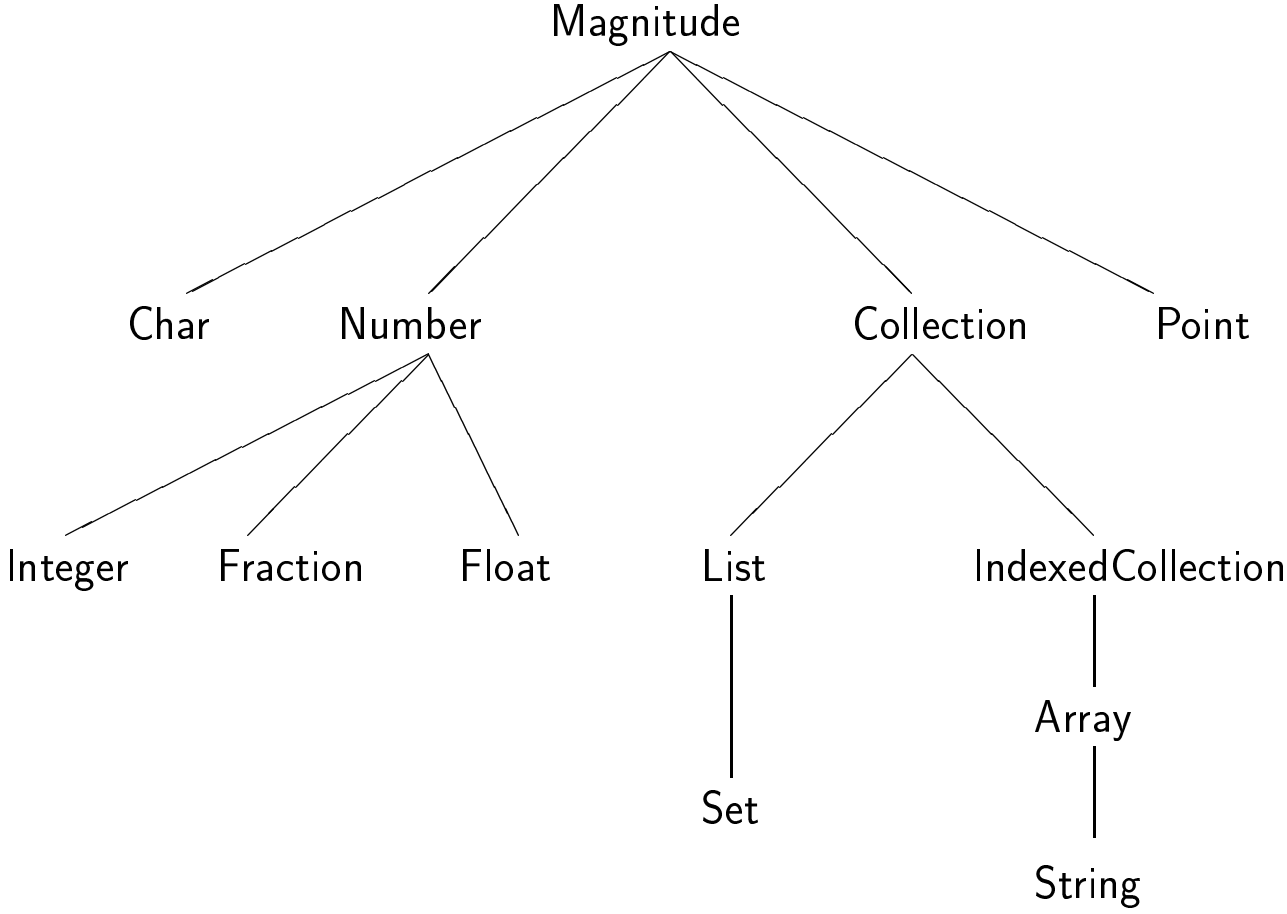
Overriding

Overriding occurs when child class changes meaning of function from parent

- Different child classes can override in different ways
- Parent class can have default behavior; child classes alternatives
- Contributes to code sharing
- Ensures common interfaces

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Subclasses of Magnitude in Little Smalltalk



© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Example – from class Magnitude

```
<= arg
    ↑ self < arg or: [ self = arg ]

>= arg
    ↑ arg <= self

< arg
    ↑ self <= arg and: [ self ~= arg ]

> arg
    ↑ arg < self
```

Overridden Methods

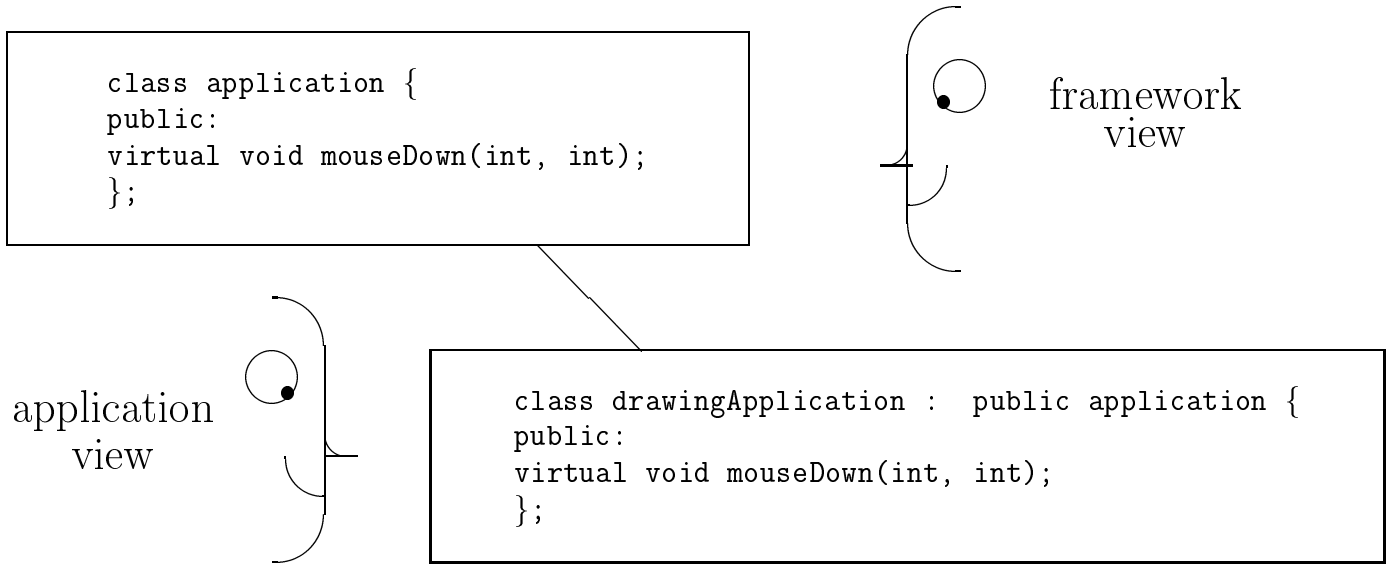
Child classes need only override *one* method (e.g., $<$) to get effect of all relationals

- Overridden in class Integer to mean integer less than
- Overridden in class Char to mean Ascii ordering sequence
- Overridden in class String to mean lexicographic ordering
- Overridden in class Point to mean lower-left quadrant

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Frameworks (from Chapter 18)

An important application of overriding is software frameworks, e.g., GUIs

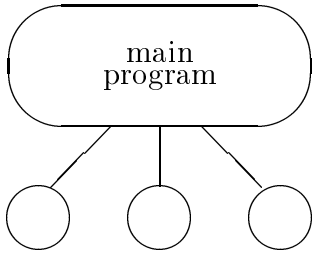


An Upside Down Library

Frameworks sometimes described as upside-down libraries

Role of user-defined code and library code reversed

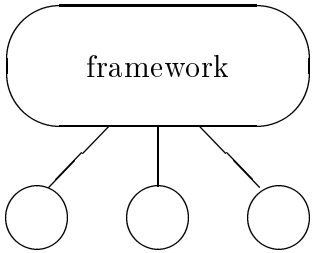
Conventional Library



user supplied code

library code

Framework



framework library

user supplied code

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Portable Algorithms

- A framework allows programmers to carry whole *algorithms* among projects
- High-level details shared
- Traditional libraries only permit sharing of low-level functions

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Deferred Methods

- Deferred method is special case of overriding
- Parent class gives type signature, but provides no implementation
- Child class *must* provide an implementation.
- Usually combined with other techniques.

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Deferred Method in C++

```
class Shape {
public:
    Point corner;
    void virtual draw() = 0;  // deferred member function
};

class Circle : public Shape {
public:
    int radius;
    void draw() { drawCircle(corner + radius, radius); }
};
```

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Pure Polymorphism

- Pure polymorphism usually means functions with polymorphic parameters
- Most methods in a framework use pure polymorphism

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Smalltalk Example

The between:and: method occurs in class Magnitude

Recall earlier discussion of relationals to see variety of uses

```
between: low and: high
    " test to see if the receiver is between two endpoints "
    ↑ (low <= self) and: [ self <= high ]
```

Generics and Templates

A *type* can be used as a *parameter* in class description

```
template <class T> class link {
public:
    T value;
    link<T> * nextLink;
};
```

The declaration of variable provides the actual type

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Polymorphism in Java

polymorphic variables? Yes

- Tree-structured subclass hierarchy using inheritance (extends)
- Subtype hierarchies using interfaces (implements)
- Variables declared with either class or interface as type
- Variables can hold value from subclass or interface subtype

overloading? Yes

- Overloading of method names (including in same class)
- No user-defined overloading of operator symbols

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Polymorphism in Java (continued)

overriding? Yes

– By default, all methods can be overridden

– Keyword final prevents overriding

deferred methods? Yes

– Keyword abstract for deferred methods, signature but no body

pure polymorphism? Yes

– dynamic binding of call to method, searching up subclass hierarchy

generics? No (perhaps in future)

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Efficiency and Polymorphism

- Programming always involves compromises
- Polymorphism is a compromise between efficiency and ease of development, use and maintenence
- Ask yourself: whose time is more important – yours or the computer's?

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)