Lecture 17: Objects Continued

COMP 524 Programming Language Concepts
Stephen Olivier
April 7, 2009

Based on slides by A. Block, notes by N. Fisher, F. Hernandez-Campos, and D. Stotts

Binding

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

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

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

B

Static binding sets the type based on the declared type of the reference

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

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

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

B

Dynamic binding sets the type based on the type of the object referenced

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

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

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

Dynamic

- Java uses dynamic binding for all methods
- C++ uses static by default by allows a function to be dynamically linked as necessary.
  - virtual keyword specifies dynamic binding

```
class foo {
  ...
  virtual print_label ()
}
```

Abstract

- Abstract classes have at least one function not defined

```
abstract class person {... };
```

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

Abstract methods and classes

- Abstract classes have at least one function not defined

This is called a purely virtual method

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

Abstract Classes and Methods

- Java specifies an abstract method (not surprisingly) using the abstract keyword
  - abstract classes may or may not have abstract methods
- A class derived from abstract class must provide a body for abstract / pure virtual functions
  - Unless the derived class is also abstract...

Generics

- Generics allow abstracting over unrelated types

```
template<class V>
class list {
  list_note<V> header;
public:
...
}
```

Generics

- Generics allow abstracting over unrelated types
- Different flavors of polymorphism
  - Dynamic method binding provides subtype polymorphism
    - Create hierarchy by extending types
  - Generics provide explicit parametric polymorphism
    - Abstract over types
- Can be used together

Multiple Inheritance

- C++ allows a class to be derived from more than one parent class:

```
class professor : public teacher,
                  public researcher {
 ... }
```

- What happens if `teacher` and `researcher` both have a `print()` method?
  - Could use scope resolution operator: `teacher::print()`
  - Ambiguous call to `print()` disallowed by compiler

Mix-in Inheritance

- This is a restricted form of multiple inheritance
- Consider the variant used in Java
  - One “real” parent class from which data members and non-virtual methods may be inherited
  - Arbitrary number of interfaces specifying only pure virtual methods and (possibly) static data members
- Much easier to implement that full-blown multiple inheritance

Smalltalk Basics

- Everything is an object (even numbers)
- Get things done by sending messages to objects
  - To add 3 + 4, send the object 3 the message + with the argument 4. The result is a reference to the object 7.
- Can provide multiple arguments with “mix-fix”:

```
myBox displayOn: myScreen at: location
```

  - Here the message is displayOn: at: and the two arguments are myScreen and location

Smalltalk Conditionals

- Even selection is done by sending message to objects

```
n < 0
  ifTrue: [abs <- n negated]
  ifFalse: [abs <- n]
```

- "< 0" message sent to n
- Resulting reference is sent arguments that are blocks
- Special value message sent back to selected block

Smalltalk Iteration

- Yep, also by sending messages to objects

```
sum <- 0.
1 to: 100 by: 2 do:
[:i | sum <- sum + (a at: i)]
```

- Code above sums up odd-indexed elements of an array

Smalltalk Closures

- Since code blocks are objects, we can have references to them:

```
b <- [n <- n + 1].
```

- This reference represents the Smalltalk version of a closure