CSE 3302 Notes 10: Object-Orientation, Polymorphism, and Generic Programming

(Last updated 11/23/15 10:46 AM)

Influences in the O-O "revolution"

Logic/Mathematics - type concepts, notation, lambda calculus
Software Engineering - need for reuse, early detection of errors
Database - need for more than tables (and "semantic")
AI/Knowledge Representation - higher-order programming
User Interfaces/Graphics
Simulation

Who was first?

10.1. Limits of Abstract Data Types

Examples of a counter :

(p. 278) `get`, `inc`, `reset` ADT
(p. 278-279) `howmany_resets` ADT (new ADT or nest original?)
Array including both types of counters? (p. 280)

What is needed? *Dynamic binding* (and method selection), but also

Encapulation and type checking (like ADTs)
Inheritance

10.2. Fundamental Concepts

Object = Data + Operations (for JavaScript, both are *properties*)

Class = (an often unnavigable) set of objects (*instantiations*)

Constructor, `this`, `public`, `private`

JavaScript arrays are actually associative arrays, typically implemented by hashing a string:

`arr[1]` and `arr["1"]` refer to the same *property* of object `arr`

`arr[5/2]` and `arr["2.5"]` refer to the same property of object `arr`

BUT, an object is an `array` only if the object was created using either of these:

```
arr=['donut','chips',{a:1,b:'cat'},undefined];

arr=new Array('donut','chips',{a:1,b:'cat'},undefined);
```

For any object, `object.property` and `object["property"]` are available, BUT . . .

`.property` possibilities are more limited than `["property"]` possibilities

For an array, `length` is one more than the maximum positive integer property name

Set operations may be based on: `for . . . in` as: *loop* (p. 24 of *The Good Parts*), along with: string/index `in` object as *operator* returning `true`/`false` to indicate property presence

`delete x[propName]` removes the property

See `http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES10/objSet.html` for manipulation of properties (on simple objects, no prototypes/inheritance, no methods)

JavaScript provides *prototypal inheritance* as a simple *delegation* mechanism:

"In JavaScript, a class is a set of objects that inherit properties from the same prototype object." (D. Flanagan, *JavaScript: The Definitive Guide*, chapter 9, along with 6 and 8)

Several ways to set the *prototype* for an object:

`Object.create()` is usually the simplest

Constructor used with `new` can lead to difficulties (see Crockford's book and webpage)

(Browser dependent techniques, including changing the prototype)

`objectName.propertyName = ... ;` can only set (l-value) the property value on `objectName`
`... = ... objectName.propertyName ... ;` searches the prototype chain (r-value)

`delete` will not follow the prototype chain - it will only remove property from provided object:

`delete` *object-reference*`[`*property-name-as-string*`]`
`delete` *object-reference*`.`*property-name*

object-reference`.has_property(`*property-name-as-string*`])` is the way to check presence of a property locally (without following prototype chain) before `delete`

(Aside: using `Object.getPrototypeOf()` to navigate prototype chain directly)

Examples:

> `http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES10/objArray.html` demonstrates simple data values as properties
>
> `http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES10/quo.html` demonstrates use of `new`
>
> `http://ranger.uta.edu/~weems/NOTES3302/LAB2SUM13/` demonstrates prototypal inheritance by having instances inherit from "class object" with needed methods. In addition, boxes may inherit drawing properties (colors for fill and stroke, thickness for stroke) from containing box

To implement a class hierarchy:

1. Root class is a simple object with class variables and functions.
2. Subclass is initiated by subClassName=Object.create(superClassName).
3. New methods and class variables are added as properties.
4. Instances of a class are initiated by instanceRef=Object.create(className).
5. Possible to encapsulate private members, e.g. using closures.

`http://ranger.uta.edu/~weems/NOTES3302/LAB/15SUM/LAB4/`

When is a subclass also a subtype?

Abstract class - cannot have an instance

(asides: C++ uses `=0` to make a virtual function "pure", Java has keyword `abstract`)

Many subtyping issues become apparent when mutability is considered (book examples)

10.3. IMPLEMENTATION ASPECTS

10.4. POLYMORPHISM AND GENERICS

W.R. COOK PAPER - section 3

O-O is about:

Functions, their application, and interfaces (figure 8)

Dynamic binding

Public interfaces (autognosis, 3.3)

Flexibility to represent infinite sets (3.4 and `http://ranger.uta.edu/~weems/NOTES3302/LAB1FALL13/`)

Bisimulation/equivalence (3.7)

Flexibility/duality (4.3)

ADTs - adding operations

O-O - adding representations

C++ Templates, Lambda Calculus, and Compile-Time Turing Computability (aside)

`http://matt.might.net/articles/c++-template-meta-programming-with-lambda-calculus/`