IOOR/ID2017

Lecture 2

Class-based Languages, Prototype-based Languages and Actor Languages

Abstraction – an example

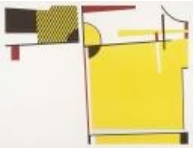
Bull I

Bull II

Bull III

Bull IV

Bull V



Bull VI

Roy Lichtenstein (American, 1923-1997), the six prints in the "Bull Profile Series,"

Abstraction – fundamental in Computer Science

- Abstraction in Computer Science often implies simplification:
  - the replacement of a complex and detailed real-world situation by an understandable model within which we can solve the problem.
  - we only want to consider the parts of reality that are important for the things we want our system to handle
  - we want to avoid everything else since it would only clutter things up and make the more important things harder to see
- Abstract doesn't mean imprecise

The Level of Abstraction Has Already Risen

- Wires
- Machine code
- Assembly Languages
- System Languages
- OOPLs
- What will be the next step?

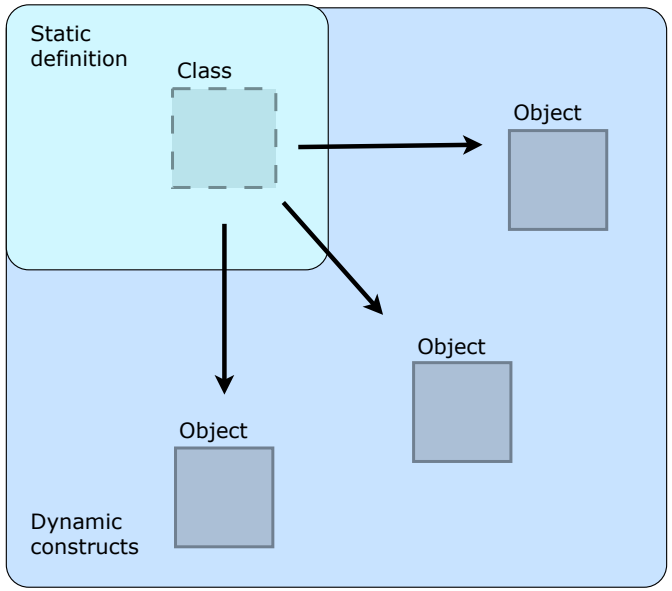
Abstraction in OO

"The essence of abstraction is to extract essential properties while omitting inessential details."

[Ross et al, 1975]

- An object is the realisation of some domain concept in a program
- A high level of abstraction
  - makes an object simpler to manage than a collection of procedures and data structures
  - facilitates use and reuse
  - facilitates changes to the implementation of a concept without affecting the rest of the program

Class-based



What is a class?

Common Answers

- A blueprint for creating objects [Sun's Java Tuturial, Eilëns00]
- A description of the shared behaviour or a special class of objects (or values)
- A description of the structure of a set of objects [Abadi & Cardelli96]
- A unifying abstraction of a set of values in the domain
- A factory for objects
- From [Craig02]:
  - a set of objects,
  - a program structure or module,
  - a factory-like entity which creates objects,
  - a data type,
  - a concept
- An extensible template for creating objects, providing initial values for instance variables and methods

What is the difference between a class and an object?

What is the difference between a singleton class and an object?

Are there any real differences between records/ structs and objects and classes?

What are the benefits of bundling state and behaviour together?

Encapsulation

- Encapsulation means separating the interface of an abstraction from its implementation
- Key difference between objects & structs
- Facilitates stronger class invariants
- Common encapsulation mechanisms
  - functions and procedures
  - modules, classes and packages

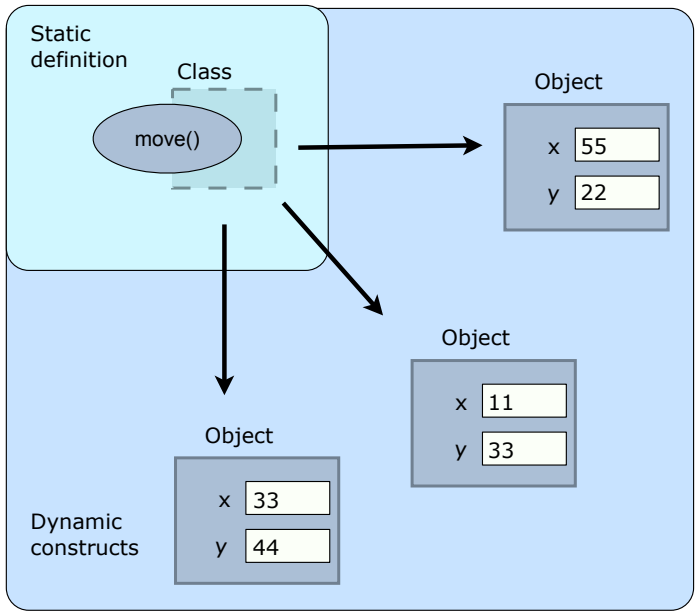
Information Hiding

- A design principle
- Hide data, structure and any differences between exposed data and internal representation
- What abstractions we use controls what information should be hidden
- Coupling and cohesion

Encapsulation ≠ Information Hiding

- But encapsulation is a prerequisite for information hiding

Class-based



Classes as first-class entities

- As a datatype, a class is usually considered as a compile-time construct
- In many languages (like Smalltalk, Ruby, Python etc.) a class is also an object -- each class is an instance of the unique metaclass, which is built in the language
- Methods can be invoked on classes just like on regular objects
- Creating objects can then be done by sending a message to the class

```
Being new initialize: "XEROX"

Being.new("Matz")
```


Metalevels in Programming Languages

Level 3 - Meta-Concepts in the metameta model, the metalanguage (language description)	Programming Language Concept
Level 2 - Language concepts (Metaclasses in the metamodel)	Class, Method, Attribute
Level 1 - Software Classes (meta-objects) (Model)	Car, void drive(){}, int[] colour
Level 0 - Software Objects	car1, car1.drive(), car1.color
Real world entities	  

Meta Classes

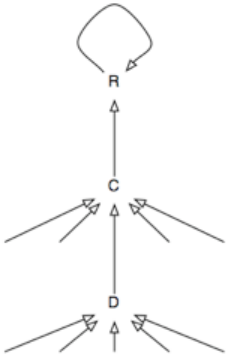
- 1 Level System
  - All objects can be viewed as classes and all classes can be viewed as objects (as in Self). "Single-hierarchy".
- 2 Level System
  - All Objects are instances of a Class but Classes are not accessible to programs. 2 kinds of distinct objects: objects and classes.
- 3 Level System
  - All objects are instances of a class and all classes are instances of Meta-Class. The Meta-Class is a class and is therefore an instance of itself. 2 kinds of distinct objects (objects and classes), with a distinguished class, the metaclass.
- 4 Level System
  - Like a 3 Level System, but there is an extra level of specialized Meta-Classes for classes.

Infinite Regression

- If the class of a class object is C, and C is an object, then what is the class of C, and what the class of its class' class object?
  - Predicative or impredicative class definitions

Stop Whenever

Class is an object that represents its own class

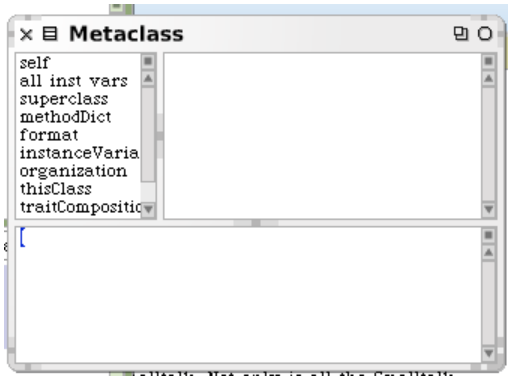


Class Creation in Smalltalk

```
Object subclass: #Car
   instanceVariableNames: 'colour'
   classVariableNames: ''
   poolDictionaries: ''
   category: 'IOOR09'
```


Class Creation in Smalltalk

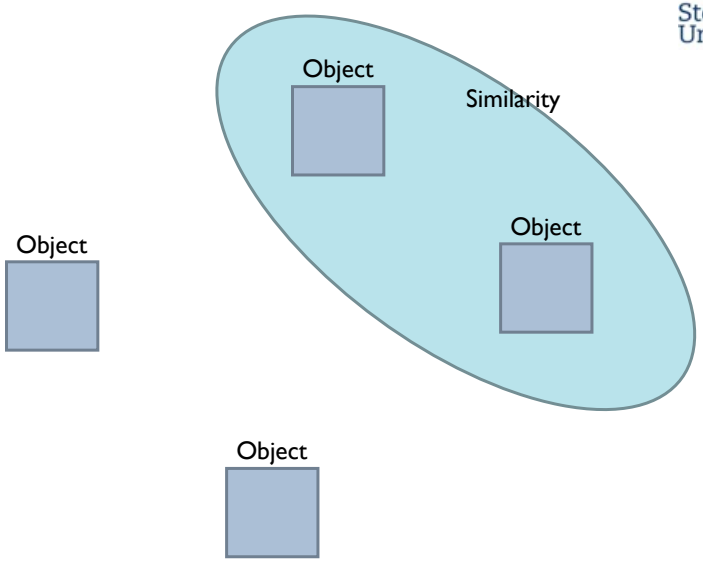
```
Object class inspect
```



Prototype-based PLs

- Invented after class-based languages in the 70'ies
- Replaces class instantiation with copying existing objects
- Replaces inheritance with more flexible delegation
- Cloned objects can can change invariantly of each other
- Also called:
  - Instance-based, Prototype-Oriented, Class-less
- Examples of languages:
  - Self, Cecil, JavaScript, Io

Prototype-based



JavaScript

- JavaScript is THE scripting language of the Web
- JavaScript is used in millions of Web pages to add functionality, validate forms, detect browsers, and much more
- But:
  - JavaScript has no direct relationship to Java
  - JavaScript can be used for other things than scripting browsers

JavaScript Syntax

Comments:	`// single line comment`
	`/* multi line`
	`comment */`

Identifiers:	First character must be a letter, _, or $; subsequent characters can be digits: `i, v17, $str, __proto__`

Basic literals:	`'a string', "another string", "that's also a string"`
	`17, 6.02e-32`
	`true, false, null, undefined`

Object literals:	`var point = { x:1, y:2 }`
	empty: `{}`
	nested: `var rect = {`
	`upperLeft: { x:1, y:2 },`
	`lowerRight: { x:4, y:5 } }`

Function literals:	`var square =` `function(x) { return x*x; }`
Array literals:	`[1,2,3]` `[]`
Operators:	assignement: = equal: == ! ! strict equal: ===

Object Properties

Reading properties

```
var book = { title:'JavaScript' };
book.title; //=>'JavaScript'
```

Adding new properties (at runtime)

```
book.author = 'J. Doe';
'author' in book; //=>true
```

Inspecting objects

```
var result = '';
for (var name in book) {
   result += name + '=';
   result += book[name] + ' ';
};

//=>title=JavaScript author=J. Doe
```

Deleting properties

```
delete book.title;
'title' in book; //=>false
```

Slots in PBLs

- Slots are simply storage locations located in objects
- Slots can be divided into two types:
  - Data slots, holding data items
  - Method slots, holding methods
- Methods are stored in exactly the same way as data items

Methods

- At runtime the keyword `this` is bound to the object of the method

```
var obj = { counter:1 };
obj.increment = function(amount) {
  this.counter += amount;
};
obj.increment(16);
obj.counter; //=> 17
```

- Accessing (vs. executing) methods

```
var f = obj.increment; typeof f; //=>
'function'
```

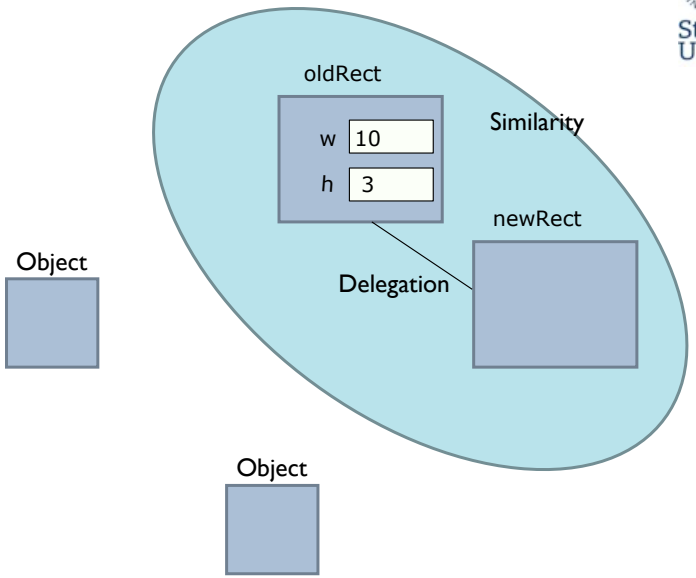

Delegation

- When an object receives a message it looks for a matching slot, if not found, the look-up continues its search in other known objects
- Typically, the search is done in the object's "parent", in its "parent's" "parent" and so on
- In JavaScript, an object delegates to its prototype object (the Mozilla interpreter allows one to access the prototype through the property __proto__)

Delegation, cont'd

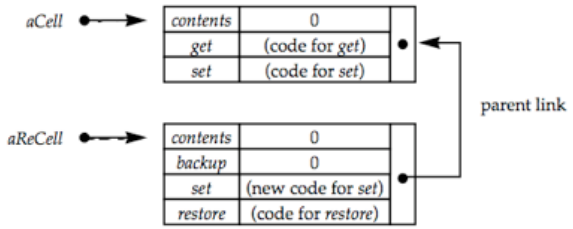
```
var oldRect = { width:10, height:3 };
var newRect = {};
newRect.__proto__ = oldRect;


"width" in newRect; //=>true newRect.hasOwnProperty
("width"); //=>false


newRect.width; //=>10
newRect.foo; //=>undefined
```

Prototype-based



Delegation

- As opposed to inheritance, delegation can be manipulated dynamically
- The method of the delegate will be executed in the scope of the original receiver
- Depending on the language, the number of possible delegates may differ

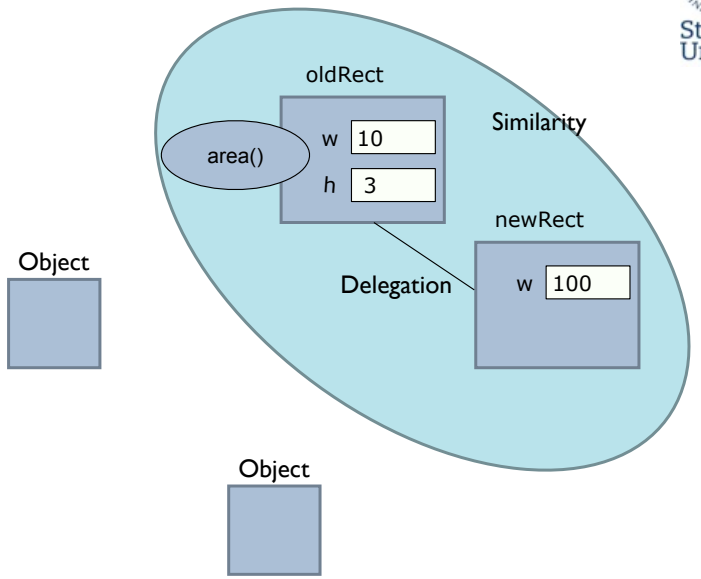


(Single-parent) Delegation

Delegation, cont'd

```
newRect.width = 100;

oldRect.area = function() {
   return this.width * this.height;
};

newRect.area(); //=>300
```


Prototype-based



Use of delegation

- Delegation — executing a method of some other object but in the context of self
- A lot more powerful than mere forwarding
- Delegation can be used to implement inheritance but not vice versa
- Very powerful — delegates are not known statically as in inheritance and can change whenever

Constructor Functions

- Constructors are functions that are used with the `new` operator to create objects

```
function Rectangle(w, h) {
   this.width = w;
   this.height = h;
   this.area = function() {
      return this.width * this.height;
   };
};

rect = new Rectangle(3,4);
rect.area(); //=>12
```

- The operator `new` creates an object and binds it to this in the constructor. By default the return value is the new object.

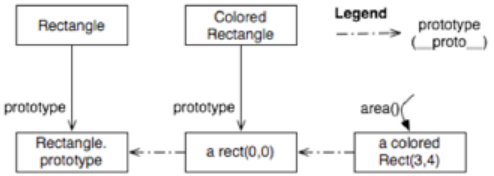
Constructor.prototype

- Each constructor has a prototype property (which is automatically initialised when defining the function)
- All objects created with a constructor share the same prototype

```
function Rectangle(w, h) {
   this.width = w;
   this.height = h;
};

Rectangle.prototype.area = function() {
   return this.width * this.height;
};
```


Constructor.prototype

```
   ...
function ColoredRectangle(w, h, c) {
   this.width = w;
   this.height = h;
   this.color = c;
};

ColoredRectangle.prototype = new Rectangle(0,0);

coloredRect = new ColoredRectangle(3,4,'red');

coloredRect.area();
```



Predefined Objects

- Global functions: Array, Boolean, Date, Error, Function, Number, Object, String,... eval, parseInt, ...
- Global objects: Math

Extending Predefined Objects

- Extending all objects:

```
Object.prototype.inspect = function() {
   alert(this);
};

'a string'.inspect();
true.inspect();
(new Date()).inspect();
```

- The last object in the prototype chain of every object is `Object.prototype`

The arguments object

```
function concat(separator) {
   var result = "";
   for (var i = 1; i < arguments.length; i++)
      result += arguments[i] + separator;
   return result;
};

concat(";", "red", "orange", "blue");
// =>"red;orange;blue;"
```


Other Prototype-based Languages

- Basic mechanisms
  - Object creation:ex nihilo, cloning, extension
  - Object representation(slots in JavaScript, Self, Io vs. attributes and methods in Agora, Kevo)
- Delegation
  - Double delegation in Io/NewtonScript
  - Multiple prototypes(aka.parents) in Self
  - Can prototype link be changed at runtime?
- Organization of programs (prototypical instance, traits, ...)

Benefits of prototypes

- Simple model, simpler than the class-based
- No use for special “inheritance” relations in the language
- Very flexible and expressive
- Changing prototypes to reflect state is a powerful concept
- Delegation is very powerful
- Handles special cases very well

Performance

- Sharing data and copy-on-write Method caches
- Inheritance (at least in static cases) costs memory in many slots
- Locality of reference if the methods are actually in the object

Prototypes vs. Classes

- Classes are static—requirements are not
- Unless you can predict all future requirements up front, class hierarchies will evolve
- Evolution of base classes is tricky and might break subclasses
- Eventually, refactoring or redesign is needed
- It is not uncommon to design a class that is only to be instantiated once. [Liebermann86]

Why do you think most OOPLs are class-based?

Stockholm University

References

- Timothy Budd,"An Introduction to Object- Oriented Programming", 2nd edition. Addison-Wesley, 2000.
- Iain Craig,"The Interpretation of Object- Oriented Programming Languages", 2nd edition,SpringerVerlag, 2002.
- Ian Joyner, "Objects Unencapsulated", Prentice-Hall, 1999.
- Henry Lieberman, "Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems", 1986.
- James Noble and Brian Foote, "Attack of the Clones", Proceedings of the 2002 conference on Pattern languages of programs.
- D.L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, Vol. 15, No. 12, 1972.

Stockholm University

References, cont'd

- Martín Abadi & Luca Cardelli, A Theory of Objects, Springer Verlag, 1996.
- Anton Eilëns, Principles of Object-Oriented Software Development, 2nd edition. Addison-Wesley, 2000.
- Kim Bruce, Foundations of Object-Oriented Languages: types and semantics, MIT Press, 2002.