Doctoral course: Object Technologies

IMO Group
Computer Science Department
University of Vigo

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

The prototype-based, object-oriented model

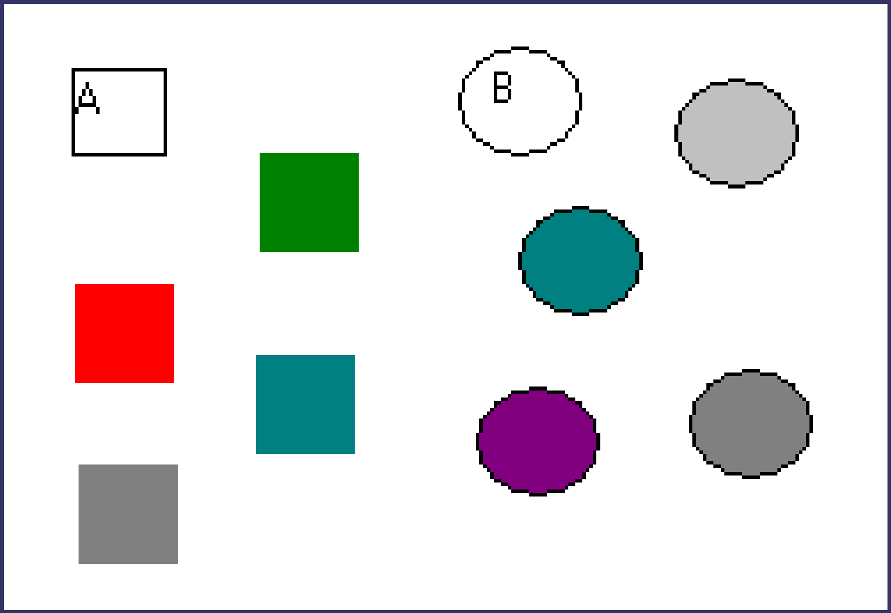
Prototype-based object orientation

- There are two main groups:
  - Class-based object-oriented programming languages: C++, Object Pascal, Java, Eiffel ... they heavily used in industry.
  - Prototype-based object-oriented programming languages: Self, Kevo, Poet/Mica, Cecil ... all of them are expermimental, thus they are not used in industry.
- **Actually, the prototype-based model is a superset of the class-based model.**

Terminology

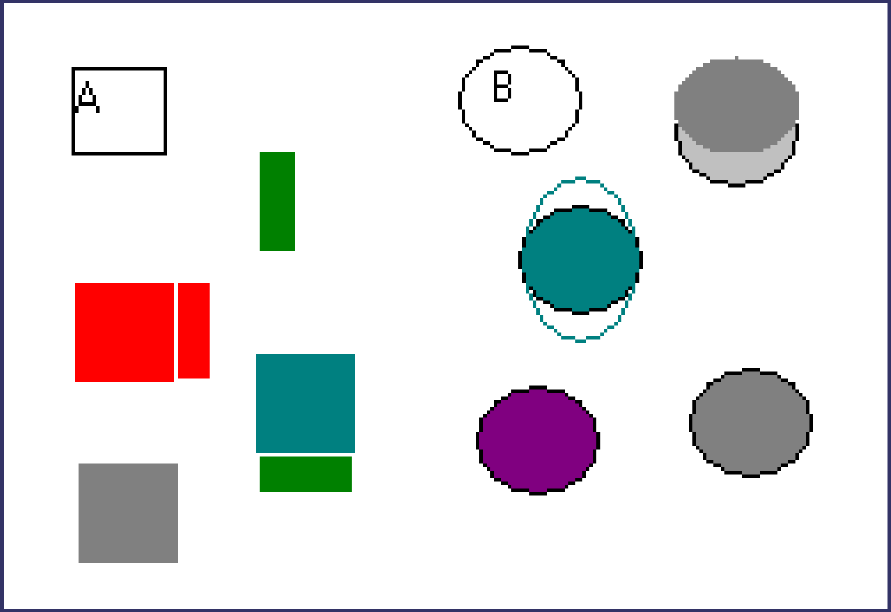
- ***State***: the *attributes* (SmallTalk terminology) or *member data (C++ terminology)* of an object. For instance, the color, power ... of a car.
- ***Behaviour***: *methods* (SmallTalk) or *member functions* (C++) of an object. For instance, turn-ing on, accelerate or brake a car.
- **Message**: execution of an object's method. If an object has a method f(), sending the message f to O has the same meaning as executing O.f()

Class-based object orientation



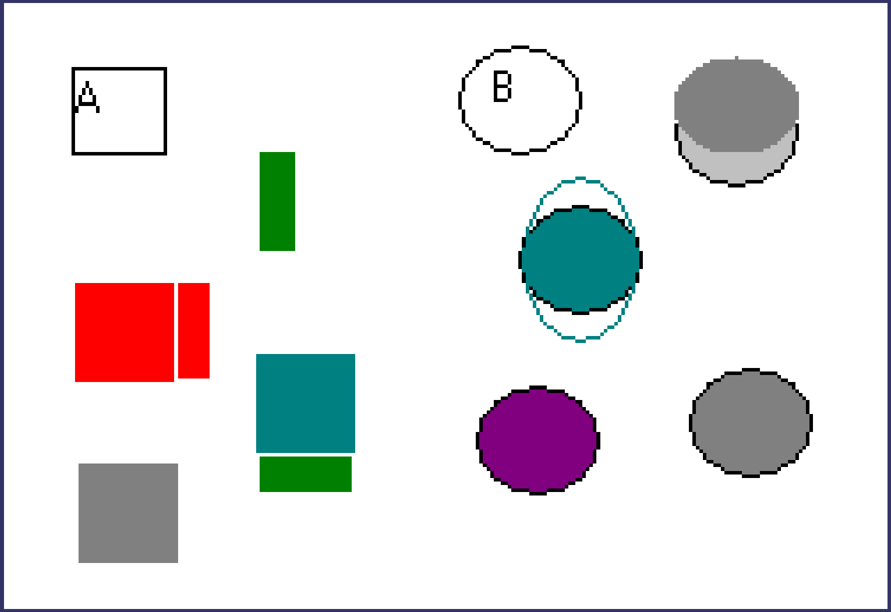
- A class is a *type* of objects, a mould from which new objects are obtained, sharing the same behaviour, changing only their state.

Prototype-based object orientation



- There are no **classes**. All objects are equal, they pertain to the same category.
- New objects are copied from other existing ones. Some of them are **prototypes**.

Prototype-based object orientation

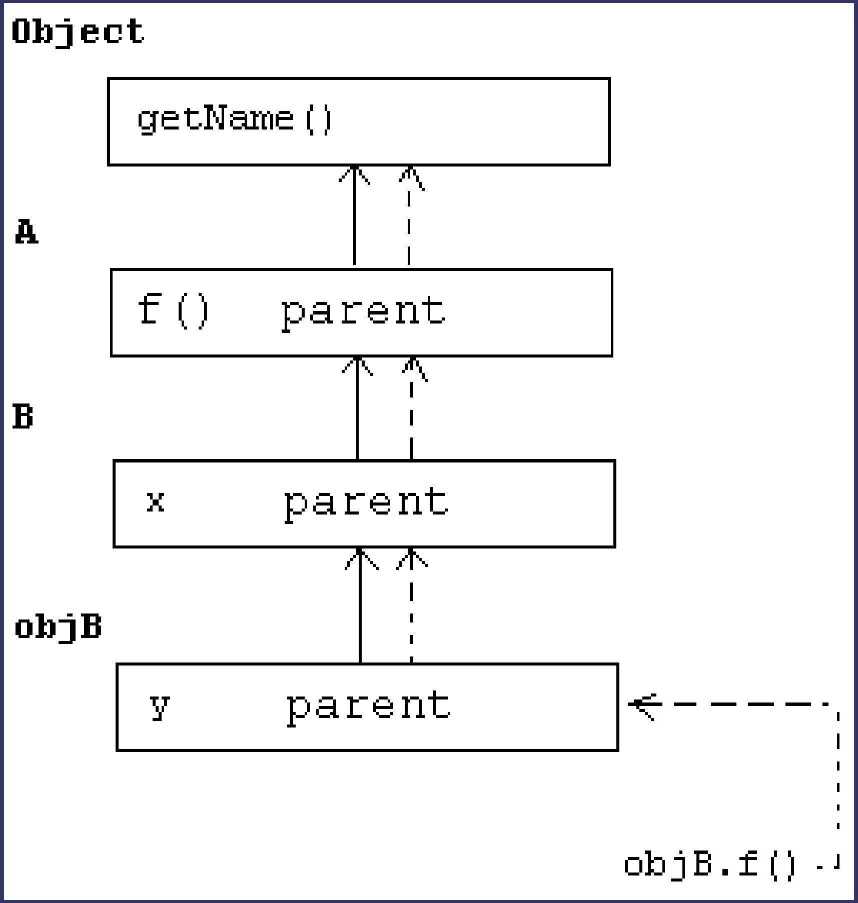


- Normally, in this kind of languages objects can be modified, appending or deleting methods and attributes.
- Every object is independent, self-describing, without any need of extra information.

Inheritance

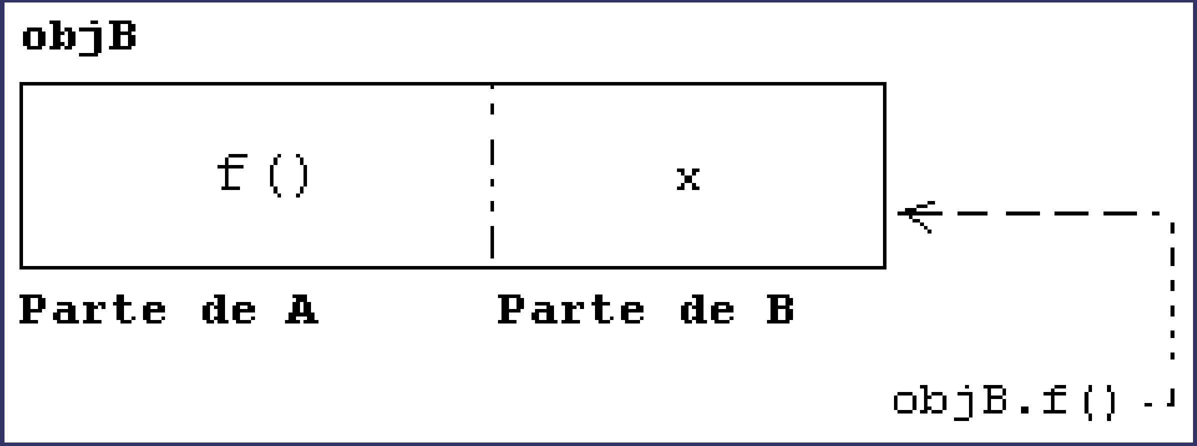
- Inheritance in prototype-based programming languages is implemented by delegation.
  - The object has one or more parent attributes, so, when it is unable to answer a message, it deleg-ates it to one of its parents.
- In class-based programming languages, in-heritance is implemented by concatenation.
  - The object is composed by the parts defining each one of the classes it inherits from.

Inheritance implemented by delegation



- There is a chain of objects pointing to their parents, linking each other until reaching the inherit-ance root.

Inheritance implemented by con-catenation



- All attributes and methods are available in the same object (however you need the class in order to interpret them).

Answering messages

- When an object is unable to answer a message, because it does not have the required method, it resends the message to the object pointed by its parent attribute.
- The inheritance relations, instead of being a special relation, are an specific case of composition relations.

Answering messages

- For instance, given the following objects:

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

object B : A
endObject
```

Answering messages

- The message:

```
        B.foo(); // MSG B foo
```

- Does not find the foo method in the B object, so it follows the parent attribute, pointing to A, which does have the method, and is finally executed.
- If it were not found, then an error would raise. which normally would be an exception. In this case, the exception would be the *method not found* one.

Object creation

- Since new objects are created by copying them, it is not necessary to have constructors found in class-based programming languages.
- Objects do not define the data types of attributes, as classes do: they directly store an initial value.

Object creation

- For instance:

```
object Person
  attribute + name = “John”;
  attribute + surname = “Doe”;
  attribute + telephone = “906414141”;
  attribute + age = 18;
  attribute + address;
  method + toString() {
    reference toret;
    toret = name.concat( surname );
    return toret
  }
endObject
```

Creación de objetos

- The *Person* object will be used to create new objects, though there are no differences between this object and any other one.
- For instance (in assembler):

```
reference paula =
   Persona.copy( “PaulaMarquez” );
```

- If this message *copy* is sento to the Person object, then an exact copy will be created, with the name “PaulaMarquez”. Its attributes will need of modification.

Creación de objetos

- Creating an object from *Person*:

```
reference paula =
  Persona.copy( "PaulaMarquez" );

paula.setName( "Rose Mary" );
paula.setSurname( "Ventura" );
paula.setTelephone( "988343536" );
...
```

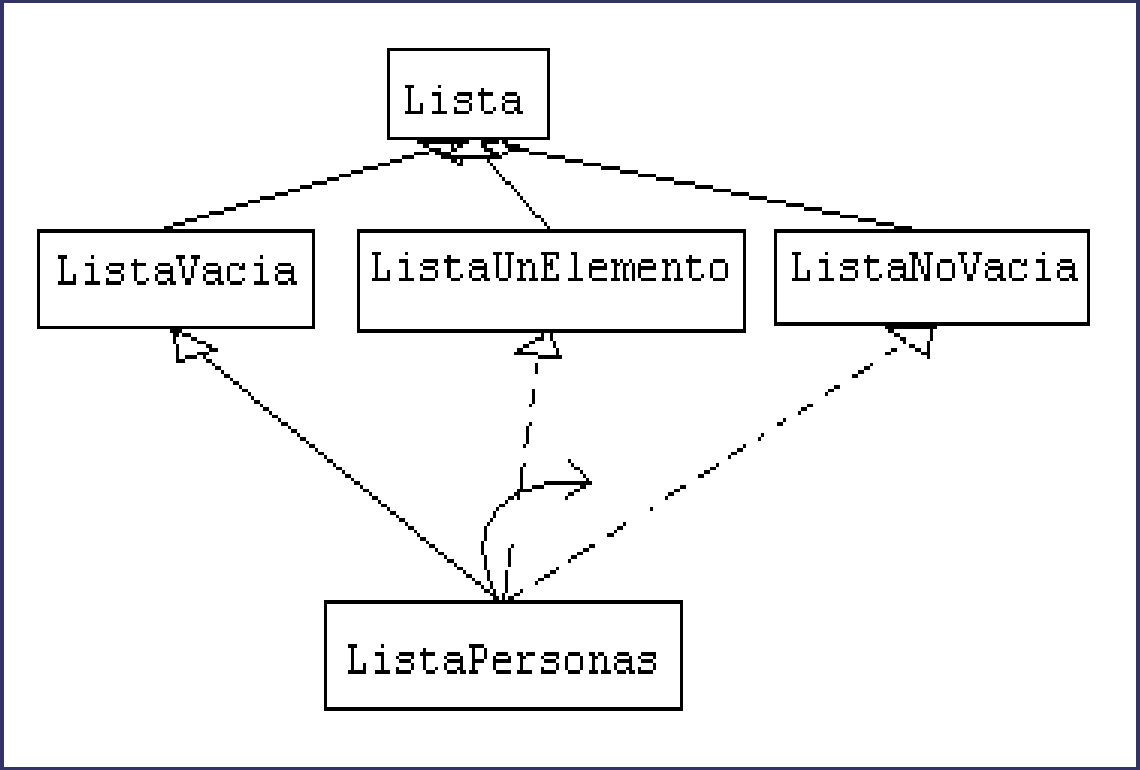
El modelo de prototipos incluye al de clases

- Those objects playing the role of prototypes are the equivalent to classes in class-based programming languages.
- The only diifference is that this model is much more flexible than the class-based one.
- Even a class can be modified, as in this model it is just an object.
- Delegation is very flexible mechanism, sep-arating objects from their prototypes, as in class-based programming languages, but not behaviour from state.

Dynamic inheritance

- In the case that inheritance is implemented by delegation, another possibility appears: the fact of being able to change an attribute, (as it is just an average attribute), pointing to the parent object, means that an object can have multiple parents, in sucesion, depend-ing in the specific moment.
- Making use of this characteristic requires changing the programming style.

Dynamic inheritance



- It is possible to change the parent of an object at run time.
- It totally contrasts to the behaviour of static languages, which leave it everything linked at compile time.

Dynamic inheritace

- The *insert* method of *EmptyList*:

```
object EmptyList : List
 method + insert(obj) {
   parent.insertarPrimero( obj );
   parent = OneElementList;
   return;
  }
 method + getElementNumber(n) {
   return Nothing;
  }
endObject
```

Dynamic inheritance

- The insert method of OneElementList:

```
object OneElementList : List
  method + insertar(n, obj) {
    super( 1, obj );
    parent = NonEmptyList;
    return;
  }
  method + getElementoNumero(n) {
    return
      parent.getPrimerElemento();
  }
endObject
```

Dynamic inheritance

- The erase method of NonEmptyList (which is also needed in the other two objects):

```
object NonEmptyList : List
 method + borrar(n) {
   super( n );
   if ( this.size() < 2 ) {
     parent = OneElementList;
   }
   return;
 }
endObject
```

Dynamic inheritance

- Its main advantage is that methods can be written following the kind of object. In EmptyList, it is not necessary to have the size() method to actually read the number of elements of the list, as the list is empty, and zero can be returned directly. The resulting source code is simpler.
- Its main disadvantage is that it is needed to coordinate various objects in order to perform some tasks. This is error-prone.

Dynamic inheritance

- Using Prowl, the inheritance coordination can be achieved this way:

```
object myList :
EmptyList( this.size() == 0),
OneElementList( this.size() == 1 ),
NonEmptyList( this.size() > 1);
```

- The system is in charge of changing parents.

Prototype-based object-oriented programming languages

- Self
  - It was created in the Sun Research Laboratories, and has served as an essay for Java.
  - It was the first one to implement the prototype-based model, invented also by Sun. Self tries to be as dynamic as possible, avoiding compile-time checkings.
  - http://research.sun.com/research/self/
- Io
  - It was created by one of the developers of Self, following its main guidelines.
  - http://www.iolanguage.com/

Prototype-based object-oriented programming languages

- Kevo
  - Kevo was developed for a doctoral thesis. as a demonstration of how a prototype-based object-oriented language could implement inheritace by concatenation, and compile-time checkings.
  - It is less flexible at runtime than *Self* o *Io*.
  - ftp://cs.uta.fi/pub/kevo

Prototype-based object-oriented programming languages

- Cecil
  - Cecil was developed at the University of Wash-ington. It includes the concept of predicate ob-jects, using dynamic inheritance to take the concept of programming by contracto to the ob-ject-level.
  - It is a prototype-based programming language.
  - http://www.cs.washington.edu/research/projects/cecil/www/cecil.html

Prototype-based object-oriented programming languages

- Zero. Deveoped at the University of Vigo.
  - Two high-level programming languages are available.
  - Simple, prototype-based.
  - It implements also transparent persistence.
  - http://trevinca.ei.uvigo.es/~jgarcia/TO/zero/

Prototype-based object-oriented programming languages

- Other languages:
  - http://www.programming-x.com/programming/prototype-based.html

References

- About Self and the prototype-based model:
  - 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.

References

- About developing applications with Self:
  - 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

References

- About Kevo and the prototype-based model:
  - 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: Delegation versus Concatenation or Cloning is Inheritance too. *OOPS Messenger 6*(3): 20-49 (1995)
  - Taivalsa, 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

Doctoral course: Object Technologies

IMO Group
Computer Science Department
University of Vigo

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