Doctoral course: object technologies

IMO Group
Department of computer science
University of Vigo

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

Implementation of object-oriented languages

Terminology

- Inheritance:
  - By concatenation: the new object (B) is built concatening the object it inherits from (A) with the same object (B).
  - By delegation: objects are independent. If an object B inherits from another object A, then when it were unable to satisfy a given message (F), it will delegate it in its parent (it will send F to A).
- Behaviour: The set of methods of an object (in many programming langages, its class).
- State: the values of all attributes in an object.

Implementation of object-oriented programming languages

- There are two main possibilities:
  - Integration of object-oriented extensions in a traditional language, such as C or Pascal.
  - Create a completely new language.
- There are two main kinds of object-oriented programming languages:
  - Class-based ones.
  - Prototype-based ones.

Class-based languages

Extending a traditional language with object-oriented capabilities.

Implementación de Lenguajes Orientados a Objetos

- Discussion about adding object-oriented capabilities to the C language.
- In fact, the first tool Bjarne Stroustrup created in order to be able to compile C++ (C with classes, at that time), was a simple preprocessor.

Classes

- Classes are moulds that allow object creation.

```
class Car {
   int numWheels;
   int color;
   int fuel;
   void startUp();
 };
 void Car::startUp() {
   fuel--;
 }
```

Clases

- The base for a class is a record (`struct`).
- The only problem is that records can't store functions ...
- ... but this can be simulated:

```
struct Car {
   int numWheels;
   int color;
   int fuel;
}
void Car_startUp(struct Coche &this) {
    this->fuel--;
}
```

Métodos

- Methods are C functions, which have a *this* argument, which is nothing else than the object executing that method at a given time.
- This means that *this* points to the appropriate `struct Coches` for each moment.
- Thus, all methods have an extra argument apart from the ones that would be declared in a method of this “C with classes” programming languages.

Static methods

- The only exception are class methods or static (in C++ terminology). They pertain to classes, not to objects. This means that this method, once translated into a C function, will **not** have the *this* parameter.

Translation example

- The following program:

```
class Car {
public:
  int numWheels;
  int color;
  int fuel;
  static void findGasStation();
  void startUp();
};
//...
int main(void) {
    Car myCar;

    myCar.color = 1; /* WHITE */
    myCar.startUp();
}
```

Ejemplo de traducción

- Would be translated as:

```
struct Car {
   int numWheels;
   int color;
}
void Car_findGasStation() {
  // ...
}
void Car_startUp(struct Car &this) {
 // ...
}
int main(void)
{
  struct Car myCar;
  myCar.color = 1; /* WHITE */
  Car_startUp( &myCar );
}
```

Compilation

- Note that it is possible to do a strict type-checking at compile time, as C++ does, in this preprocessor.
- Compile-time type checking is one of the strongest points of C++, as it is a way of detecting errors before the execution of a program.
- It is very easy to add visibility criteria (*private*, *protected*).

What is it left for implementation?

- Encapsulation is directly supported by the implementation of translation given here.
- Inheritance can be easily added by merging structures when one derives from another one. This is inheritance by concatenation.
- However, polymorphism is not so simple to implement. It is needed an structure as the *vtable* employed in C++.

Class-based languages

Creation of an object-oriented programming language from zero

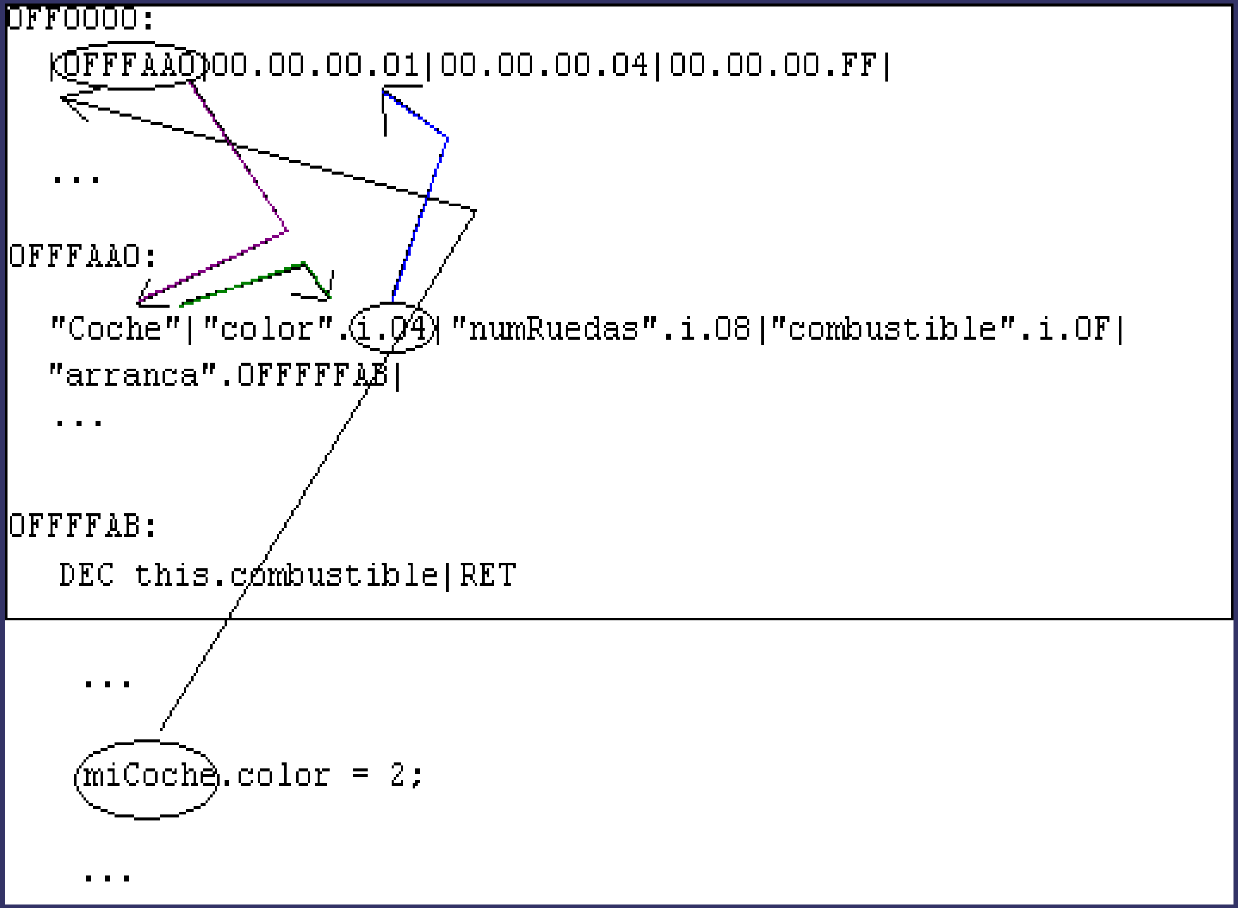
Classes and objects

- It will be mandatorily needed to distinguish between:
  - behaviour (*methods*, which will be stored in the class, and the <u>description</u> of the *attributes*), and
  - *state* (the values of the attributes, which will be stored in the object).
- It is possible to dismiss the class information at run time, as C++ does, or keep them as introspection information.
- If they are mantained at run time, then they are known as meta objects.

Clases y objetos

- Supposing the same class example than previously:
  - Although it is not necessary, methods can still be implemented as functions of the programming language that accept an extra argument *this*.
  - There will be the object, holding the state of the object, and the metaobject, i.e., the class, as a common resource for all objects, at runtime.
  - The metaobject is consulted in order to resolve calls to attributes and methods.

Schematic representation of objects in memory



```
0FF0000:
   |0FFFAA0|00.00.00.01|00.00.00.04|00.00.00.FF|

   ...

0FFFAA0:

   "Coche"|"color".i.04|"numRuedas".i.08|"combustible".i.0F|
   "arranca".0FFFFFAB|
   ...

0FFFFAB:
   DEC this.combustible|RET

     ...

    miCoche.color = 2;

     ...
```

- Metaobjects contain the shift for each attribute and pointers to the functions what play the role of methods.
- Objects just contain the state.

Schematic representation of objects in memory

- Thus, for resolving `myCar.color = 2;`,
  - firstly the pointer “myCar” is dereferenced.
  - from there, the metaclass is reached (the class information) “Car”.
  - The shift for the attribute “color” is found.
  - The pointer “myCar” is shifted as specified by the metaclass for “color”.
- Finally, the translation in C language would be “`*((int *)myCar + 4) = 2;`”

Compilation time or execution time

- The previous process can happen at compile or execution time.
  - At compilation time: - flexible, + strong type-checking. These are languages such as C++.
  - In execution: + flexible, - strong type checking. These are languages such as SmallTalk, Python ...
- It is still posible an intermediate language, dynamic as Self, but doing compile time checking. This is the case of Kevo.

Prototype-based programming languages

Creation of a prototype-based programming language

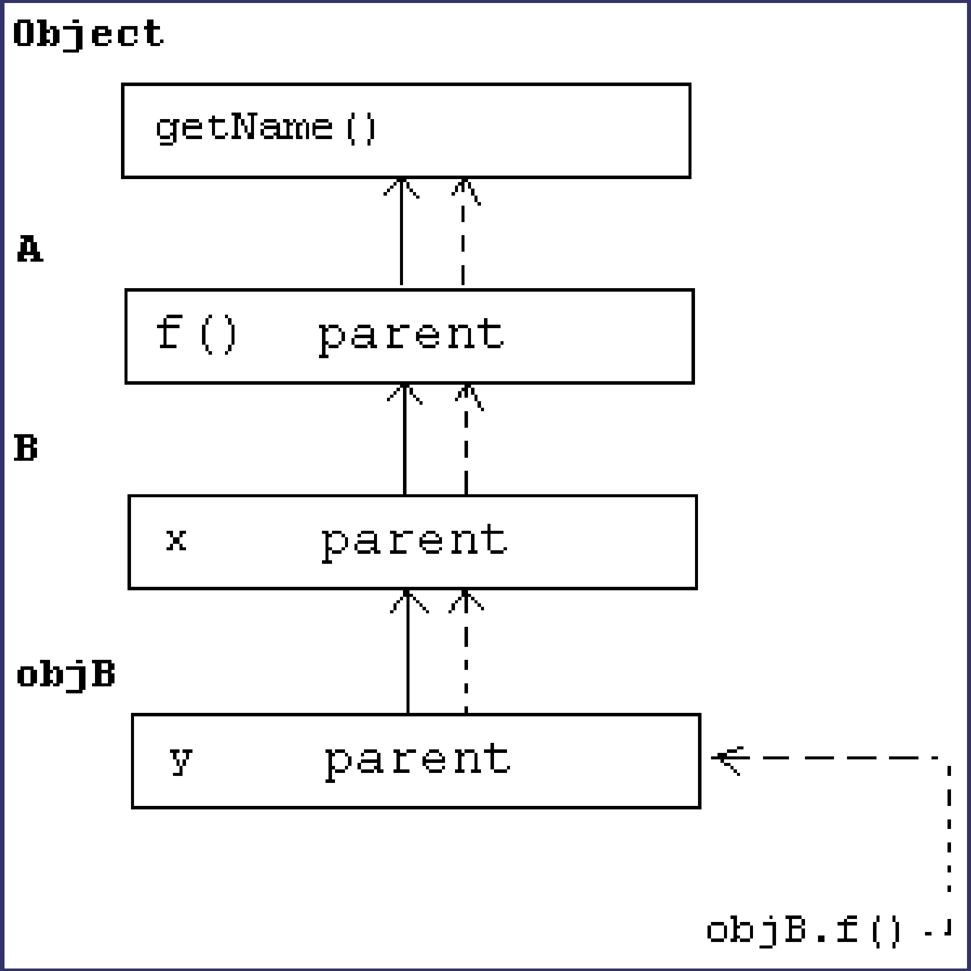
Prototype-based programming language

- There are no classes, objects are created by copying other objects.
- The objects that are copied are called *prototypes*. However, a big difference is that new objects can be modified independently from their prototypes.
- It's a model very flexible and simple.
- It is able to represent the class-based model.

Implementation

- As objects do not depend of a class, and are independent of the prototype it was copied from, the structure of the object must contain state and behaviour (attributes and methods).
  - Methods and attributes are contained in a set, in the same space in memory.
- Inheritance is implemented by delegation, although special programming languages such as Kevo prove that this is not mandatory.

Schematic representation of objects in memory



- Objects contain methods and attributes.
- When a message cannot be satisfied, it is delegated in its parent.
- Inheritance, by delegation, can be flexible and therefore dynamic.

Conclusions

Object-oriented programming models

- There are two extremes in the spectrum of object-oriented programming models.
  - The more restrictive model is the class-based one.
    - Strong compile-time tpye-checking.
    - Inheritance by concatenation.
  - The more flexible model is the prototype-based one.
    - There are not many compile-time verifications.
    - Inheritance by delegation.
- However, intermediate object-oriented programming models are possible (for example, the programming language Kevo).

Implementation

- The characteristics that thus condition the implementation of a programming language:
  - Existence of classes.
  - Kind of inheritance.
- However, implementation can be separated in many layers.
  - Zero is a virtual machine that implements the prototype-based model. However, there are compilers that generate *bytecode* to be consumed by the virtual machine. One of them, J-- does compile-time type-checking, while the other one PROWL, is a pure prototype-based programming language.

References

- Bjarne Stroustrup, designer of C++:
  - Personal web page
    - http://www.research.att.com/~bs/
  - “The C++ programming language”
    - http://www.research.att.com/~bs/3rd.html
  - “Design and evolution of C++”
    - http://www.research.att.com/~bs/dne.html
  - Other publications:
    - http://www.research.att.com/~bs/books.html
- SmallTalk
  - Squeak (current implementation): http://www.squeak.org/
  - SmallTalk documentation: http://www.esug.org/

References

- Self
  - Web page: http://research.sun.com/self/index.html
  - Implementation: http://research.sun.com/research/self/papers/elgin-thesis
  - Other: http://research.sun.com/research/self/papers/papers.html
- Kevo
  - http://burks.brighton.ac.uk/burks/foldoc/44/63.htm
- Python
  - Class-based languages, implemented with prototypes.
  - Web page: http://www.python.org/

References

- Looking for papers:
  - http://www.researchindex.com
  - http://scholar.google.com

Doctoral course: object technologies

IMO Group
Department of computer science
University of Vigo

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