Lecture 21: OO Languages

CSC 131
Spring, 2019

Kim Bruce

OO Keywords

- Object
- Message
- Class
- Instance
- Method
- Subtype
- Subclass

Objects

- Internal data abstractions
- Hide representation
- Have associated state
- Methods have access to its state
- Self

Object Types

- Allow objects to be first class
- Allow use in assignment, parameters, components of structures
- Allow objects to be classified via subtyping

Classes

- Templates for creation of objects
  - Initialization code
  - Contain definitions for all methods
- Can modify or extend by creating subclasses.
- Can be used as types

Dynamic Method Invocation

- Each object responsible for keeping track of implementation of its own operations
- When evaluate o.m(...) at run-time, code run depends on operations associated with o.
- Static overloading is different
  - resolved at compile time.

Multi-methods

- Code executed depends on more than one argument.
- Example m(a,b) -- choice of code to be executed depends on run-time types of a and b.
- CLOS is example of such language
- Behavior and implementation quite different from single-dispatch languages.

Subtyping

- Already discussed
- Relation between interfaces, independent of implementations

Subclasses

Support incremental modification of code.

```
class Point
   var
      x = 0: Int;
   methods
      fun getx():int {return x}
      proc move(dx: int)
          {x := x + dx}
end class;
```

Subclasses

```
subclass Colorpoint of Point
                      modifying move
   var
      color = blue: ClrType
   methods
      fun getColor():ClrType {return color}
      proc setColor(nuColor: ClrType)
           {color := nuColor}
      proc move(dx: int) {super.move(dx);
                          color := red}
end class;
```

Static Overloading vs Dynamic Dispatch

- Dynamic dispatch - object receiving message determines which code will be executed.
  - *Determined at run-time.*
- Static overloading occurs when an object supports two or more implementations of a message name -- generally w/different types.
  - *Determined at compile-time.*
  - *e.g.,* moveTo(x,y) *&* moveTo(locn)
- *Confusion* when coexist in same language.

Overloading vs Dynamic Dispatch

```
class C { …
  fun eq(other: Ctype):boolean {…}  (1)
}
class SC of C modifying eq { …
  fun eq(other: Ctype):boolean {…}  (2) override
  fun eq(other: SCtype):boolean {…} (3) overload
}

c,c′: Ctype; sc: SCtype;

c = new C;        c′ = new SC;      sc = new SC;
```

What code is executed?

```
c.eq(c);          c.eq(c′);         c.eq(sc);
c′.eq(c);         c′.eq(c′);        c′.eq(sc);
sc.eq(c);         sc.eq(c′);        sc.eq(sc);
```

Type Restrictions

Type systems limit changes in method types in subclasses.

```
class C
      ...
   methods
      clone(): CType {...};
      equals(other: CType) {...};
end class
```

How can we change these types in subclasses?

More flexible subclasses

Why restrict changing types in subclasses?

Methods can be mutually recursive!

```
class Example
      :
  methods
     proc m(s:S,...) {... self.n(s) ...}
     fun n(x: S): T {...}
end class;

subclass SubExample of Example modifying n
      :
  methods
    fun n(x: S'): T' {...}
    proc newMethod(...){...}
end class;
```

What must be relationship of new type of n to old to preserve type safety?

Changing Types in Subclasses

Subtype will always be fine!

I.e., $S <: S'$ and $T' <: T$ equivalently, $S' \to T' <: S \to T$

E.g., can change return type of clone in subclass to type of objects from subclass.

Cannot specialize parameter types in equals

Binary methods

If subclass updates method types so they are subtypes of original then type-safe.

If restricted in this way, subclasses will always generate subtypes.

What about instance variables?

Instance variables can be values and receivers

- No subtypes!

If Circle has instance variable center: Point, ColorCircle's center must have same type.

Hard to redefine getCenter in ColorCircle, even if legal!

Important problem in OO language design and type theory

OO Languages

- Simula 67
- Smalltalk-72, -74, ... -80
- C++, Object Pascal, Object Cobol, ...
- Eiffel, Sather
- Java
- Scala
- *Dart, Grace?*

Simula 67

Simula 67

- First OO language
- *You* read in text
- Also added coroutines
- Use of "inner" rather than "super" in constructors

Inner

```
Class A; Begin startA; Inner; endA End;
A Class B; Begin startB Inner; endB End;
B Class C; Begin startC Inner; endC End;
Ref(C) X;

X :- New C;
```

- Results in execution of:

  ```
  startA startB startC endC endB endA
  ```

- Beta supports similar in all methods & classes

Smalltalk

Smalltalk

- New features:
  - Everything is an object, including classes
  - No operations -- only message-sending
  - Used to build customizable environment
  - Abstraction -- private instance variables, public methods
- Dynamically typed

Dynabook

- Laptop computer -- Alan Kay 1970's
  - Turing award 2003
- Proposed in 1970's - aimed at children & adults
  - Neal Stephenson's "The Diamond Age or, a young lady's illustrated primer" is the next step
- Programmable environment
- Smalltalk as OS and programming language

Syntax

- `n <- 3+4`
  - send "+" message to 3 w/param 4 and insert in n
- `n between: 10 and: 100`
  - send "between: and:" message to n w/params 10, 100
- `[ :params | <message-expressions> ]`
  - lexical closure - equiv to lambda expression
  - positiveAmounts :=
    allAmounts select: [:amt | amt isPositive]

Smalltalk class

```
class name                Point
super class               Object
class var
instance var                x   y
class messages and methods
!...names and code for methods..."
 instance messages and methods
moveDx: dx Dy: dy ||
   x <- dx+x
   y <- dy + y
x
 ^ x
...
```

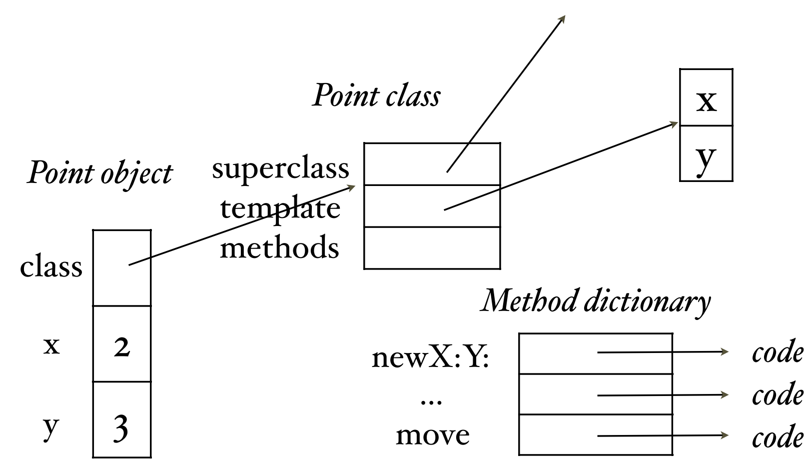
Commands

- Loops example:

```
1 to:10 do:[:i|
  Transcript show: (i asString).
].
```

- Conditional
  - `(x>0) ifTrue:[ x:=x+1. ] ifFalse:[ x:=0 ].`
  - `true and false are special values like lambda calculus encodings`

Run-time representations



Dynamic Method Invocation

- Start with object's class and search up superclasses.
- When call method inside, start search from self again.
- *Most other OO languages do not implement dmi in this way -- too inefficient!*

Key ideas of Smalltalk

- Everything is an object
- Information hiding - instance variables protected.
- Dynamic typing, so subtyping determined by whether can masquerade -- "message not understood"
- Inheritance distinct from subtyping

Smalltalk

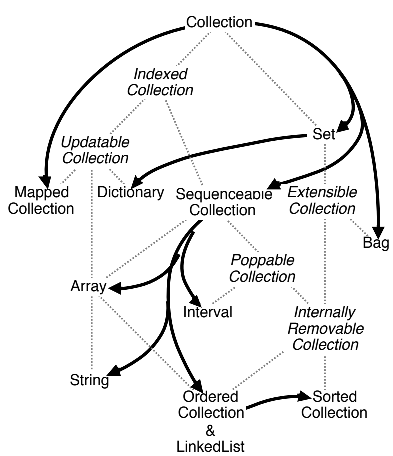


Figure 5: Interfaces versus Inheritance

C++