PROG-3

SUBTOPIC 1

INTRODUCTION TO THE OBJECT ORIENTED PARADIGM

Pedro J. Ponce de León
Translated into English by Juan Antonio Pérez

Version 0.8

lsi Departamento de Lenguajes y Sistemas Informáticos

Universitat d'Alacant
Universidad de Alicante

Contents

- **Evolution of the concept of abstraction**
  - Abstraction definition
  - Programming languages and abstraction layers
  - Main programming paradigms
  - Abstraction mechanisms in programming languages

- The object oriented paradigm
  - Object oriented languages (OOL); main features
  - OOL: optional features
  - History of OOL
  - Objectives of object oriented programming (OOP)

Evolution of the concept of abstraction
Definition

- **Abstraction** (from Wikipedia)
  - *Abstractions may be formed by reducing the information content of a concept or an observable phenomenon, typically to retain only information which is relevant for a particular purpose.*

- Some information is shown and some other is purposely hidden at each layer (level) of abstraction.
  - Example: different scales on a map.

- By means of abstraction, different REALITY MODELS are created.
- A critical problem is finding the right level of abstraction.

Evolution of the concept of abstraction
Programming languages and abstraction layers

- Layers of abstraction depend on the mechanisms offered by a particular programming language.
- OOP may be seen as a natural evolution of different abstractions.

- Assembly
- Procedures
- Modules

Functional perspective

- Packages
- Abstract data types (ADT)

Data perspective

- Objects
  - ADT
  - + message passing
  - + inheritance
  - + polymorphism

Service perspective

Evolution of the concept of abstraction

Programming languages and abstraction layers

- Programming languages offer abstractions

Problem space

Object oriented languages (OOL)

Pure OOL

Smalltalk, Eiffel

Solution space

Assembly

Imperative languages (C, Fortran, BASIC)

Less general-purpose languages (LISP, PROLOG)

Hybrid OOL (Multiparadigm)

C++, Object Pascal, Java,...

Evolution of the concept of abstraction
Main paradigms

- **PARADIGM**
  - A generally accepted world view (Oxford English Dictionary)
  - A set of assumptions, concepts, values, and practices that constitutes a way of viewing reality for the community that shares them, especially in an intellectual discipline (The Free Dictionary)
  - Any example or model.

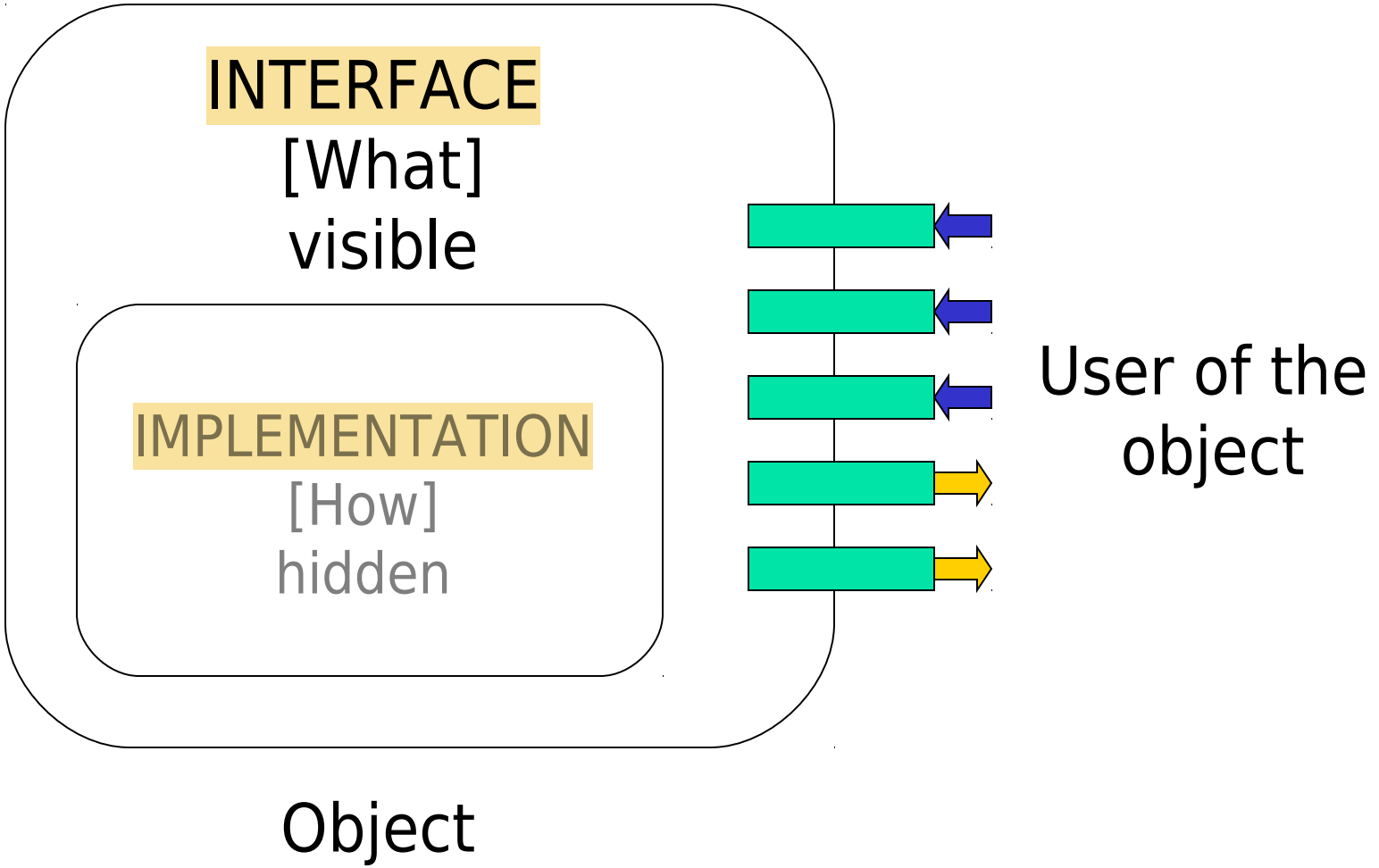
- A **programming paradigm** is a way of conceptualizing what it means to perform computation. Main programming paradigms:
  - Functional paradigm: language describes processes
    - Lisp (abstraction: everything is a list) and its dialects (e.g. Scheme), Haskell, ML
  - Logical paradigm
    - Prolog
  - Imperative (or procedural) paradigm
    - C, Pascal
  - *Object oriented paradigm* (abstraction: a program is a collection of interacting objects)
    - Java, C++, Smalltalk, ...

Evolution of the concept of abstraction
Abstraction mechanisms in programming languages

- **Information hiding**
  - Ability to prevent certain aspects of a class or software component from being accessible to its clients, through an explicit exporting policy. (Wikipedia)
  - Purposeful omission of details in the development of an abstract representation. (T. Budd)

- When a strict division holds between the inner view of a component (object) and the outer view, the term **ENCAPSULATION** is used.
  - These two views comprise:
    - **INTERFACE**: WHAT the object can do (outer view)
    - **IMPLEMENTATION**: HOW it is done (inner view)
  - A software module hides information by encapsulating the information into a module or other construct which presents an interface (Wikipedia).
  - Replacement of components, communication among members of the development team, and interconnection of independently developed parts of the application are made easier.

Evolution of the concept of abstraction
Abstraction mechanisms in programming languages



Contents

- Evolution of the concept of abstraction
  - Abstraction definition
  - Programming languages and abstraction layers
  - Main programming paradigms
  - Abstraction mechanisms in programming languages

- **The object oriented paradigm**
  - Object oriented languages (OOL); main features
  - OOL: optional features
  - History of OOL
  - Objectives of object oriented programming (OOP)

The object oriented paradigm

- Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of **objects**, each of which represents an **instance** of some **class**, and whose classes are all members of a **hierarchy** of classes united via **inheritance** relationships. (Grady Booch)

- OOP changes:
  - How a program is organized:
    Classes (data+operations on data).
  - How a program is run
    Message passing

- True object oriented programming means far more than using an object oriented language. The object oriented paradigm must be followed.

The object oriented paradigm
Why OOP is so popular?

- OOP has become the dominant paradigm in application development for the last two decades and a solution to overcome the so called *software crisis* (hardware costs fall but software development keeps on being difficult and expensive)

- Reasons behind this domination:
  - OOP scales very well.
  - The OO paradigm relies on a set of principles that resonate with techniques people use to solve problems in their everyday lives (metaphors)
    - Smalltalk was initially conceived as a programming language for children of all ages.
    - Alan Kay found that it was easier to teach Smalltalk to children than to computer professionals because of their preconceptions.
  - Lot of tools (IDEs, libraries, etc.) available in different domains.
  - But OOP is not a panacea.

The object oriented paradigm
A new way of viewing the world

- Example (by T. Budd): Luis wants to send some flowers to Alba, who lives in a city many miles away.
  - Luis goes to the nearest flower shop, run by a florist called Pedro.
  - Luis tells Pedro about the variety and quantity of flowers that he wishes to send and Alba's address

- The mechanism for solving this problem consists of
  - Finding an appropriate **agent** (Pedro)
  - Passing an appropriate **message** with the request (send flowers to Alba)
  - Pedro's responsibility is to satisfy the request
  - Possibly, Pedro has some **method** (an algorithm or a list of operations) to perform it

- Luis does not need (indeed, he probably does not want to) to know the particular method used by Pedro to satisfy the request: this information is HIDDEN from inspection (but it possibly involves more agents, messages and methods).

- The solution to the problem needs the cooperation of several individuals.

- If problems are defined in terms of responsibilities, the level of abstraction increases and stronger independence is attained between objects.

The object oriented paradigm
A new way of viewing the world

A world composed of:

- Agents and communities
- Messages and methods
- responsibilities
- Objects and classes
- Class hierarchies
- Method linking

The object oriented paradigm
A new way of viewing the world

- **Agents and communities**
  - An object oriented program is structured as a community of interacting agents, called objects. Each object has a role to play. Each object provides a service, or performs an action, that is used by other members of the community.

The object oriented paradigm
A new way of viewing the world

- **Messages and methods:**
  - A message is transmitted to an object.
  - The object selects the most appropriate method to satisfy the request.
  - This is known as **message passing**
  - Syntax of a message:

receiver.selector(arguments)

```
unJuego.mostrarCarta(laCarta,42,47)
```

The object oriented paradigm
A new way of viewing the world

- **Messages and methods**
  - A message is different from a procedure call in at least two things:
    - A message has a designated receiver (at least, one parameter)
    - The interpretation of the message may vary with different receivers.
      - A procedure name is assigned 1:1 with the code which will be run, whereas this is not the case with a message (for instance, if the specific receiver is not known until run time, that is, if there is late binding between the message and the code fragment used to respond to it).
    - Example:

```
JuegoDeCartas juego = new Poker … or … new Mus … or …
juego.repartirCartas(numeroDeJugadores)
```

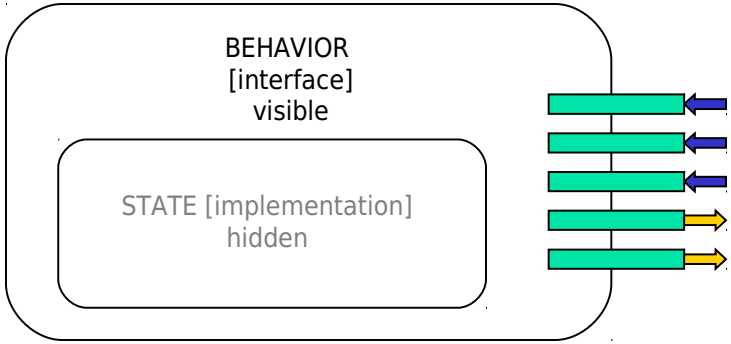
The object oriented paradigm
A new way of viewing the world

- **Responsibilities**
  - Object behavior is defined in terms of responsabilities
    - Greater independence between objects
  - **Protocol**: entire collection of responsabilities of an object
  - OOP vs. imperative programming

Ask not what you can do to your data structures
Ask what your data structures (objects) can do for you.

The object oriented paradigm
A new way of viewing the world

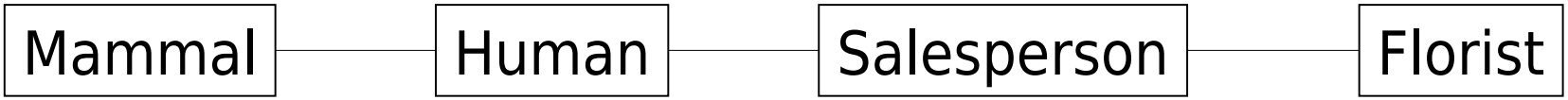
- **Objects and classes**
  - An object **encapsulates** a **state** (data values) and a **behavior** (operations).



  - Objects group into categories (**classes**).
    - An object is an **instance** of a class.
    - Pedro is an instance of the Florist category (class)
    - The method executed by an object in response to a message is determined by the class of the receiver object.

The object oriented paradigm
A new way of viewing the world

- **Class hierarchies**
  - In everyday life, things and concepts may be classified in the form of hierarchies. **Generalization** of a concept is an extension of the concept to less-specific criteria.
  - Generalization in OOP is implemented as **inheritance**, which is the principle that knowledge of a more general category is also applicable to a more specific one.
    - Pedro, as a florist, is also a salesperson; salesperson is a generalization of florist because every florist is a salesperson , and there are salespersons who are not florists
    - A salesperson is a human
    - Humans are mammals
  - The inheritance relationships of classes give rise to a hierarchy. A child class inherits attributes and behavior of a (more general) parent class:



The object oriented paradigm
A new way of viewing the world

- **Method binding**

  The time at which a method call is associated with the code to be run.

  - **Static** (or early) binding: at compile time
  - **Dynamic** (or late) binding: at run time

- Let's assume that in the following example the real assignment to variable ‘juego’ depends on some interaction with the user (at run time):

```
JuegoDeCartas juego = new Poker … or … new Mus … or …
juego.repartirCartas(numeroDeJugadores)
```

Message ‘repartirCartas’ should have dynamic binding.

Contents

- Evolution of the concept of abstraction
  - Abstraction definition
  - Programming languages and abstraction layers
  - Main programming paradigms
  - Abstraction mechanisms in programming languages

- The object oriented paradigm
  - **Object oriented languages (OOL); main features**
  - OOL: optional features
  - History of OOL
  - Objectives of object oriented programming (OOP)

Main characteristics of a OOL

- As identified by Alan Kay (1993):

(1) Everything is an **object.**

(2) Every object is built from other objects.

(3) Every object is an **instance** of a **class.**

(4) All the objects that are instances of the same class can perform the same actions (receive the same messages). The class is the repository for **behavior** associated with an object.

(5) Classes are organized into a singly rooted tree structure, called the inheritance hierarchy.

As a circle is a shape, a circle will always accept all the messages that could be sent to a shape.

(6) A program is a set of objects communicating with each other by sending and receiving **messages**.

Optional characteristics of an OOL (1/3)

- **Polymorphism**
  - A feature of a variable that can take on values of several different types at different execution times.

    E.g., dynamic binding
- **Genericity**
  - Use of generic classes, which include at least one to-be-specified-later type (*templates in C++, generics in Java*).

    E.g.: List<T> : where T can be any type.
- **Error management**
  - Error conditions are managed with **exceptions**
- **Assertions**
  - Predicates which indicate *what* software does instead of *how*. They can function as a form of documentation.
    - **Preconditions**: they describe the state the code expects to find before it runs
    - **Postconditions**: they describe the state the code expects to result in when it is finished running
    - **Invariants**: assertions describing permanent restrictions

Optional characteristics of an OOL (2/3)

- **Static typing**
  - Type checking is performed during compile-time
    - Specifically, the compiler ensures that an object understands the messages sent to it.
  - It allows many type errors to be caught early in the development cycle (before run time)

- **Garbage collection**
  - The run-time system automatically recycles the memory which will be no longer used.

- **Concurrency**
  - It allows different objects to execute their methods at the same time by using different threads.

hilo

Optional characteristics of an OOL (3/3)

- **Persistence**
  - A persistent object continues to exist outside of the execution time of programs that manipulate the object.
    - It usually implies that a database is used to store the state of the object.

- **Reflection**
  - Reflection is the process by which a computer program can observe (do type introspection) and modify its own structure and behavior at runtime.
    - Normally, instructions are executed and data is processed.
    - However, in some languages, programs can also treat instructions as data and therefore make reflective modifications:

```
String instr = “System.out.println(”;
ejecuta(instr + ”27)”);
Class c = Class.forName("String");
Method m = c.getMethod("length", null);
m.invoke(instr,null);
```

OOP
Optional characteristics of an OOL: conclusions

- Ideally, a language should provide as many of the previous features as possible
  - Object oriented is not a black or white concept: a particular language can be 'more' object oriented than another one.

Contents

- Evolution of the concept of abstraction
  - Abstraction definition
  - Programming languages and abstraction layers
  - Main programming paradigms
  - Abstraction mechanisms in programming languages

- The object oriented paradigm
  - Object oriented languages (OOL); main features
  - OOL: optional features
  - **History of OOL**
  - Objectives of object oriented programming (OOP)

History of OOL

Year	**Language**	Designers	Comments
1967	**Simula**	Norwegian Computer Center	***class, object, encapsulation, simulation of real life systems***
1970s	**Smalltalk**	Alan Kay	***Methods, message passing, dynamic binding,inheritance***
1985	**C++**	Bjarne Stroustrup	Bell Labs. C extension. Great commercial success (1986->)
1986	**First OOPSLA Conference**		**Objective C, Object Pascal, C++, CLOS**,… Extensions of non-OO languages (C, Pascal, LISP,…)
‘90s	**Java**	Sun	OOP becomes the dominant paradigm. Java: execution on a virtual machine.
‘00->	**C#, Python, Ruby,…**		More than 170 OO languages. TIOBE index (top 10 colonized by OOL)

History of OOL: **present time**

- Great success of OO technologies and languages from the nineties.
- Currently, the most used languages include **Java**, **C++** y **PHP** (TIOBE index)
- **C#, Python, Objective-C** are very used as well.
- Hybrid languages (OO/procedural): PHP, C++, Visual Basic, JavaScript
- More: Delphi, Ruby, ActionScript,...

Contents

- Evolution of the concept of abstraction
  - Abstraction definition
  - Programming languages and abstraction layers
  - Main programming paradigms
  - Abstraction mechanisms in programming languages

- The object oriented paradigm
  - Object oriented languages (OOL); main features
  - OOL: optional features
  - History of OOL
  - **Objectives of object oriented programming (OOP)**

Objects of OOP
Software quality (Bertrand Meyer)

- The main goal of the increment in abstraction attainable wit OOP is
  - IMPROVE THE QUALITY OF APPLICATIONS
  - “Engineering seeks quality; software engineering is the production of quality software.”
- Software quality is described as a combination of several factors
  - **INTERNAL FACTORS**
  - **EXTERNAL FACTORS**

Objectives of OOP
External factors

- **Reliability: correctness** + **robustness**:
  - **Correctness** is the ability of software products to perform their exact tasks, as defined by their specification.
  - **Robustness** is the ability of software systems to react appropriately to abnormal conditions.
- The pursuit of this factors is the central task of object-oriented software construction.
- If a system does not do what it is supposed to do, everything else about it — whether it is fast, has a nice user interface... — matters little.
- Robustness complements correctness. Correctness addresses the behavior of a system in cases covered by its specification; robustness characterizes what happens outside of that specification.

Objectives of OOP
Internal factors

- **Modularity: extendibility** + **reusability**:
  - **Extendibility** is the ease of adapting software products to changes of specification.
    - Design simplicity is essential for improving extendibility.
  - **Reusability** is the ability of software elements to serve for the construction of many different applications.
    - The need for reusability comes from the observation that software systems often follow similar patterns.
- Software **maintenance**, which consumes a large portion of software costs, is penalized by the difficulty of implementing changes in software products.

Bibliography

- Cachero et. al.
  - ***Introducción a la programación orientada a objetos***
    - Chapter 1
- Timothy Budd
  - ***An introduction to OO Programming, 3rd edition***
    Addison Wesley, 2002
    - Chapters 1 and 2
- Bertrand Meyer
  - ***Object Oriented Software Construction***
- Bruce Eckel
  - ***Thinking in Java, 4th edition***
    (Thinking in C++ / Thinking in Java, online)
    - *Chapter 1*