Zero: una máquina virtual persistente, orientada a objetos pura y basada en prototipos

Grupo IMO – Universidad de Vigo
J. Baltasar García Perez-Schofield
http://webs.uvigo.es/jbgarcia/

El modelo de orientación a objetos basado en prototipos

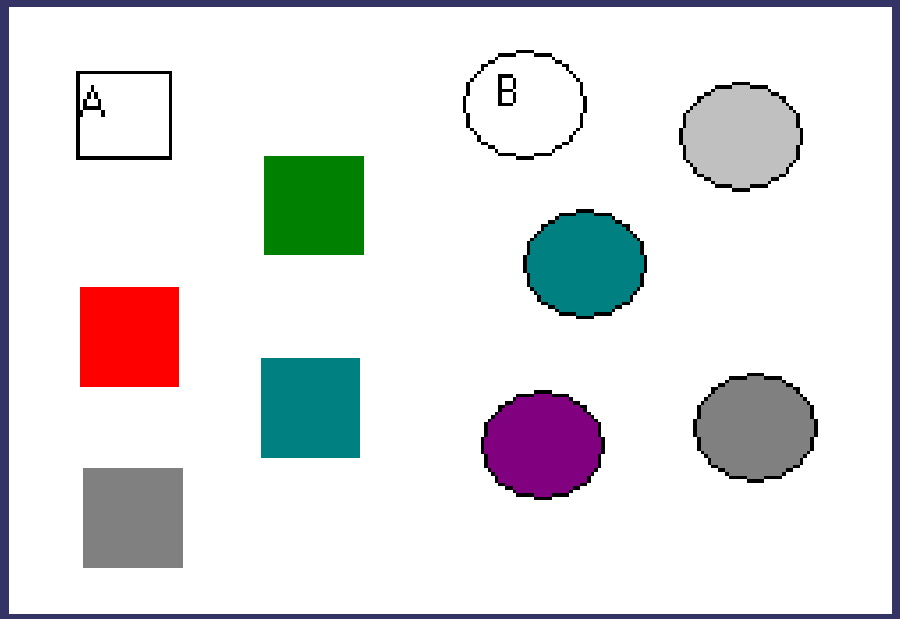
Orientación a objetos basada en prototipos

- Existen dos corrientes principales:
  - Lenguajes orientados a objetos basados en clases: C++, Object Pascal, Java, Eiffel ... son los más utilizados por la industria.
  - Lenguajes orientados a objetos basados en prototipos: Self, Kevo, Poet/Mica, Cecil ... son todos ellos experimentales, es decir, no se utilizan en la industria.
- **En realidad, el modelo de prototipos engloba al de clases.**

Terminología

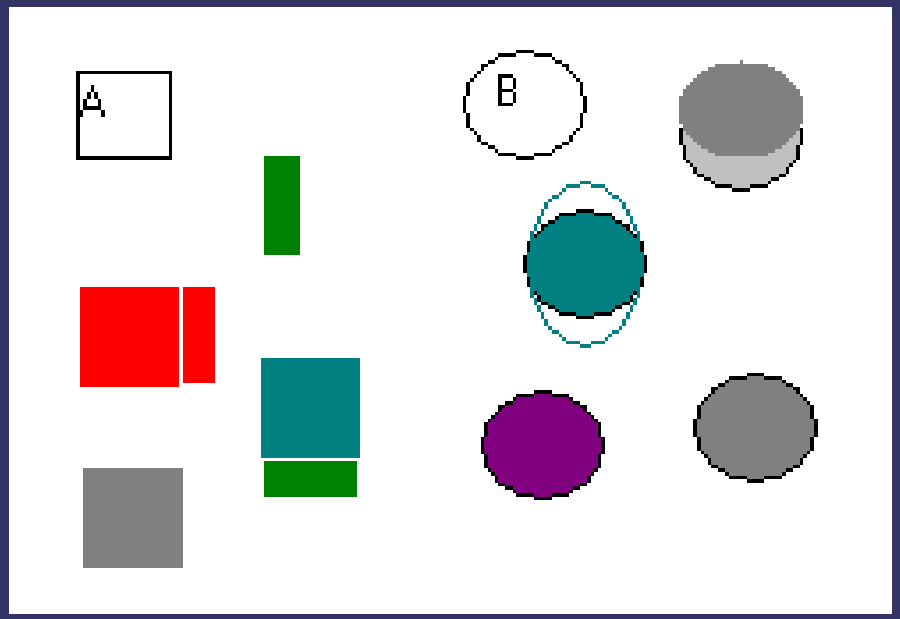
- ***Estado***: los *atributos* (terminología SmallTalk) o *datos miembro (terminología C++)* de un objeto. En el caso de un coche, su color, su longitud, cilindrada, ...
- ***Comportamiento***: los *métodos* (SmallTalk) o *funciones miembro* (C++) de un objeto. En el caso de un coche, arrancar, acelerar, frenar, apagar.
- **Mensaje**: ejecución de un método de un objeto. Si un objeto tiene un método f(), mandarle a O el mensaje f es lo mismo que ejecutar O.f()

Orientación a objetos basada en clases



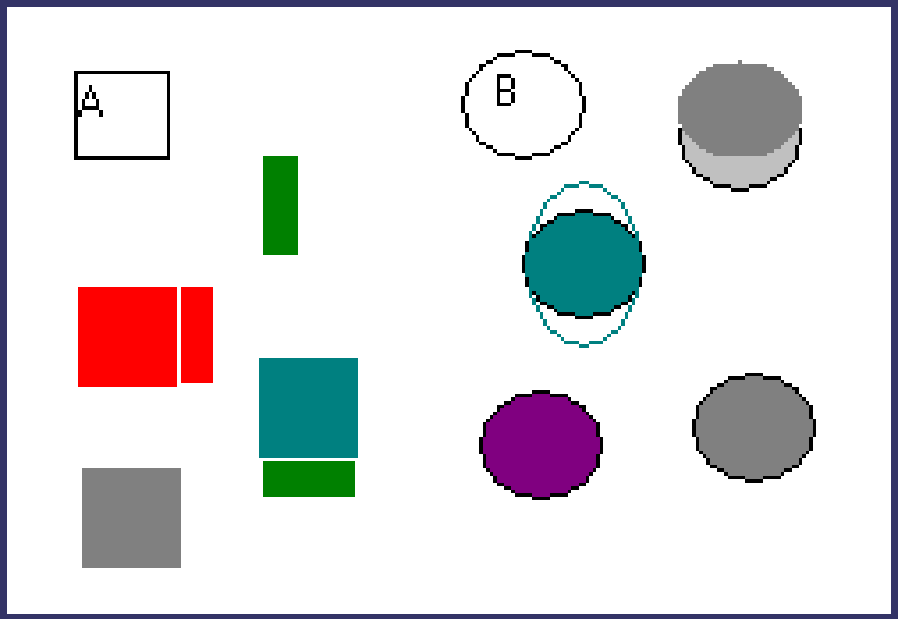
- Una clase es un “tipo” de objetos, es decir, un molde del que se obtienen nuevos objetos, que comparten similar comportamiento, cambiando el estado de los mismos.

Orientación a objetos basada en prototipos



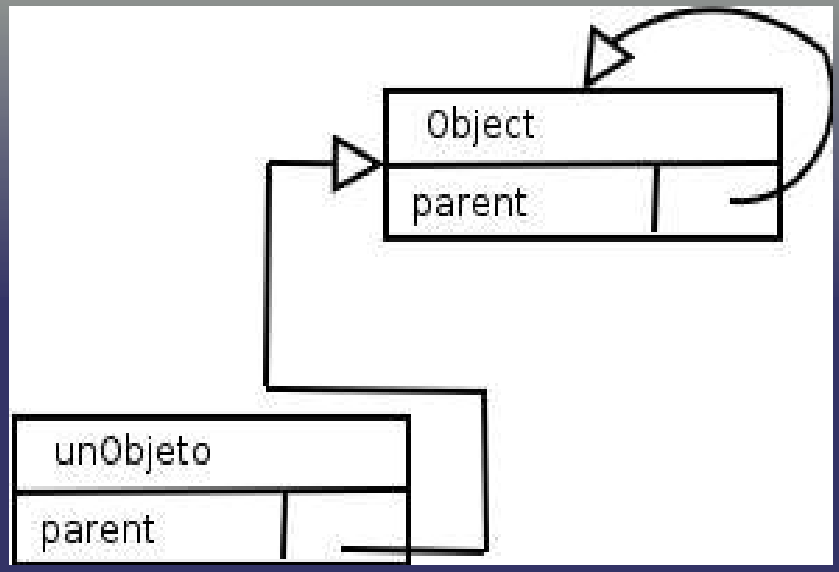
- No existen las **clases**. De hecho, todos los objetos son iguales en cuanto a categoría.
- Los nuevos objetos se copian de otros ya existetntes. Algunos de ellos son **prototipos**.

Orientación a objetos basada en prototipos



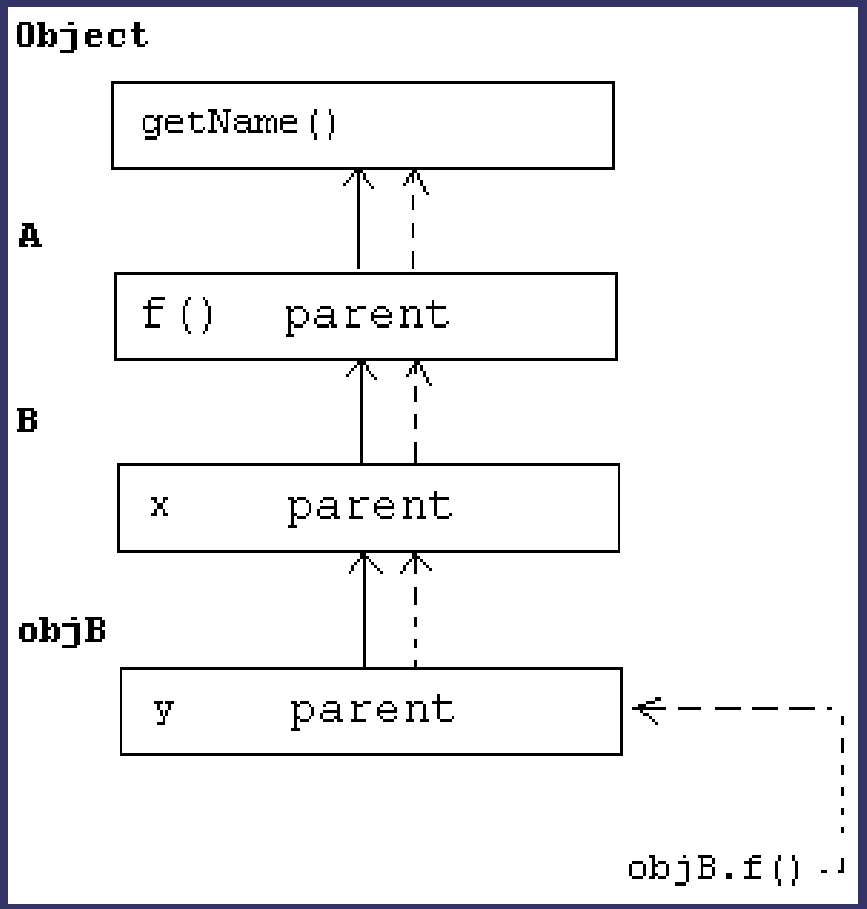
- Normalmente, en este tipo de lenguajes los objetos pueden modificarse, añadiendo o borrando métodos y atributos.
- Cada objeto es independiente, no necesitando información extra de ningún tipo.

Herencia



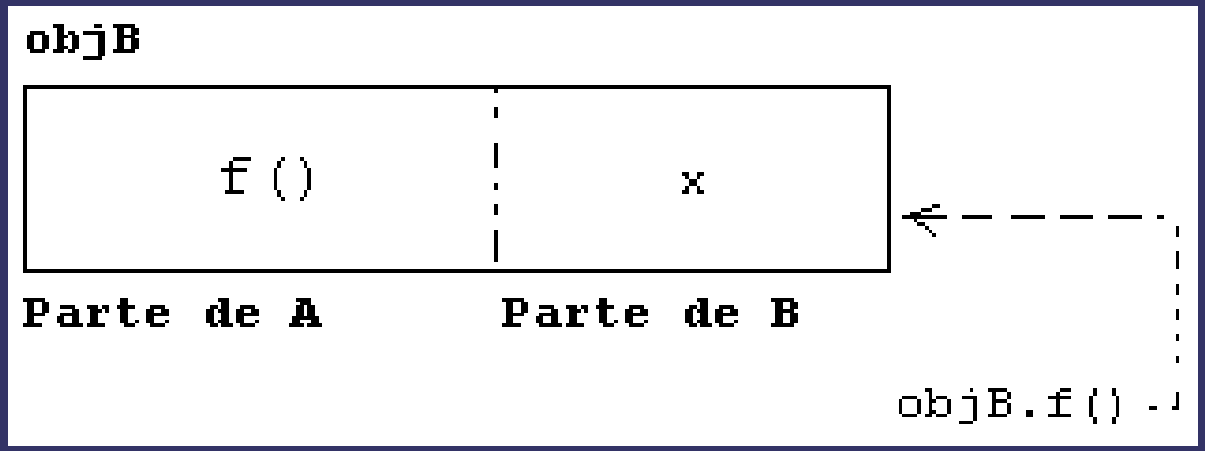
- La herencia en lenguajes basados en prototipos suele ser por delegación.
  - El objeto tiene uno o más aributos *parent*, de forma que cuando no puede responder a un mensaje, le reenvía éste a su padre.
- En el caso de los lenguajes basados en clases, ésta suele presentarse como concatenación
  - El objeto está compuesto por las partes que define cada una de las clases de las que hereda.

Herencia mediante delegación



- Existe una cadena de objetos apuntando a sus padres, hasta llegar a un objeto padre de todos.

Herencia por concatenación



- Todos los atributos y métodos heredados están disponibles en el mismo objeto (si bien se necesita la clase para poder interpretarlos).

Respuesta a mensajes

- Cuando un objeto no puede responder un mensaje, porque no posee el miembro (atributo o método), que se le pide, reenvía el mensaje al objeto que marca su atributo *parent*.
- Las relaciones de herencia, en lugar de ser un caso aparte, pasan a ser un caso particular de las relaciones de composición.

Respuesta a mensajes

- Ejemplo. Dados los objetos:

```
object A
  method + foo()
    System.console.write( “foo” )
    return
  endMethod
endObject

object B : A
endObject
```

Respuesta a mensajes

- El mensaje:

```
B.foo()
```

- No encuentra el método *foo*() en el objeto *B*, así que se sigue el atributo parent, que apunta a *A*, que sí tiene ese método, y es ejecutado.
- Si no se encontrara, entonces se produciría un error, que normalmente se traduce en una excepción. En este caso, la excepción producida sería:
  - "Método no encontrado" (EmethodNotFound)
  - "Mensaje no entendido" (EnumArguments).
  - Puede incluso no encontrarse al receptor, y entonces la excepción es "Objeto no encontrado", EObjectNotFound.

Creación de Objetos

- Al crearse los nuevos objetos mediante copia, no es necesario que existan los constructores de los lenguajes orientados a objetos basados en clases.
- Los objetos no sólo definen los tipos de datos de los atributos, como en las clases, sino que además ya tienen un valor asociado.

Creación de Objetos

- Por ejemplo:

```
object Persona

  attribute + nombre    = “Juan”
  attribute + apellidos = “Nadie”
  attribute + telefono  = “906414141”
  attribute + edad      = 18
  attribute + direccion = “Rue 13, Percebe”

  method + toString()
    toret = nombre.concat( apellidos )
    return toret
  endMethod

endObject
```

Creación de objetos

- El objeto *Persona* es un prototipo que servirá para crear nuevos objetos, aunque no existe ninguna diferencia entre un prototipo y cualquier otro objeto.
- Por ejemplo:

```
reference p2 = Persona.copy( “” )

p2.deleteAttributeNumber(
        getAttributeNumberByName(
                “direccion” )
)

reference p3 = p2.copy( “” )
```

Creación de objetos

- Crear un objeto de *Persona*:

```
reference p2 = Persona.copy( “” )
p2.ponNombre( “Paula” )
p2.ponApellidos( “Márquez Márquez” )
...
```

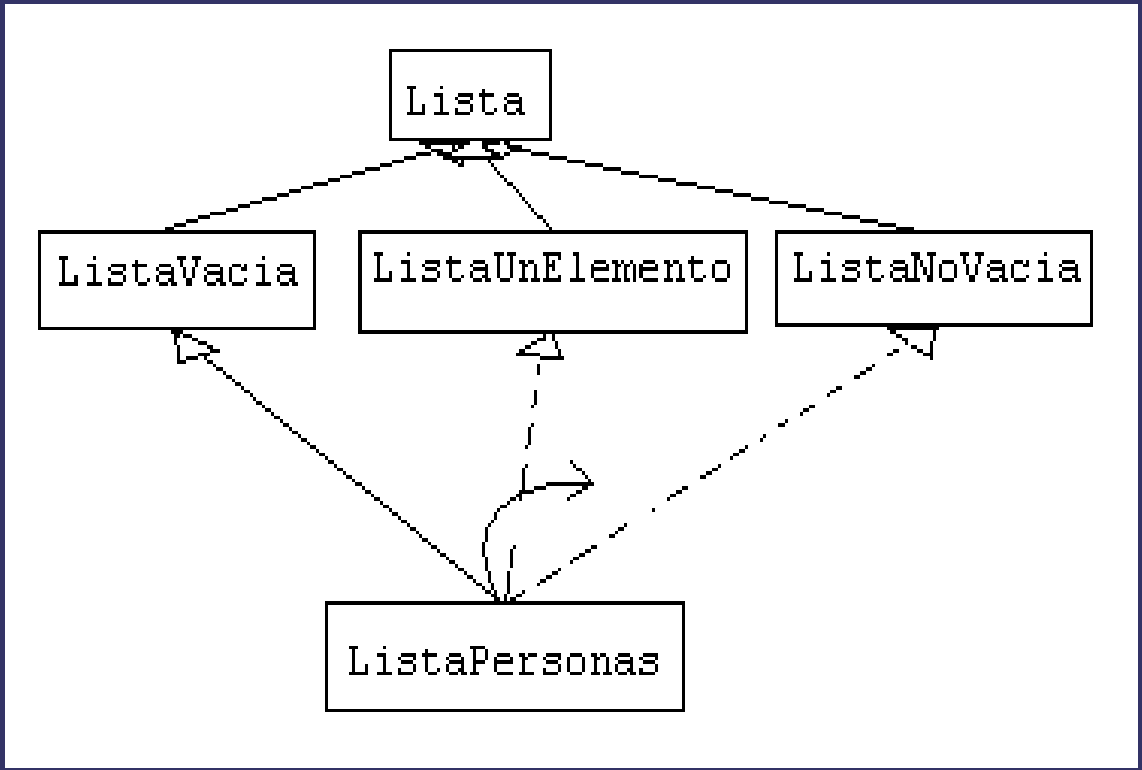
El modelo de prototipos incluye al de clases

- Los objetos que sirven de prototipos son equivalentes a las clases de aquellos lenguajes orientados a objetos basados en clases.
- La diferencia es que este modelo es mucho más flexible que el de clases.
- Incluso una “clase” en este modelo puede modificarse, al no ser más que un objeto.
- La delegación es un mecanismo altamente flexible, separando a los objetos de sus prototipos, como en los lenguajes basados en clases, pero no al *comportamiento* del *estado*.

Herencia dinámica

- En el caso de estar implementada por delegación, se abre una nueva posibilidad: el hecho de poder cambiar el atributo (ya que, normalmente, es un atributo más) que señala al *padre* del objeto, hace que un objeto pueda ser “hijo” de varios objetos, dependiendo del momento de la ejecución.
- El aprovechamiento de esta característica requiere cambiar ligeramente el tipo de programación.

Herencia dinámica



- Es posible cambiar, en tiempo de ejecución, al “padre” de un objeto.
- Es totalmente contrario a la corriente actual, que trata de detectar todos los errores posibles en tiempo de compilación.

Herencia dinámica

- En el método insertar de ListaVacia:

```
object ListaVacia : Lista

  method + insertar( obj )
    __this.^insertarPrimero( obj )
    parent = ListaUnElemento
  return
  endMethod

  method + size()
    return 0
  endMethod

endObject
```

Herencia dinámica

- En el método insertar de ListaUnElemento:

```
object ListaUnElemento : Lista

  method + insertar( n, obj )
    __this.^insertar( 1, obj )
    parent = ListaNoVacia
  return
  endMethod

  method + size()
    return 1
  endMethod

endObject
```

Herencia dinámica

- En el método borrar de ListaNoVacia:

```
object ListaNoVacia : Lista
  method + borrar( n )
    __this.^borrar( n )

    reference num = __this.size()
    num.isLessThan( 2 )
    jumpOnFalseTo fin
      parent = ListaUnElemento
  :fin
  return
  endMethod
endObject
```

Herencia dinámica

- Sus principales ventajas:
  - Simplificación: los métodos pueden escribirse según el tipo del objeto. El código es más simple.
  - Refuerza el papel de *clasificación* de la herencia.
- Su principal desventaja es que precise coordinar varios tipos para realizar una serie de tareas. Ésto puede conllevar errores y puede hacer las modificaciones de código más simples o más complicadas ...

Persistencia

Persistencia

- Persistencia es el término utilizado para la salvaguarda y recuperación de datos, concretamente, de objetos.
- La mayoría de las aplicaciones siguen una estructura sencilla de funcionamiento: recuperación de datos de una anterior ejecución, procesado, y salvaguarda de los nuevos datos.

Terminología

- *Serialización*: guardar el estado de un objeto directamente como una secuencia de bytes en un archivo en disco.
- *Swizzling*: conversión de punteros, de su formato en disco a su formato en memoria, y viceversa.
- *Activación*: Carga en memoria de un objeto que reside en disco.
- *Pasivación*: Salvaguarda en disco de un objeto que reside en memoria.

Persistencia

- El proceso de recuperación se denomina *unflattening*, desaplanar, y el de salvaguarda *flattening* (aplanar).
- Lo que está sucediendo en realidad en ambos procesos es una codificación de los datos contenidos en los objetos de la aplicación a un fichero, y viceversa.
- ¿Por qué no guardar directamente los objetos, y recuperarlos cuando sea necesario?

Salvaguarda de objetos

- Depende del lenguaje en el que se esté trabajando:
  - JAVA: posee un sistema de serialización de objetos a disco, más o menos automático.
  - C++: no posee ningún sistema de serialización, utilizándose mecanismos que tienen como base la grabación directa de objetos a disco:

```
fwrite(&objeto, sizeof(objeto), 1,Fichero)
```

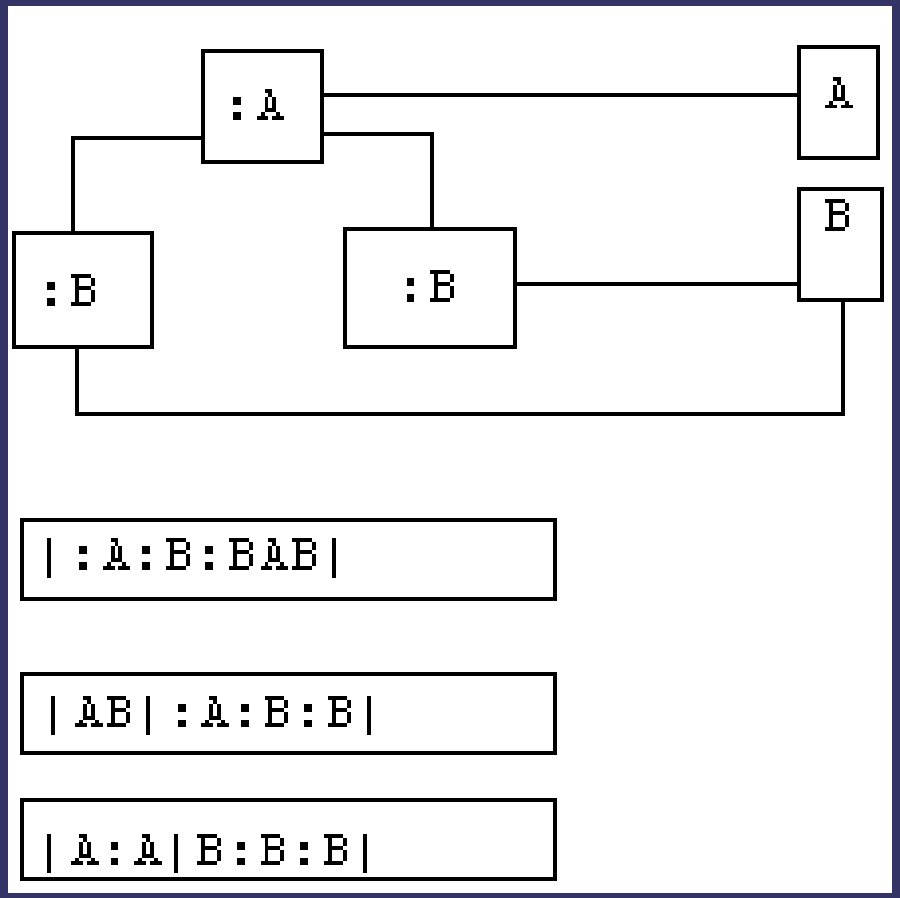
Recuperación de objetos

- En los lenguajes como C++, sólo puede recuperarse el estado de un objeto guardado previamente; sin embargo no puede saberse a qué clase pertenece el objeto, y ni siquiera si es un objeto.
- En lenguajes como Java, la recuperación es un poco mejor, puesto que podemos obtener el archivo .class y el de datos; sin embargo, la recuperación y el tratamiento de estos archivos sólo es realmente sencillo si es la misma aplicación que los grabó la que los va a utilizar.

La verdadera naturaleza de la persistencia

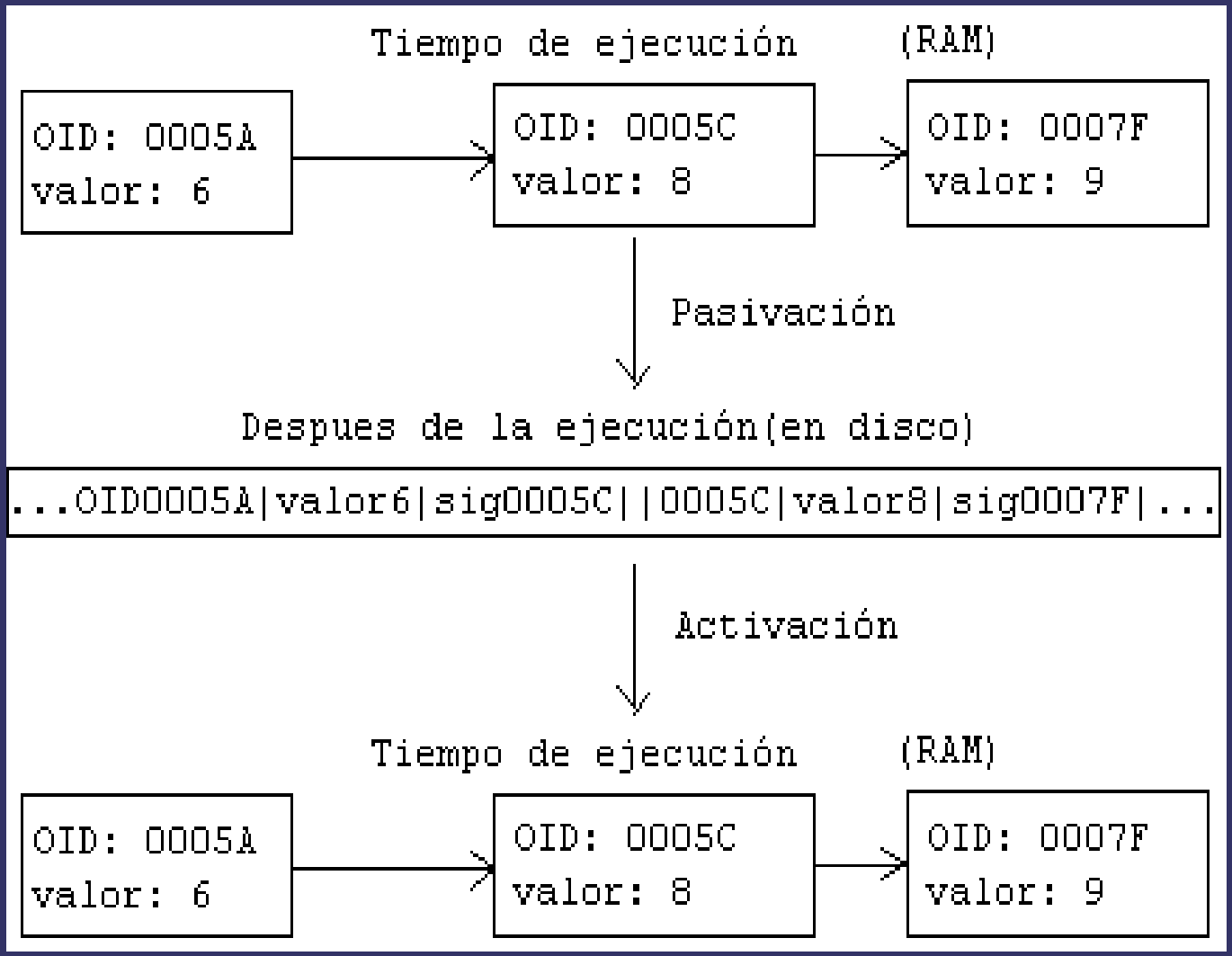
- La investigación en persistencia trata de ir un paso más allá que la idea inicial de guardar objetos en archivos de la misma forma que es posible guardar todo tipo de datos en ellos.
- Se trata de proporcionar un mecanismo tan automático como sea posible para la recuperación y salvaguarda de objetos, resultando por tanto obsoletos los conceptos de:
  - Archivo: ya no es necesario.
  - Distinción entre memoria primaria y secundaria: ya no es necesaria.

Clustering



- Deben ser necesarias las menos operaciones de lectura posibles para trabajar con esos objetos.
- En la primera, sólo es necesario cargar 1 cluster.
- En la segunda, dos clusters.
- En la segunda, también dos clusters.

Swizzling



- Cuando se guardan objetos, se sustituyen sus punteros por OID's.
- El proceso contrario sucede cuando se cargan esos objetos en memoria.

Evolución del esquema
(Schema Evolution)

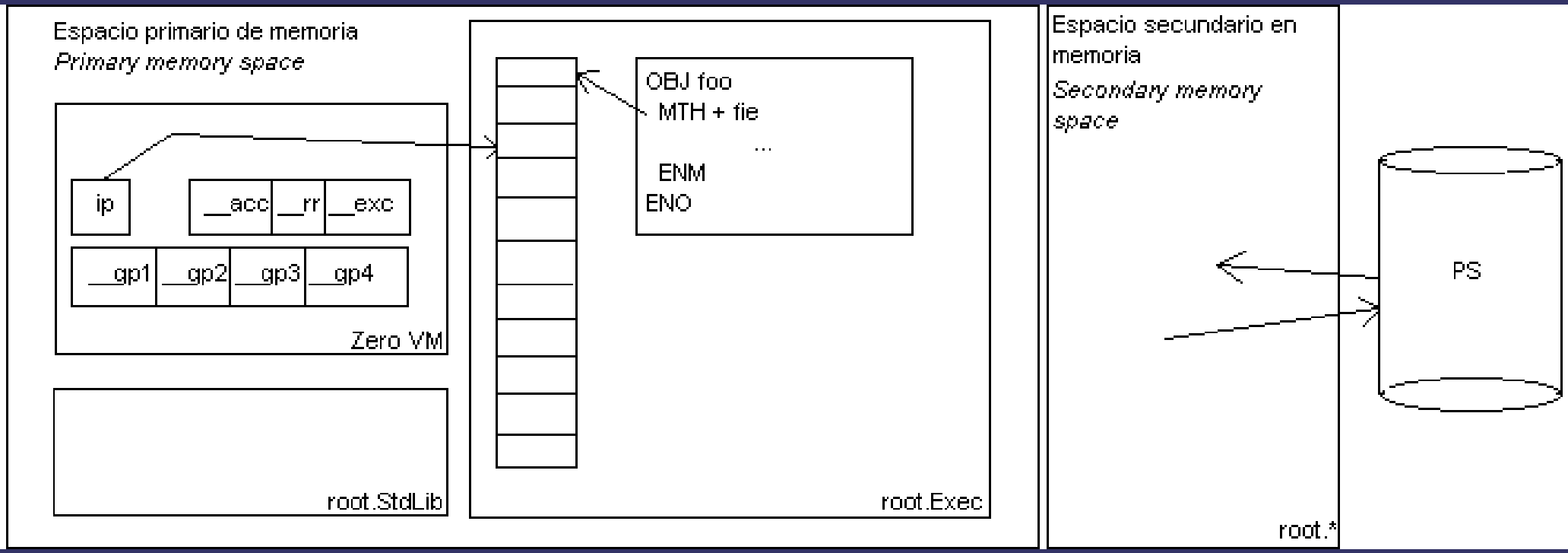
- Se trata del mismo problema que sucede cuando en una base de datos relacional se cambia una tabla: todos los registros de esa tabla deben ser adaptados.
- En una base de datos orientada a objetos, o en un almacenamiento persistente, éste problema sucede cuando una clase cambia, es decir, es modificada. Todos sus objetos deben cambiarse también.

Máquina Virtual Zero

Máquina virtual Zero

- Modelo de OO basado en prototipos
- Incorporará persistencia basada en contenedores.
- Recolección de basura basada en conteo de referencias por objeto.

Arquitectura de Zero



Lenguajes que siguen el modelo de orientación a objetos basado en prototipos

- Self
  - Fue creado en los laboratorios de Sun, y muchas partes de su sistema son las precursoras de Java.
  - Fue el primero en implementar el modelo de prototipos, que también fue inventado por ellos. Self trata de ser todo lo dinámico que sea posible, haciendo el mínimo chequeo en tiempo de compilación.
  - http://research.sun.com/research/self/
- Io
  - Fue creado por uno de los desarrolladores de Self, siguiendo sus directrices principales.
  - http://www.iolanguage.com/

Lenguajes que siguen el modelo de orientación a objetos basado en prototipos

- Kevo
  - Kevo fue desarrollado para una tesis doctoral, como una demostración de que un lenguaje orientado a objetos y basado en prototipos podía incorporar técnicas modernas como la comprobación de errores que realiza *C++*, por ejemplo, en tiempo de compilación.
  - Implementa herencia por concatenación.
  - Es menos flexible en tiempo de ejecución que *Self* o *Io*.
  - ftp://cs.uta.fi/pub/kevo

Lenguajes que siguen el modelo de orientación a objetos basado en prototipos

- Cecil
  - Cecil fue desarrollado en la universida de Washington. Incorpora el concepto de objetos predicados, es decir, que mediante una condición inherente al objeto, y la herencia dinámica, es posible llevar el concepto de programación por contrato al nivel del objeto.
  - Es un lenguaje basado en prototipos.
  - http://www.cs.washington.edu/research/projects/cecil/www/cecil.html

Lenguajes que siguen el modelo de orientación a objetos basado en prototipos

- Zero. Aún en desarrollo, en la Universidad de Vigo.
  - Todavía no tiene lenguaje, sino sólo un macroensamblador.
  - Simple, basado en prototipos.
  - Su principal característica es que incorporará persistencia.
  - http://trevinca.ei.uvigo.es/~jgarcia/TO/zero/

Lenguajes que siguen el modelo de orientación a objetos basado en prototipos

- Otros lenguajes
  - http://www.programming-x.com/programming/prototype-based.html

Bibliografía

- Sobre Self y el modelo de prototipos en general:
  - Cuesta, P., García Perez-Schofield, B., Cota, M. (1999). “Desarrollo de sistemas orientados a objetos basados en prototipos”. Actas del Congreso CICC' 99. Q. Roo, México.
  - Smith & Ungar (1995). “Programming as an experience, the inspiration for Self”. European Congress on Object-Oriented Programming, 1995.
  - Ungar & Smith. (1987). “Self: The power of simplicity”. Actas del OOPSLA.

Bibliografía

- Sobre Self y el modelo de prototipos. Cuestiones prácticas sobre desarrollo de aplicaciones:
  - Ungar, Chambers *et al*. (1991). "Organizing programs without classes". *Lisp and Symbolic Computation* 4(3), Kluwer Academic Publishers, June, 1991
  - Chambers, Ungar, Chang y Hözle. (1991). "Parents are Shared Parts: Inheritance and Encapsulation in Self". Lisp and Symbolic Computation 4(3), Kluwer Academic Publishers, June, 1991

Bibliografía

- Sobre Kevo y el modelo de prototipos:
  - Taivalsaari, Antero (1996). *Classes Versus Prototypes: Some Philosophical and Historical Observations*. ResearchIndex, The NECI Scientific Literature Digital Library: http://citeseer.nj.nec.com/taivalsaari96classes.html
  - Antero Taivalsaari (1996): On the Notion of Inheritance. *ACM Comput. Surv. 28*(3): 438-479
  - Antero Taivalsaari (1995) Delegation versus Concatenation or Cloning is Inheritance too. *OOPS Messenger 6*(3): 20-49
  - Taivalsaaari, A., Kevo - *A prototype-based object-oriented language based on concatenation and module operations*. University of Victoria Technical Report DCS-197-1R, Victoria, B.C., Canada, June 1992

Zero: una máquina virtual persistente, orientada a objetos pura y basada en prototipos

Grupo IMO – Universidad de Vigo
J. Baltasar García Perez-Schofield
http://webs.uvigo.es/jbgarcia/