Curso de Doctorado: Tecnologías de Objetos

Grupo IMO
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática

J. Baltasar García Perez-Schofield
http://webs.uvigo.es/jbgarcia/

Implementación de Lenguajes Orientados a Objetos

Terminología

- Herencia:
  - Por concatenación: el nuevo objeto (B) se construye concatenando el objeto del que hereda (A) con el mismo objeto (B).
  - Por delegación: los objetos son independientes. Si un objeto B hereda de A, entonces, cuando no sea capaz de satisfacer un mensaje F, delegará (le pasará) ese mensaje F en el objeto A.
- Comportamiento: El conjunto de métodos de un objeto (en muchos lenguajes, su clase).
- Estado: los valores de los atributos de un objeto.

Implementación de Lenguajes Orientados a Objetos

- Existen dos puntos de vista principales:
  - Integración de unas extensiones de orientación a objetos en un lenguaje tradicional, como C o Pascal.
  - Creación de un lenguaje totalmente nuevo.
- Por otra parte, existen dos tipos principales de lenguajes orientados a objetos:
  - Aquellos basados en clases
  - Los que están basados en prototipos

Lenguajes basados en clases

Extensión de un lenguaje tradicional para el soporte de extensiones de orientación a objetos

Implementación de Lenguajes Orientados a Objetos

- Veremos la posibilidad de dotar a un lenguaje como C de unas extensiones de Orientación a objetos.
- De hecho, la primera herramienta que Bjarne Stroustrup creó para poder compilar C++ (en aquel momento, “C con clases”), era un simple preprocesador.

Clases

- La clase es una plantilla a partir de la cual se crean los objetos.

```
class Coche {
   int numRuedas;
   int color;
   int combustible();
   void arranca();
 };
 void Coche::arranca() {
   combustible--;
 }
```

Clases

- La base de una clase es un registro (`struct`).
- La única dificultad es que los registros no admiten métodos ...
- ... pero puede simularse su *pertenencia:*

```
struct Coche {
    int numRuedas;
    int color;
    int combustible;
}
void Coche_arranca(struct Coche &this) {
    this->combustible--;
}
```

Métodos

- Los métodos son funciones en C, que tienen un argumento *this*, que no es más que el objeto que está ejecutando ese método en un momento dado.
- Es decir, *this* señala a la `struct Coches` apropiada para cada momento.
- Así, todos los métodos tienen un argumento más del que declararíamos en nuestro “C con clases”.

Métodos estáticos

- La única excepción a lo anterior son los métodos de clase o estáticos (notación C++). Éstos pertenecen a la clase, no al objeto. En realidad, ésto significa que ese método, una vez traducido a función, **NO** posee un argumento extra llamado *this*.

Ejemplo de traducción

- El siguiente programa:

```
class Coche {
public:
   int numRuedas;
   int color;
   int combustible;
   static void encontrarGasolinera();
   void arranca();
};
//...
int main(void) {
    Coche micoche;

    miCoche.color = 1; /* BLANCO */
    micoche.arranca();
}
```

Ejemplo de traducción

- Sería traducido como:

```
struct Coche {
    int numRuedas;
    int color;
}
void Coche_encontrarGasolinera() {
  // ...
}
void Coche_arranca(struct Coche &this) {
  // ...
}
int main(void)
{
  struct Coche miCoche;
  miCoche.color = 1; /* BLANCO */
  Coche_arranca(&miCoche);
}
```

Compilación

- Nótese que es posible, a pesar de todo, realizar una comprobación de tipos estricta como la que hace C++, en este preprocesador de “C con clases” a C.
- La comprobación estática de tipos, en tiempo de compilación, es uno de los puntos más fuertes de C++, ya que es una forma de detectar errores antes de la ejecución de un programa.
- Se pueden añadir fácilmente criterios de visibilidad (*private*, *protected*).

¿Qué quedaría por implementar?

- La implementación que se ha presentado hasta ahora permite encapsulación directamente, por lo que sería conveniente para la creación de TADS (Tipos Abstractos de Datos).
- Podría implementarse la herencia de forma sencilla, (simplemente, añadiendo a una clase la definición de la clase de la que hereda), por concatenación.
- Sin embargo, es más trabajoso soportar polimorfismo. Para ello, será necesaria una estructura como la vtable implementada en C++.

Lenguajes basados en clases

Creación genérica de un lenguaje orientado a objetos desde cero

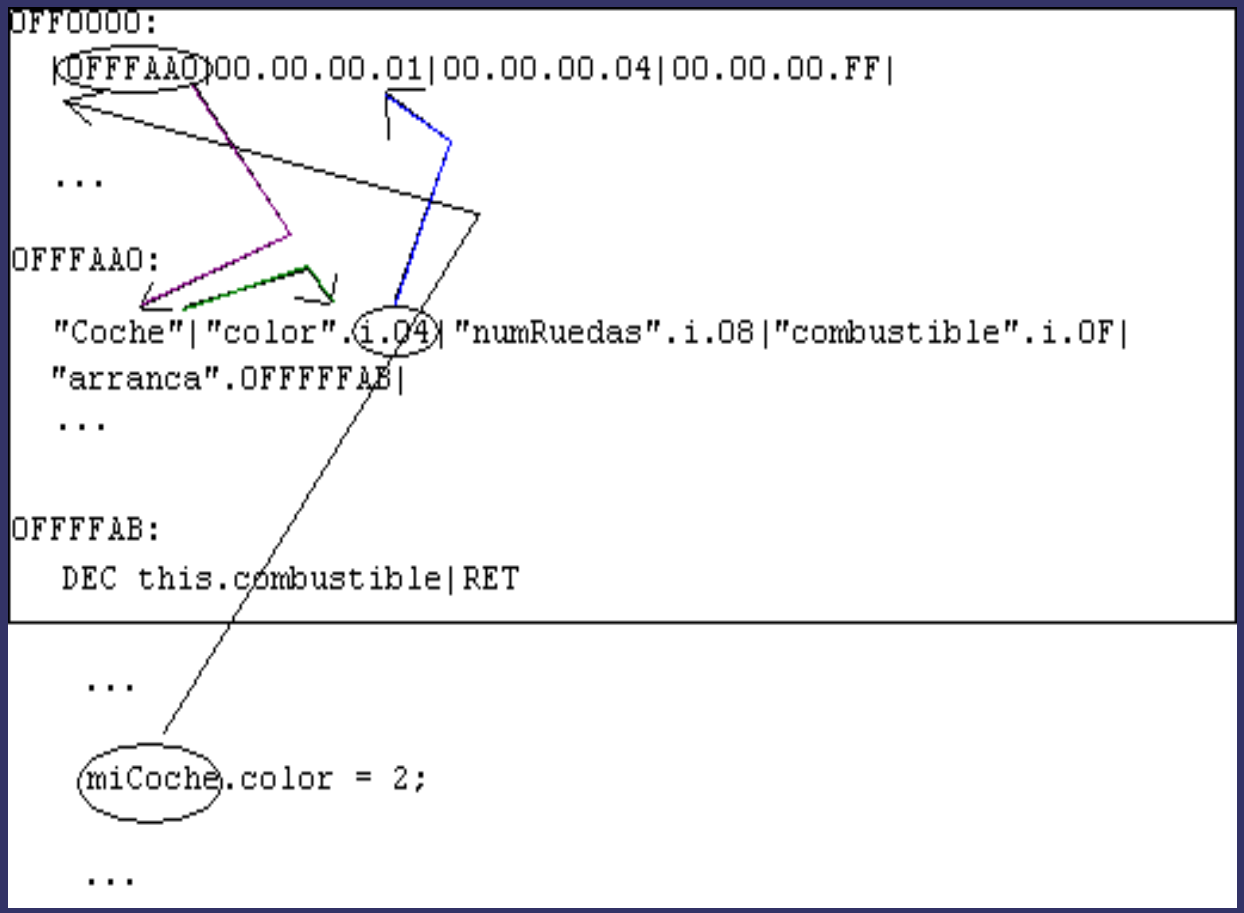
Clases y objetos

- Será necesario distinguir, obligatoriamente, entre:
  - comportamiento (los *métodos*, que residirán en la clase, y la descripción de los *atributos*), y
  - el *estado* (los valores de los *atributos*, que residirán en el objeto).
- Es posible prescindir de las clases en tiempo de ejecución, tal y como hace C++, o mantenerlas.
- Si se mantienen, entonces esas estructuras son conocidas como metaobjetos.

Clases y objetos

- Suponiendo la misma clase que la anterior:
  - Aunque no es necesario, los métodos pueden seguir implementándose como funciones del lenguaje al que se les pasa un puntero extra llamado *this*.
  - Existirá el objeto, con el estado, diferente, para cada objeto, y el metaobjeto, es decir, la clase, común a todos los objetos.
  - El metaobjeto es consultado para resolver las llamadas a los atributos y métodos de los objetos

Representación esquemática de objetos en memoria



- El metaobjeto contiene los desplazamientos de los atributos en el objeto y punteros a las funciones que actúan como métodos.
- El objeto contiene únicamente el estado.

Representación esquemática de objetos en memoria

- Así, para resolver `miCoche.color = 2;`,
  - primero se busca la dirección de memoria que representa la referencia “miCoche”.
  - desde allí, se va a la metaclase (la información de la clase) “Coche”.
  - Se localiza el atributo “color”.
  - Se le suma a la referencia de “miCoche” el desplazamiento de “color”.
- Así, finalmente, se obtendría la traducción, si fuese lenguaje C, de “`*((int *)miCoche + 4) = 2;`”

Tiempo de compilación o tiempo de ejecución

- Lo anterior puede suceder en tiempo de compilación, o bien en tiempo de ejecución.
  - En compilación: - flexible, + comprobaciones. Suelen ser lenguajes del tipo de C++.
  - En ejecución: + flexible, - comprobaciones. Suelen ser lenguajes del tipo de SmallTalk, Python ...
- Es todavía posible un tipo intermedio, como veremos, en el que un lenguaje como Self, basado en prototipos realiza comprobación estática de tipos al compilar. El ejemplo es Kevo.

Lenguajes basados en prototipos

Creación genérica de un lenguaje orientado a objetos desde cero

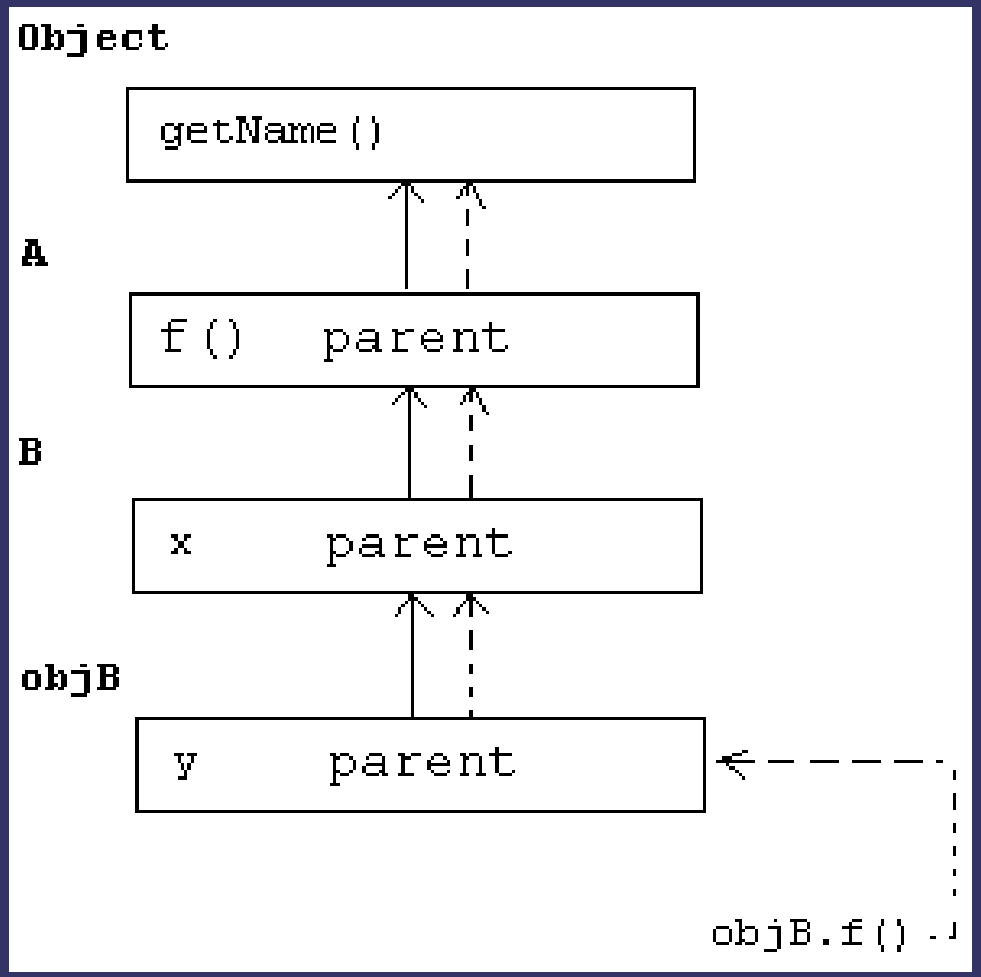
Lenguajes basados en prototipos

- Los lenguajes basados en prototipos no disponen de clases, sino que los nuevos objetos son copiados de otros objetos ya existentes.
- Esos objetos son los *prototipos*. Sin embargo, los nuevos objetos pueden ser modificados al margen de los prototipos.
- Es un modelo muy flexible y sencillo.
- Engloba al modelo de clases.

Implementación

- Puesto que cada objeto no depende de una clase y puede cambiar independientemente al margen del prototipo del que se copió, la estructura del objeto debe albergar no sólo el estado, sino también el comportamiento.
  - Métodos y atributos conviven en el mismo espacio en memoria para un objeto.
- La herencia se implementa por delegación, si bien sería posible implementarla por concatenación, como en Kevo.

Representación esquemática de objetos en memoria



- El objeto contiene métodos y atributos.
- Cuando no puede satisfacer un mensaje, lo *delega* en su padre.
- La herencia, por *delegación*, puede ser dinámica.

Conclusiones

Modelos de Orientación a Objetos

- Dos extremos del espectro de modelos orientados a objetos.
  - El modelo más restrictivo, en potencia, es el de clases.
    - Verificación de tipos en tiempo de compilación.
    - Herencia por concatenación.
  - El modelo más flexible, en potencia, es el de prototipos.
    - No hay casi comprobaciones en tiempo de compilación.
    - Herencia por delegación.
- Sin embargo, es posible crear lenguajes que empleen modelos intermedios dentro de este rango.

Implementación

- Las características que condicionan la implementación de un lenguaje son:
  - Herencia
  - Existencia de clases
- Sin embargo, debe recordarse que es posible separar la implementación en varias capas.
  - es posible para un compilador de un lenguaje basado en clases generar código para una máquina virtual orientada a objetos y basada en prototipos.

Bibliografía

- Bjarne Stroustrup, creador de C++:
  - Página personal
    - http://www.research.att.com/~bs/
  - “El Lenguaje de programación C++”
    - http://www.research.att.com/~bs/3rd.html
  - “Diseño y evolución de C++”
    - http://www.research.att.com/~bs/dne.html
  - Otras publicaciones:
    - http://www.research.att.com/~bs/books.html
- SmallTalk
  - Squeak: http://www.squeak.org/
  - SmallTalk en general: http://www.esug.org/

Bibliografía

- Self
  - Página web: http://research.sun.com/self/index.html
  - Implementación: http://research.sun.com/research/self/papers/elgin-thesis.h
  - Otros: http://research.sun.com/research/self/papers/papers.html
- Kevo
  - http://burks.brighton.ac.uk/burks/foldoc/44/63.htm
- Python
  - Lenguaje basado en clases, implementado sobre prototi-pos.
  - Página web: http://www.python.org/

Bibliografía

- Búsqueda de artículos:
  - http://www.researchindex.com

Curso de Doctorado: Tecnologías de Objetos

Grupo IMO
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática

J. Baltasar García Perez-Schofield
http://webs.uvigo.es/jbgarcia/