Objects and Classes

March 28, 2001
CS 131: Programming Languages

Questions to Think About

• What is an object?
• What is a class?
• What is a method?
• What makes a language object-oriented?

Extensibility

• Frequently a problem domain consists of
  – A collection of sorts of data item
  – Operations on data items

• Extensibility can be a problem.
  – How hard is it to add new sorts of data?
  – How hard is it to add new operations?

SML vs. OOP approaches

• Example:
  – Geometric shapes (circles, squares, ...)
  – Operations on shapes (translate, rotate, scale, ...)

• SML
  – Create a single datatype shape
  – Operations pattern-match on the sort of shape

• OOP
  – Create subclass of shape class for each shape
  – Operations become methods of the class
Prototypical Class

class Cell is
  field contents : int = 0;
  method get() : int is
    return this.contents;
  end;
  method set(n:int) is
    this.contents := n;
  end;
end;

Prototypical Usage of a Class

mycell = new Cell;
mycell.contents := 7;
mycell.set(12);
x = mycell.get();

procedure double(c:Cell) is
  c.set(2 * c.get());
end;
double(mycell);

Prototypical Subclass

subclass ReCell of Cell is
  field backup : int = 0
  override set(n:int) is
    this.backup := this.contents;
    super.set(n);
  end;
  method restore() is
    this.contents := this.backup;
  end;
end;

Prototypical Usage of a Subclass

mycell = new Cell;
myrecell = new ReCell;

procedure f(x:Cell) is
  x.set(3);
end;
f(mycell);
f(myrecell);
(dynamic dispatch)
Subclassing vs. Subtyping

Is it safe to conclude that every circle is a shape?

```java
class shape is
    ...
    method clone() : shape;
end;
subclass circle of shape is
    override clone() : circle;
end;
```

Subclassing vs. Subtyping

Assume Vegetables $\preceq$ Food. Is it safe to conclude that every vegetarian is a person?

```java
class person is
    ...
    method eat(food : Food);
end;
subclass vegetarian of person is
    override eat(food : Vegetables);
end;
```

Subclassing vs. Subtyping

Assume Vegetables $\preceq$ Food. Is it safe to conclude that every omnivorous cow is a cow?

```java
class cow is
    ...
    method eat(food : Vegetables) {
        ...
    end;
subclass omnivorouscow of cow is
    override eat(food : Food) {
        ...
    end;
```

Subclassing vs. Subtyping

Is it safe to conclude that every colored point is a point?

```java
class point is
    field x : int = 0;
    method equal(other : point) : bool {
        ...
    end;
subclass cpoint of point is
    field c : color = red;
    overload equal(other : cpoint) : bool end;
```
Subclassing vs. Subtyping

Example usage

```plaintext
function f(p1:point, p2:point):bool {
    return p1.equal(p2);
}
p1 = new point;
p2 = new point;
x1 = f(p1,p2);
cp1 = new cpoint;
cp2 = new cpoint;
x2 = f(cp1,cp2);
```

Subclassing vs. Subtyping

Now, is it safe to conclude that every colored point is a point?

```plaintext
class point is
    field x: int = 0;
    method equal(other : point) : bool ...
end;
subclass cpoint of point is
    field c: color = red;
    override equal(other : cpoint) : bool end;
```

Subclassing vs. Subtyping

• Summary: When overriding methods it’s safe to
  – Make the arguments of methods less specific
  – Make the return types of methods more specific

• Java permits neither
• C++ permits only the latter.
• Eiffel permits "wrong" overriding.
What Do Classes Provide?

- Abstract types
  - Plus ways to create values of this type (constructors)
- Subtyping hierarchy
- Inheritance (code reuse)
- Modularity?
  - Collects related code
  - Classes with only static methods awfully like modules
  - Visibility restrictions (public, private, etc.)
  - But what do you need to know to use a class?

Mixins

- Some people argue for a finer-grained notion of inheritance than classes.
- A mixin is a collection of facilities that we would like to add to several classes.
  - Sort of a partial class
  - Can create a whole class by combining mixins
  - Can be simulated in C++ with templates or multiple inheritance

```cpp
template <class Super>
class Mixin : public Super {
  ...
}
```

Mixin Example [Booch]

```java
class Rose: public Plant, public FlowerMixin ...
class Carrot: public Plant, public FruitVegetableMixin ...
class Cherry: public Plant, public FruitVegetableMixin, public FlowerMixin ...
```

Is SML Object-Oriented?

```sml
fun cell() =
  let
    val r = ref 0
  in
    {contents = r,
     get = (fn () => !r),
     set = (fn n => (r := n))}
  end
val mycell = cell()
val x = (#set cell)((#get cell()) * 2)
```
Object-Based Languages

• Some object-oriented languages don't have a built-in notion of class.
  – Objects are just ordinary values, like integers or pairs in SML
  – Objects are created by:
    • Listing all the values for the fields and methods
    • Or, adding new fields and methods to an empty object. (Usually, but not always, untyped.)

Example: JavaScript

```javascript
function cell_get() {
  return this.contents;
}
function cell_set(n) {
  this.contents = n;
}
var mycell = new Object();
mymcell.contents = 0;
mymcell.get = cell_get;
mymcell.set = cell_set;
```

Example: JavaScript

```javascript
function Cell {
  this.contents = 0;
  this.get = cell_get;
  this.set = cell_set;
}
var mycell = new Cell();
```

Delegation

• Subclassing without classes
• Each object may have a "parent"
  – If we fail to find a field or method in an object, try looking in the object's parent, the parent's parent, etc.
  – Efficiency: many objects can share the same parent, so they don't have to each have a copy of the parent's fields and methods.
  – Parent can be changed at run-time
    • Nice implementation for mode-switching (e.g., a shape's drawing operations must work for iconified and open modes.)
CLOS: Common Lisp Object System

• Class-based, but classes specify only fields ("slots")

(defclass figure ())
(defclass circle (figure)
  ((center :initform 0)
   (radius :initform 1)))
(defclass square (figure)
  ((lowerleft :initform (0 0))
   (upperright initform (1 1)))

CLOS

• Q: If objects contain only fields, what’s the point?
• A: Dynamic dispatch via multimethods ("generic functions")
  – Functions where we can keep adding new cases
  – True dynamic dispatch, not just overloading!

(defmethod draw ((f circle) window)
  ...draw the circle f in the given window...)
(defmethod draw ((f square) window)
  ...draw the square f in the given window...)

Multimethods

• C++ or Java are called single-dispatch languages because the code invoked depends on the class of a single object: mycell.set(3)
• CLOS supports multiple-dispatch.

(defmethod intersect ((c1 circle) (c2 circle))
  ...find the intersection of circles f1 and f2...)
(defmethod intersect ((c circle) (f figure))
  ...find the intersection of circle c and arbitrary figure f...)