# Teoría de Lenguajes Teoría de la Programación

Clase 7: Programación orientada a objetos

#### **Principios**

- ADTs:
  - Encapsulamiento
  - Composición
  - Instanciación / Invocación

- Objetos
  - Herencia

# Objetos y clases

#### **Objeto**

Entidad con estado interno modificable a través de sus métodos.

#### Clase

Definición de un objeto en forma incremental.

Un objeto es una *instancia* de una clase.

#### Principios de OOP

- SOLID
  - Single responsibility
  - Open / Closed
  - Liskov substitution
  - Interface segregation
  - Dependency inversion
- GRASP (General responsibility assignment software patterns)
  - Polimorfismo
  - Alta Cohesión (High cohesion)
  - Bajo acoplamiento (Low coupling)
  - 0 ...
- Patrones de diseño

# Tipado

#### ¿Cuándo?

Tipado dinámico Tipado estático

#### ¿Qué?

Nominal

Estructural (compile time)

Duck Typing (runtime)

El tipado puede ser implicito(inferido) o explicito

#### OOP en Oz

```
class Counter
   attr val
   meth init(Value)
       val:=Value
   end
   meth browse
       {Browse @val}
   end
   meth inc(V)
       val:=@val+V
   end
end
```

```
Definición de la
clase
                    Instanciación
C1 = {New Counter init(10)}
{C1 browse}
{C1 inc(5)}
{C1 browse}
```

## OOP en Oz - First class messages

Los métodos de una clase funcionan como patterns.

Llamar a un método es enviar un mensaje a mapear con esos patrones.

El mensaje puede ser estático o dinámico.

#### OOP en Oz - Definición de métodos

- Argumentos fijos o variables
- Valores por defecto
- Modificadores de acceso
- Método por defecto (otherwise)

#### Herencia

```
class ColoredPoint from Point
   attr color
  meth init(X Y C)
      Point,init(X Y)
      color:=C
  end
  meth display
      {Browse "colored"#@color#" point at ("#@x#" , "#@y#")"}
   end
end
```

#### Herencia

Posibilidad de encapsular estados en tipos de datos:

Interfaz mas sencilla

Efectos secundarios

# Conceptos adicionales

### Mixin

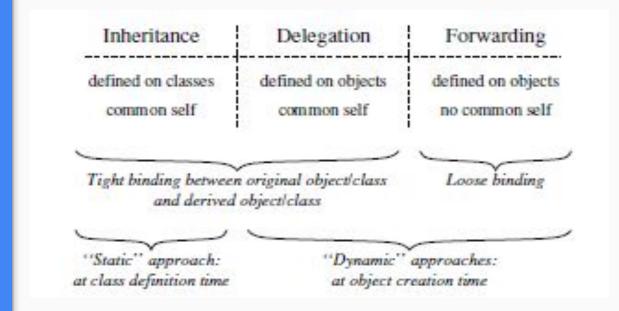
Clase que existe solamente para agregar comportamiento a otras via composición o herencia

A veces se conocen como Trait

Se usan para herencia multiple

# Mecanismos para extender funcionalidad

- Herencia
- Delegation
- Forwarding



## Forwarding

Una clase es forwardeable entonces instancia define a qué otra instancia enviarle un mensaje que desconoce.

# Delegation - Prototype based programming

Delegation lo que hace es similar a Forwarding pero preserva el estado de la instancia que delega

Principal uso en prototype-based programming

# Design by contract

Introducido por Bertrand Meyer en el diseño de Eiffel.

A cada rutina se le agrega en forma explícita una precondición y una poscondición.

Se agregan en forma explícita invariantes de clase

# Bibliografía

 Concepts, Techniques, and Models of Computer Programming - Capítulo 7, Peter Van Roy and Seif Haridi