Prototype-based object-oriented programming

Lalginar, 21. april 2011

Jure Mihelič

Literature

- Antero Taivalsaari
  - Classes vs. prototypes – some philosophical and historical observations, 1996.
  - On the notion of inheritance, ACM Comp. Surveys, 1996
- Iain D. Craig, Object Oriented Programming Languages: Interpretation.
- JavaScript, IO, Lua, and Self tutorials.
- Wikipedia and internet.

Philosophy

- *Plato*.
  - **Forms**:
    - stable, immutable, „ideal“ description of things;
    - world of ideas.
  - **Things**:
    - instances of forms;
    - world of instances.
  - Forms have existence that is more real than the concrete entities and beings in the real world.

Philosophy

- *Aristotle.*
  - Taxonomy of all natural things – plants, animals, minerals, etc.
  - **Objects** belong to the same **category** if they have the same properties.
  - **Categories** of objects are defined by common properties.
  - **New categories** defined from existing ones:
    - *essence* = *genus* + *differentia*.

Philosophy

- Common idea (at least in the West).
  - There is a single correct taxonomy of natural things.
- Aristotle noted that many objects have „accidental“ properties.
- The level of categorization depends on who is doing categorization and on what basis.

Philosophy

- *W. Whewell, W. S. Jevons, 19th century.*
  - There are no universal rules to determine what properties to use as the basis of classification.
  - Classification in not a mechanical process but requires creative invention and evaluation.

Philosophy

- *Ludwig Wittgenstein*, 1950s.
  - Gave several examples of seemingly simple concepts that are extremely difficult to define in terms of shared properties, e.g. game, work of art.
  - Defined notion of „family resemblance“.
  - Meaning of a (non-mathematical) concept is determined not by definition, but by family resemblances.

Philosophy

- Family resemblances.
  - Things are connected by a series of **overlapping similarities**, where **no** one **property** is common to all.
  - Games do not have any shared, common defining characteristics.
  - Similar for some properties, but different for the others.

Philosophy

- *Eleanor Rosch*, mid-1970s.
  - Categories defined by properties:
    - no member is a better example of the category than any other member.
  - Categories defined by family resemblances:
    - categories, in general, have **best examples**.

Philosophy

- *Aristotelian* view – Class-based OOP.
  - Has limited modelling capabilities.
    - Usually good enough for real problems.
  - No optimum class hierarchies.
    - Consensus-driven design and „good-enough“ models.
  - „Basic“ classes and the need for iteration.
    - Basic classes are usually found first, more general and more specific are deduced later.
    - The basic classes usually end up in the middle for the class hierarchy.

Philosophy

- *Wittgenstein*'s view – Prototype-based OOP.
  - There are no classes at all.
    - All programming is done in terms of objects, often reffered to as **prototypes**.
  - A prototype is a representative example of a concept.
    - Classification based on similarity.
  - Mutable objects.
    - It is possible to add or remove attributes and methods at the level of objects.

Class-based OOP

- Classes and instances.
- Concepts.
  - Data abstraction.
  - Encapsulation.
  - Message passing.
  - Modularity.
  - Polymorphism.
  - Inheritance.

Prototype-based OOP

- Object is a collection of slots.
- Slot.
  - Contains a value or a method.
  - *name* = *value*
  - Slots can be constant/mutable, private/public, etc.
- Behaviour reuse.
  - Performed by cloning existing objects.

Object creation

- Fresh objects (*ex nihilo*).
  - Creating completely new objects by listing slots.
- Cloning / copying.
  - By copying existing objects.
  - Shallow vs. deep copy.
  - Parent-child relationship.

Concatenation

- Concatenation model.
  - No (visible) links to the original prototype object.
  - Prototype is copied exactly, but given different name or reference.
  - Object copies are independent.
    - Copy can be altered without side-effects
      - across other children of the parent or across children.
    - Propagating changes is difficult.
      - Additional primitives are provided in some languages (Kevo).
  - Naïve implementations waste a lot of memory.

Delegation

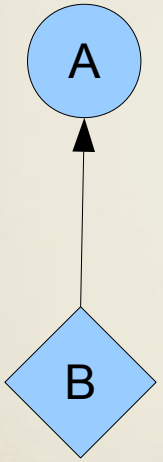
- Delegation model.
  - Based on parent-child relationship.
  - Method dispatching is based on following the delegation pointers.
    - Delegating to parents.
  - Child can be altered without side-effects to parents.
  - Parent alteration has side-effects to children.

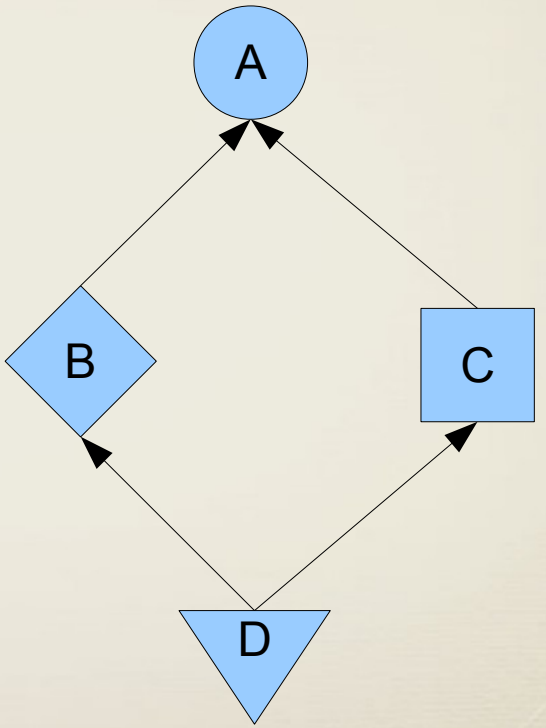
Delegation.

- Slot request.
  - If the object does contain the slot, the slot is used.
  - If the object does not contain the slot, the request is delegated to the object's parents.
  - If the immediate parent does not contain the slot, the request is delegated to the parent's parents.
  - Etc.

Delegation.

- Single parent vs. multiple parents.





Delegation.

- Dynamic / computed delegation.
  - Parents can be changed or computed during runtime.

Disadvantages

- Static vs. dynamic type system.
  - Most PBOOPL have dynamic type systems.
  - Same concerns: correctness, safety, predictability, efficiency, etc.
- Unfamiliarity.

Prototype-based OOP languages

- Self (the first one)
- Newton, Lisaac, Rebol, Cecil, Neko, R, Slate,
- Omega (static typing)
- ECMAScript (Javascript)
- Kevo (concatenation)
- Lua, IO
- Rox
- Etc.

Self

- *David Ungar*, *Randall Smith*, 1987.
- Based on Smalltalk.
- Object creation by cloning.
- Multiple dynamic delegation.
  - One or more slots can indicate *parent* objects.
- Trait objects - traits.
  - An object used as a parent for other objects.

Lua

- *Roberto Ierusalimschy, et.al.*, 1993.
- Tables.
  - Table is an array and a dictionary at the same time.
- Objects are tables.

```
Vector = {}                     -- Create a table to hold the class methods
function Vector:new(x, y, z)  -- The constructor
  local object = { x = x, y = y, z = z }
  setmetatable(object, { __index = Vector })  -- Inheritance
  return object
end
function Vector:magnitude()   -- Another member function
  -- Reference the implicit object using self
  return math.sqrt(self.x^2 + self.y^2 + self.z^2)
end

vec = Vector:new(0, 1, 0)     -- Create a vector
print(vec:magnitude())        -- Call a member function using ":"
print(vec.x)
```

JavaScript (ECMAScript)

- *Brendan Eich*, 1995.
- Objects are associative arrays.

Io

- *Steve Dekorte*, 2002.
- Based on Smalltalk, Self, NewtonScript, Act1, LISP, Lua.

```
Account := Object clone do(
    balance := 0
    deposit  := method(v, balance = balance + v)
    withdraw := method(v, balance = balance - v)
    show := method(writeln("Account balance: $", balance))
)

myAccount := Account clone
myAccount show
"Depositing $10\n" print
myAccount deposit(10)
myAccount show
```

Rox

- Object creation.
  - Ex nihilo.
  - Cloning and delegation.
    - Currently: single parent, planned: multiple parents.
- Functional programming.
  - First class functions, closures, matching.
- Everything is an object / expression.
  - Even suites/code, statements, etc.