Introduction to Object Oriented Programming 2E Timothy A. Budd

Chapter 10

Subclasses and Subtypes

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

Related Concepts

The following concepts are related, but are not exactly the same:

- Static versus Dynamic Typing – how variables and values are related
- Static versus Dynamic Classes – values a variable can maintain
- Static versus Dynamic Method Binding – how method invocation is understood
- Subclass versus Subtype – how a class can be used in place of another

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

Typing

- A *variable* or *identifier* is simply a name – a handle to access values
- A *value* is a string of bits – the current contents in computer memory
- In statically typed languages, *types* associated with variables (e.g., Pascal, C)
- In dynamically typed languages, types associated with values (e.g., Lisp, Snobol)

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

Error Detection with Static Typing

Statically typed languages permit type error detection at compile time

```
double r;

r= 'x';
```

Above results in a compiler error message

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

Polymorphic Variables

Object-Oriented concepts introduce a new twist:

- Instance of child class is in all important respects representative of parent
- Thus, reasonable to assign child instance to variable of parent class
- However, assigning parent instance to child class not generally permitted

```
class Dog extends Mammal { ... }
class Cat extends Mammal { ... }
...
Mammal pet;
Dog fido;
Cat felice;
pet  = fido;    // legal
pet  = felice;  // legal
fido = pet;     // illegal !
```

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

Static and Dynamic Classes

```
class Dog extends Mammal { ... }
class Cat extends Mammal { ... }
...
Mammal pet;
Dog fido;
Cat felice;
pet = fido;    // legal
```

- Class of declaration is the static class for the variable
- Class of value is the dynamic class of the variable
- Even statically typed OOP languages permit these to differ

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

Subclasses and Subtypes

- A subclass is a class formed using inheritance
- A subtype is a class that can be substituted (using polymorphism) with no observable effect
- Not all subclasses are subtypes
- In some languages (e.g., C++) can ONLY be formed using subclasses
- In other languages (e.g., Smalltalk, Objective-C) subclasses and subtypes are totally independent
- In Java subtypes formed using subclasses or using interfaces

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

Reverse Polymorphism

Polymorphism allows value from child class to be assigned to variable of parent class

Under what conditions can value of parent class be assigned to variable of child class?

```
class Dog extends Mammal { ... }
class Cat extends Mammal { ... }
...
Mammal pet;
Dog fido;
Cat felice;
pet = fido; // legal
fido = pet; // is this legal?
```

This is problem of *reverse polymorphism*

Two Aspects of Reverse Polymorphism

Problem of identity – whether a variable declared as parent class actually holds value from subclass

Task of assignment – assigning a value from parent class to a variable declared as a subclass

Some languages mechanisms address these two problems together

Other languages separate them

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

The Container Problem

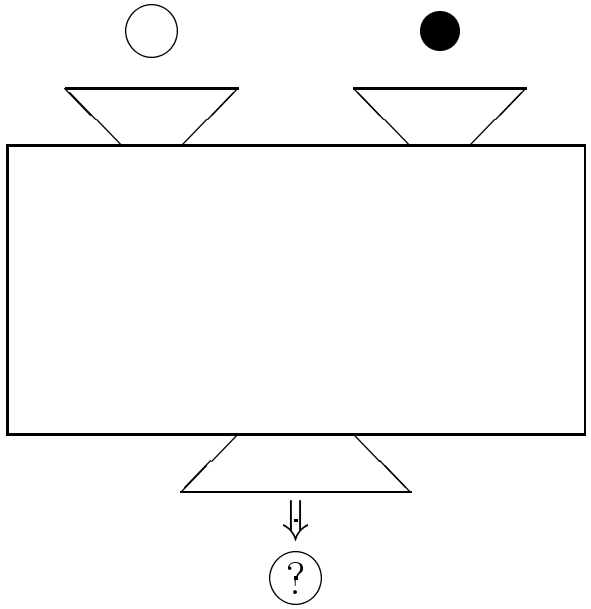
- Reverse polymorphism often encountered with collections of values

  – given list of items from parent class, determine whether extracted item is subclass instance

- Generally occurs in languages with single inheritance tree
- Solving problem generally requires values to "know" their own type

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

Can We Tell What Comes Out?



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

Static and Dynamic Method Binding

Should binding be on static class of variable or dynamic class of value?

- Alice holds small Mammal – asks Bill "does this animal give birth to live young?"

  Static answer: All mammals give birth to live young, therefore yes

- What if the Mammal is a platypus?

  Dynamic answer: Platypus lay eggs, therefore no

- Even statically typed OOP languages can use dynamic binding

  But may use static type to determine legality of operation

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

Merits of Static versus Dynamic Binding

Efficiency: static binding requires fewer CPU cycles

Error detection: static binding permits error detection at compile time

Flexibility: dynamic binding permits greater flexibility, static binding creates rigidity and inhibits reuse

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

Binding in Smalltalk

- Dynamically typed language
- All values polymorphic – maintain their own type information
- Legality of message passing not checked until run-time
  – can be tested directly by programmers if desired
- Reverse polymorphism not problem in dynamically typed languages

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

Binding in C++

- Instances of classes are statically typed, not polymorphic
  - pointers and references can hold polymorphic values
- Originally, values did not have any type information
  - now does have limited info with RTTI facilities
- Method binding can optionally be performed using dynamic class for pointers and references (for virtual functions)
- Legality of message passing checked at compile time based on static type
- Originally, no facilities for the testing of reverse polymorphism
  - now does have dynamic_cast and typeid
- Assignment of reverse polymorphic variables can be performed using cast or dynamic_cast

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

Binding in Java

- All variables know their dynamic type
- Reverse polymorphism performed with a cast
  - cast is *dynamically checked*.
- Can test the run-time type of a value (instanceof)
- No virtual keyword
- Unique feature: data fields can also be overwritten
- Subtype hierarchy can be formed using interfaces
  - independent of subclass hierarchy

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

Java Interfaces

- Can be implemented by classes
- Similar to classes, but have no implementation
- Support inheritance, even multiple inheritance
- Define subtypes – inheritance defines subclasses.

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