Typology of programming languages

~ Prototype based object system ~

Engineering Properties, L.Caardelli 1996

- **Economy of execution.**
  How fast does a program run?
- **Economy of compilation.**
  How long does it take to go from sources to executables?
- **Economy of small-scale development.**
  How hard must an individual programmer work?
- **Economy of large-scale development.**
  How hard must a team of programmers work?
- **Economy of language features.**
  How hard is it to learn or use a programming language?

Table of Contents

Problem Statement

Traditional class-based OO languages are based on a deep-rooted duality:

- **Classes**: defines behaviours of objects.
- **Object instances**: specific manifestations of a class

> Unless one can predict with certainty what qualities a set of objects and classes will have in the distant future, one cannot design a class hierarchy properly

Self

Invented by David Ungar and Randall B. Smith in 1986 at Xerox Park

Overview:

- Neither classes nor meta-classes
- Self objects are a collection of *slots*. Slots are accessor methods that return values.
- Self object is a stand-alone entity
- An object can delegate any message it does not understand itself to the parent object
- Inspired from Smalltalks blocks for flow control
- Generational garbage collector

Example in self

- Copy object lecture and set fill title to TYLA

```
tyla := lecture copy title: 'TYLA'.
```

- add slot to an object

```
tyla _AddSlots: (| remote <- 'true'|).
```

- Modifies at runtime the parent

```
myObject parent: someOtherObject.
```

Impacts

- Javascript
- NewtonScript
- Io
- Rust
- Go

Table of Contents

Rust, Go, ...

Gang of 4 quote

Object-oriented programs are made up of objects. An object packages both data and the procedures that operate on that data. The procedures are typically called methods or operations.

Rust's documentation

Even though structs and enums with methods aren't called objects, they provide the same functionality, according to the Gang of Four's definition of objects.

Example in Rust

```
trait Foo {
  fn method(&self) -> String;
}

impl Foo for u8 {
  fn method(&self) -> String
       { format!("u8: {}", *self) }
}

impl Foo for String {
  fn method(&self) -> String
       { format!("string: {}", *self) }
}

fn do_something<T: Foo>(x: T) {
    x.method();
}
```

Duck Typing

If it walks like a duck and it quacks like a duck,
then it must be a duck

Deffered to a later lecture (about Genericity)

Table of Contents

CLOS

Developed in mid 80's.

Overview:

- Metaobject Protocol
- Meta-class
- Multiple Inheritance
- Multiple dispatch
- Generic Functions
- Method Qualifier
- Introspection

Small Example

```
(defclass human () (name size birth-year))
(make-instance 'human)
```

```
(defclass Shape () ())
(defclass Rectangle (Shape) ())
(defclass Ellipse (Shape) ())
(defclass Triangle (Shape) ())

(defmethod intersect ((r Rectangle) (e Ellipse))
    ...)

(defmethod intersect ((r1 Rectangle) (r2 Rectangle))
    ...)

(defmethod intersect ((r Rectangle) (s Shape))
    ...)
```

Summary