Object orientation

Amer Diwan

Main components of object-orientation?

• Encapsulation
  – Group data and associated operations
• Information hiding
  – Need to know principle!
• Inheritance
Inheritance

- **TYPE Shape = OBJECT**
  
  `color: Colors;`  
  
  **END;**

- **TYPE Circle = Shape OBJECT**
  
  `radius: INTEGER;`  
  
  **END;**

One way to think about inheritance

- An object contains
  - All the fields of its supertype, followed by all the fields declared in the object
  - (All the methods of the supertype followed by all the methods in the object)

- What’s a subtype?
What do they look like in memory?

- Run time representation needs
  - space for data
  - (mapping of methods)
- What do Shape and Circle objects look like?
- How are they different from records?

Another example

- **TYPE FilledCircle = Circle OBJECT**
  
  *fill_color: Colors;*

  **END;**

- A FilledCircle is an object with
  - all the fields of a shape, followed by
  - all the fields of a circle, followed by
  - all the fields declared in FilledCircle

- FilledCircle <: Circle <: Shape
Inheritance and subtyping

- Is a FilledCircle bigger or smaller than a Circle?
- Does this make sense from a subtyping point of view?

An example

- \( T = \text{OBJECT} \ldots \text{END}; \)
- \( S = T \text{OBJECT} \ldots \text{END}; \)
- \( x: \text{REFANY}; \)
- \( x := \text{NEW}(S); \)
- \( \text{NARROW}(x, T) \)
- What representation does this need?
Another example

• \( l: \text{ShapeList.T}; \)
  \( l := \text{ShapeList.Cons} (\text{NEW}(\text{Circle}), \ l) \)
  \( l := \text{ShapeList.Cons} (\text{NEW}(\text{FilledCircle}), \ l) \)
  \( l := \text{ShapeList.Cons} (\text{NEW}(\text{Square}), \ l) \)
• Is this legal?

Continuing example

• \( \text{WHILE} \ l.\text{tail} \neq \text{NIL} \ \text{DO} \)
  \( l.\text{head}.\text{radius} := 10 \)
  \( l := l.\text{tail}; \)
  END
• \( \text{WHILE} \ l.\text{tail} \neq \text{NIL} \ \text{DO} \)
  \( \text{IF} \ l.\text{head}.\text{color} = \text{Colors.White} \ \text{THEN} \)
    \( \ldots \)
  END;
  \( l := l.\text{tail}; \)
  END;
Continuing example

- **PROCEDURE** \( f(s: \text{Shape}) = \)**
  
  ```plaintext
  IF s.color ≠ \text{Colors.White} THEN
  ...
  END
  ```

Polymorphism

- **Contrast with** \textit{monomorphic}:  
  - Functions and their operands have a unique type  
  - Every value and variable can be interpreted to be of one and only one type

- **More directly**:  
  - Functions work uniformly on a range of operand types  
  - Some values and variables may have more than one type
Polymorphism versus monomorphism

• Expressiveness

• Simplicity

• Ease of implementation

• Efficiency

Adding methods to objects: support for encapsulation

• TYPE Shape = OBJECT
  color: Colors;
  METHODS
    setColor(to: Color) := ...
  END;
• TYPE Circle = Shape OBJECT
  radius: INTEGER;
  END;
Let’s look at the guts of a method

- **TYPE Shape = OBJECT**
  - color: Colors;
- **METHODS**
  - setColor(to: Color) := setColorShape;
- **END;**
- **PROCEDURE setColorShape(to: Color) =**
  - ...
- How does setColorShape access the “color” instance variable?
Two approaches

• Java, C++, Smalltalk:
  – pass implicit parameter called “this” or “self”
• Modula-3:
  – s.setColor(c) => setColorShape(s, c)

Overriding methods

• Overriding gives a new implementation for an existing method
• TYPE Shape = OBJECT
  color: Colors;
  METHODS render(); END;
  TYPE Circle = Shape OBJECT
    radius: INTEGER;
  OVERRIDES render := ...; END;
  TYPE Square = Shape OBJECT
    side: INTEGER;
  OVERRIDES render := ...; END;
Representation

- Each type has its own mapping of methods to implementations
- The correct implementation is picked at run time!

Some implications of inheritance

- WHILE l.tail ≠ NIL DO
  f(l.head)
  l := l.tail;
END;
- PROCEDURE f(s: Shape) =
  s.render();
- What method is executed here?
How to implement method dispatch?

• Assumptions:
  – Static typing
  – Single inheritance (we will look at multiple inheritance later)

V-Tables

• Idea:
  – Prepend the methods of a supertype to a subtype
  – A method $T::m$ appears in the same position in all $T$’s subclasses

```
T [f,g]
 U [h]
 W [j]

V [i]
 X [k]
```

```
T [f0,g:1]
 U [f0,g:1,h:2]
 V [f0,g:1,i:2]
 W [f0,g:1,i:2,j:3]
 X [f0,g:1,i:2,k:3]
```
V-tables (cont.)

Construct a v-table for each class

V-tables are typically part of the type descriptor

V-tables(cont.)

\[ t \rightarrow g() \] becomes
\[ \text{vp} = t \rightarrow \text{vtable_ptr} \]
\[ \text{gaddr} = *(\text{vp} + \text{g's offset}) \]
\[ (*\text{gaddr})() \]
More examples

- \( t \) : Shape;
  \( t := \text{NEW Circle}; \)
  \( t.\text{render}(); \)
- \( t \) : Circle;
  \( t := \text{NEW Circle}; \)
  \( t.\text{render}(); \)
  \( \text{NARROW}(t, \text{Shape}).\text{render}(); \)

Overriding versus redefining

- TYPE Shape = OBJECT
  color: Colors;
  METHODS render(); END;
  TYPE Circle = Shape OBJECT
  radius: INTEGER;
  METHODS render := ...; END;
  TYPE Square = Shape OBJECT
  side: INTEGER;
  METHODS render := ...; END;
Back to our old examples

• WHILE l.tail # NIL DO
  f(l.head)
END;

• PROCEDURE f(s: Shape) =
  s.render();

Back to our old examples (cont.)

• t: Shape;
  t := NEW Circle;
  t.render();

• t: Circle;
  t := NEW Circle;
  t.render();
  NARROW(t, Shape).render();
Contravariance and inheritance

• Issues with
  – The “id” function
  – The “clone” method

Next topic: Case study

• Reading: Read up about objects in Java in your favorite Java book