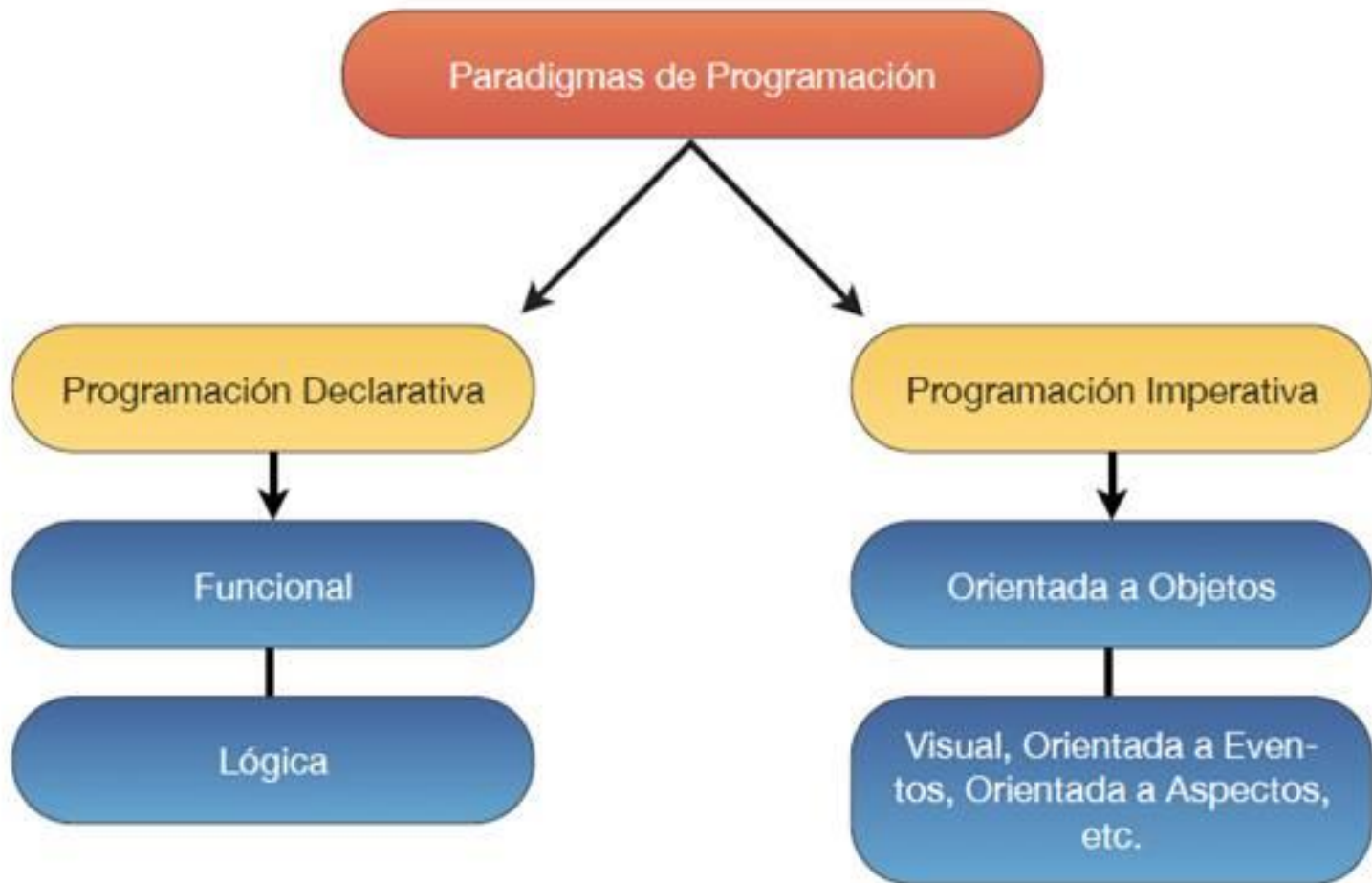


Paradigmas de programación

Profesor Francisco Alejandro Medina

Paradigmas de programación



Paradigmas de programación

- **Imperativo**
 - modelo de Von Neuman, cuello de botella de Von Neuman
- **Orientado a Objetos**
 - TDAs, encapsulación, modularidad, reutilización
- **Funcional**
 - noción abstracta de función, cálculo lambda, recursividad, listas
- **Lógico**
 - Lógica simbólica, programación declarativa

Modelo Imperativo

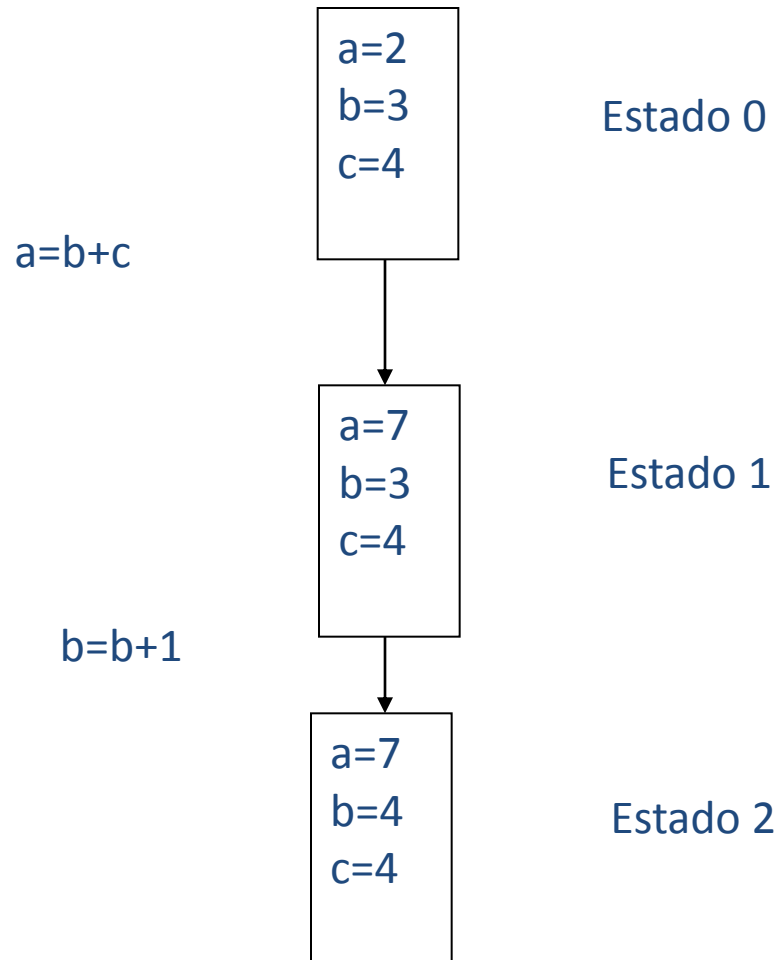
- Describe la programación como una secuencia instrucciones o comandos que cambian el estado de un programa.
- El código máquina en general está basado en el paradigma imperativo.
- Su contrario es el paradigma declarativo.
- En este paradigma se incluye el paradigma procedimental (procedural) entre otros.

Ejemplo

Programa

$a = b + c$

$b = b + 1$



Ejemplo Programación Imperativa

```
#include <iostream>
using namespace std;
int main ()
{
    int a,b;
    cout << "Ingrese el primer número:";
    cin >> a;
    cout << "Ingrese el segundo numero:";
    cin >> b;
    if (a==b)
        cout << "Los números son iguales"<< endl;
    else
        if (a > b)
            cout << a << " es mayor"<< endl;
        else
            cout << b << " es mayor" << endl;
    return 0;
}
```

Dev-C++ 5.2.0.0 - [Robotbesturing] - Robotbesturing.dev

File Edit Search View Project Execute Debug Tools CVS Window Help

(globals) CharArrayToSignal(const char* in) : unsigned ct

Project Classes Debug

main.cpp resource.h resource.rc

```
62 unsigned char CharArrayToSignal(const char* in) {
63     unsigned char result = 0;
64     for(int i = 0; i < 8; i++) {
65         if(in[7-i] == '1') {
66             result += pow(2,i);
67         }
68     }
69     return result;
70 }
71
72 bool WriteByte(const char byte) {
73     if(ComHandle != INVALID_HANDLE_VALUE) {
74         DWORD byteswritten = 0;
75         if(!WriteFile(ComHandle,&byte,1,&byteswritten,NULL)) {
76             LogWrite("Error writing byte\r\n");
77             MessageBeep(MB_ICONERROR);
78             return false;
79         } else {
80             LogWrite("Success!\r\n");
81             return true;
82         }
83     } else {
84         LogWrite("Can't send data to broken COM!\r\n");
85         MessageBeep(MB_ICONERROR);
86         return false;
87     }
88 }
89
90 void UpdateSignal() {
91     char signal[9] = "00000000";
92     if(turbo) {
```

Line: 62 Col: 1 Sel: 0 Lines: 446 Length: 14387 Insert Done parsing in 0.16 seconds



Modelo Declarativo

- No se basa en el cómo se hace algo (cómo se logra un objetivo paso a paso), sino que describe (declara) cómo es algo.
 - En otras palabras, se enfoca en describir las propiedades de la solución buscada, dejando indeterminado el algoritmo (conjunto de instrucciones) usado para encontrar esa solución.
 - Es más complicado de implementar que el paradigma imperativo, tiene desventajas en la eficiencia, pero ventajas en la solución de determinados problemas.

Programación Declarativa

- Usa bloques de construcción como las funciones, la recursión o la equipación de patrones, para especificar más la solución que su cálculo de bajo nivel.
- Tipos:
 - Lenguajes funcionales
 - Lenguajes lógicos

Programación Funcional

- Usan funciones libres de efectos secundarios como bloques primitivos de construcción de programas.
 - Estas funciones pueden aplicarse, construirse y pasarse como argumento a otras funciones.
- Concibe a la computación como la evaluación de funciones matemáticas y evita declarar y cambiar datos.

Programación Funcional

- En otras palabras, hace hincapié en la aplicación de las funciones y composición entre ellas, más que en los cambios de estados y la ejecución secuencial de comandos (como lo hace el paradigma procedimental).
- Permite resolver ciertos problemas de forma elegante y los lenguajes puramente funcionales evitan los efectos secundarios comunes en otro tipo de programaciones.
 - Haskell, Miranda, Scala, Lisp, Scheme, Ocaml, SAP, Standard ML, Erlang, R, F#

Ejemplo Programación Funcional

```
1  > (define (responder-saludo s)
2      (if (string? s)
3          (if (equal? "hola" (substring s 0 4))
4              "¡hola, gusto de verte!"
5              "¿perdón?"
6          )
7          "perdón, ¿qué?"
8      )
9  )
10 > (responder-saludo "hola programa")
11 "¡hola, gusto de verte!"
12 > (responder-saludo 3.1416)
13 "perdón, ¿qué?"
14 > (responder-saludo "El día está muy bonito, ¿verdad?")
15 "¿perdón?"
```

Ejemplo Dr. Racket

```
File Edit View Language Racket Insert Tabs Help
Untitled-1 (define ...) Save [icon] Check Syntax [icon] Debug [icon] Macro Stepper [icon] Run [icon] Stop [icon]

(define first car)
(define rest cdr)

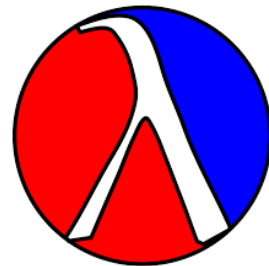
(define (addWithCarry x y carry)
  (cond
    ((and (null? x)(null? y)) (if (= carry 0) '() '(1)))
    ((null? x) (addWithCarry '(0) y carry))
    ((null? y) (addWithCarry x '(0) carry))
    (#t (let ((bit1 (first x))
              (bit2 (first y)))
          (cond
            ((= (+ bit1 bit2 carry) 0) (cons 0 (addWithCarry (rest x) (rest y) 0)))
            ((= (+ bit1 bit2 carry) 1) (cons 1 (addWithCarry (rest x) (rest y) 0)))
            ((= (+ bit1 bit2 carry) 2) (cons 0 (addWithCarry (rest x) (rest y) 1)))
            (#t (cons 1 (addWithCarry (rest x) (rest y) 1)))))))

(define (multBins x y)
  (cond
    ((null? y) '() )
    ((= (first y) 0) (multBins (cons 0 x) (rest y)))
    (#t (addWithCarry x (multBins (cons 0 x) (rest y)) 0))))

(multBins '(1 0 1 1)'(1 1 0 1))

Welcome to DrRacket, version 5.3 [3m].
Language: R5RS; memory limit: 128 MB.
[icon] [icon] application: not a procedure;
expected a procedure that can be applied to arguments
given: (0 0 0 1 0 1 1)
arguments...: [none]
> |

R5RS 7:2 [icon]
```

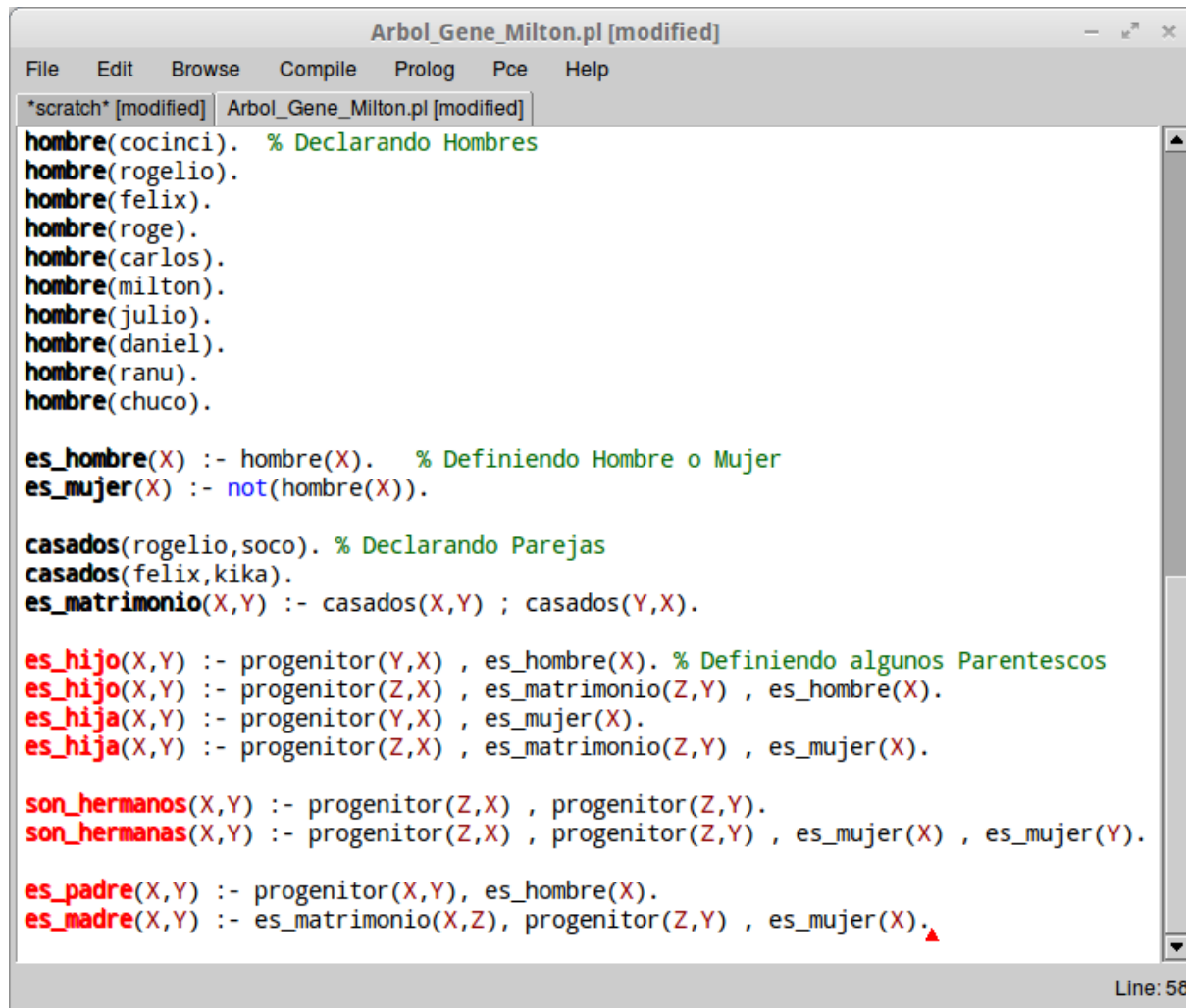


Paradigma lógico

- Se basa en la definición de reglas lógicas para luego, a través de un motor de inferencias lógicas, responder preguntas planteadas al sistema y así resolver los problemas.
- Ej.: prolog.



El Lenguaje ProLog



```
Arbol_Gene_Milton.pl [modified]
File Edit Browse Compile Prolog Pce Help
*scratch* [modified] Arbol_Gene_Milton.pl [modified]

hombre(cocinci). % Declarando Hombres
hombre(rogelio).
hombre(felix).
hombre(roge).
hombre(carlos).
hombre(milton).
hombre(julio).
hombre(daniel).
hombre(ranu).
hombre(chuco).

es_hombre(X) :- hombre(X). % Definiendo Hombre o Mujer
es_mujer(X) :- not(hombre(X)).

casados(rogelio,soco). % Declarando Parejas
casados(felix,kika).
es_matrimonio(X,Y) :- casados(X,Y) ; casados(Y,X).

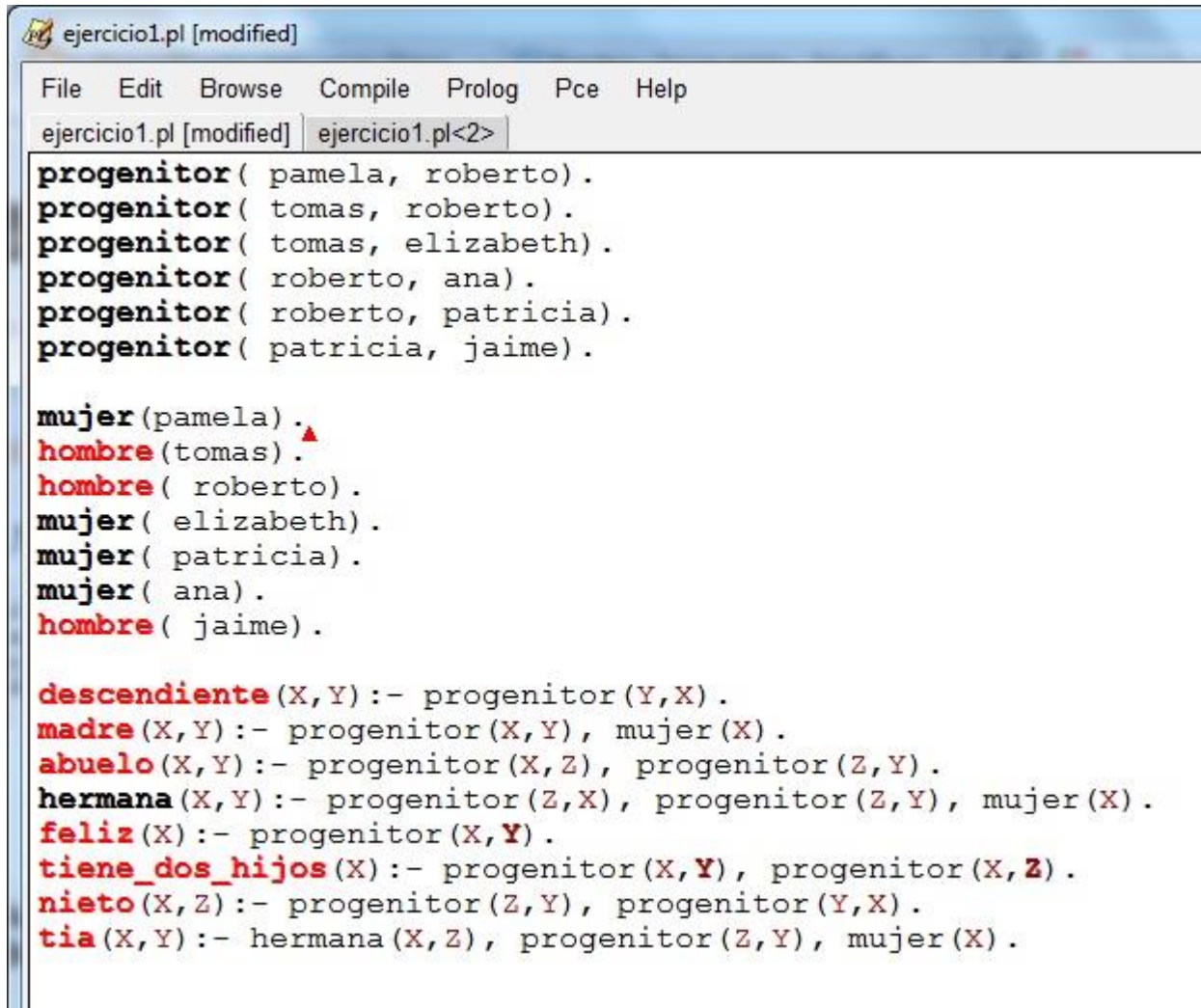
es_hijo(X,Y) :- progenitor(Y,X) , es_hombre(X). % Definiendo algunos Parentescos
es_hijo(X,Y) :- progenitor(Z,X) , es_matrimonio(Z,Y) , es_hombre(X).
es_hija(X,Y) :- progenitor(Y,X) , es_mujer(X).
es_hija(X,Y) :- progenitor(Z,X) , es_matrimonio(Z,Y) , es_mujer(X).

son_hermanos(X,Y) :- progenitor(Z,X) , progenitor(Z,Y).
son_hermanas(X,Y) :- progenitor(Z,X) , progenitor(Z,Y) , es_mujer(X) , es_mujer(Y).

es_padre(X,Y) :- progenitor(X,Y), es_hombre(X).
es_madre(X,Y) :- es_matrimonio(X,Z), progenitor(Z,Y) , es_mujer(X).
```

Line: 58

Ejemplo ProLog



```
ejercicio1.pl [modified] ejercicio1.pl<2>

progenitor( pamela, roberto).
progenitor( tomas, roberto).
progenitor( tomas, elizabeth).
progenitor( roberto, ana).
progenitor( roberto, patricia).
progenitor( patricia, jaime).

mujer( pamela).
hombre( tomas).
hombre( roberto).
mujer( elizabeth).
mujer( patricia).
mujer( ana).
hombre( jaime).

descendiente(X,Y):- progenitor(Y,X).
madre(X,Y):- progenitor(X,Y), mujer(X).
abuelo(X,Y):- progenitor(X,Z), progenitor(Z,Y).
hermana(X,Y):- progenitor(Z,X), progenitor(Z,Y), mujer(X).
feliz(X):- progenitor(X,Y).
tiene_dos_hijos(X):- progenitor(X,Y), progenitor(X,Z).
nieto(X,Z):- progenitor(Z,Y), progenitor(Y,X).
tia(X,Y):- hermana(X,Z), progenitor(Z,Y), mujer(X).
```


Paradigma Orientado a Objetos

- Basado en la idea de encapsular estado y operaciones en objetos.
- En general, la programación se resuelve comunicando dichos objetos a través de mensajes (programación orientada a mensajes).
- Se puede incluir -aunque no formalmente- dentro de este paradigma, el paradigma basado en objetos, que además posee herencia y subtipos entre objetos. Ej.: Simula, Smalltalk, C++, Java, Visual Basic .NET, etc.
- Su principal ventaja es la reutilización de códigos y su facilidad para pensar soluciones a determinados problemas.

Paradigma Orientado a Objetos

- ¿Por qué Orientación a Objetos (OO)?
 - Se parece más al mundo real
 - Permite representar modelos complejos
 - Muy apropiada para aplicaciones de negocios
 - Las empresas ahora sí aceptan la OO
 - Las nuevas plataformas de desarrollo la han adoptado (Java / .NET)

¿Qué es un Objeto?

- Informalmente, un objeto representa una entidad del mundo real
- Entidades Físicas
 - (Ej.: Vehículo, Casa, Producto)
- Entidades Conceptuales
 - (Ej.: Proceso Químico, Transacción Bancaria)
- Entidades de Software
 - (Ej.: Lista Enlazada, Interfaz Gráfica)

¿Qué es un Objeto?

- Definición Formal (Rumbaugh):
 - “Un objeto es un concepto, abstracción o cosa con un significado y límites claros en el problema en cuestión”
- Un objeto posee (Booch):
 - Estado
 - Comportamiento
 - Identidad

Un objeto posee Estado

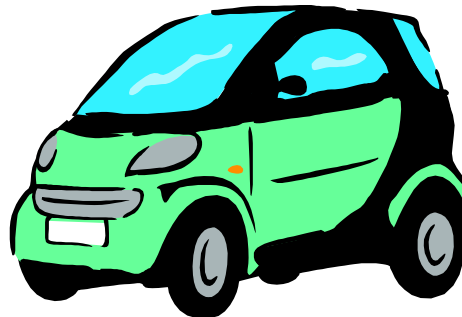
- *Lo que el objeto sabe*
- El estado de un objeto es una de las posibles condiciones en que el objeto puede existir
- El estado normalmente cambia en el transcurso del tiempo
- El estado de un objeto es implementado por un conjunto de propiedades (atributos), además de las conexiones que puede tener con otros objetos

Un objeto posee Comportamiento

- ***Lo que el objeto puede hacer***
- El comportamiento de un objeto determina cómo éste actúa y reacciona frente a las peticiones de otros objetos
- Es modelado por un conjunto de mensajes a los que el objeto puede responder (operaciones que puede realizar)
- Se implementa mediante métodos

Un objeto posee Identidad

- Cada objeto tiene una identidad única, incluso si su estado es idéntico al de otro objeto

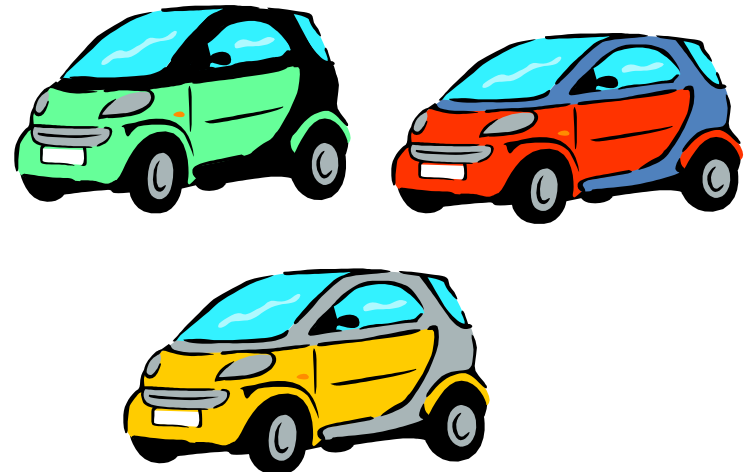
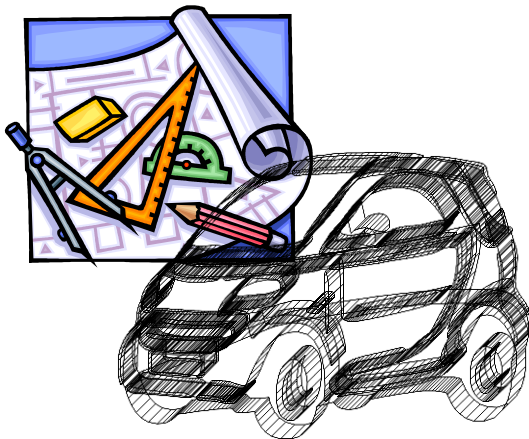


¿Qué es una Clase?

- Una clase es una descripción de un grupo de objetos con:
 - Propiedades en común (atributos)
 - Comportamiento similar (operaciones)
 - La misma forma de relacionarse con otros objetos (relaciones)
 - Una semántica en común (significan lo mismo)
- Una clase es una abstracción que:
 - Enfatiza las características relevantes
 - Suprime otras características (simplificación)
- Un objeto es una instancia de una clase

Objetos y Clases

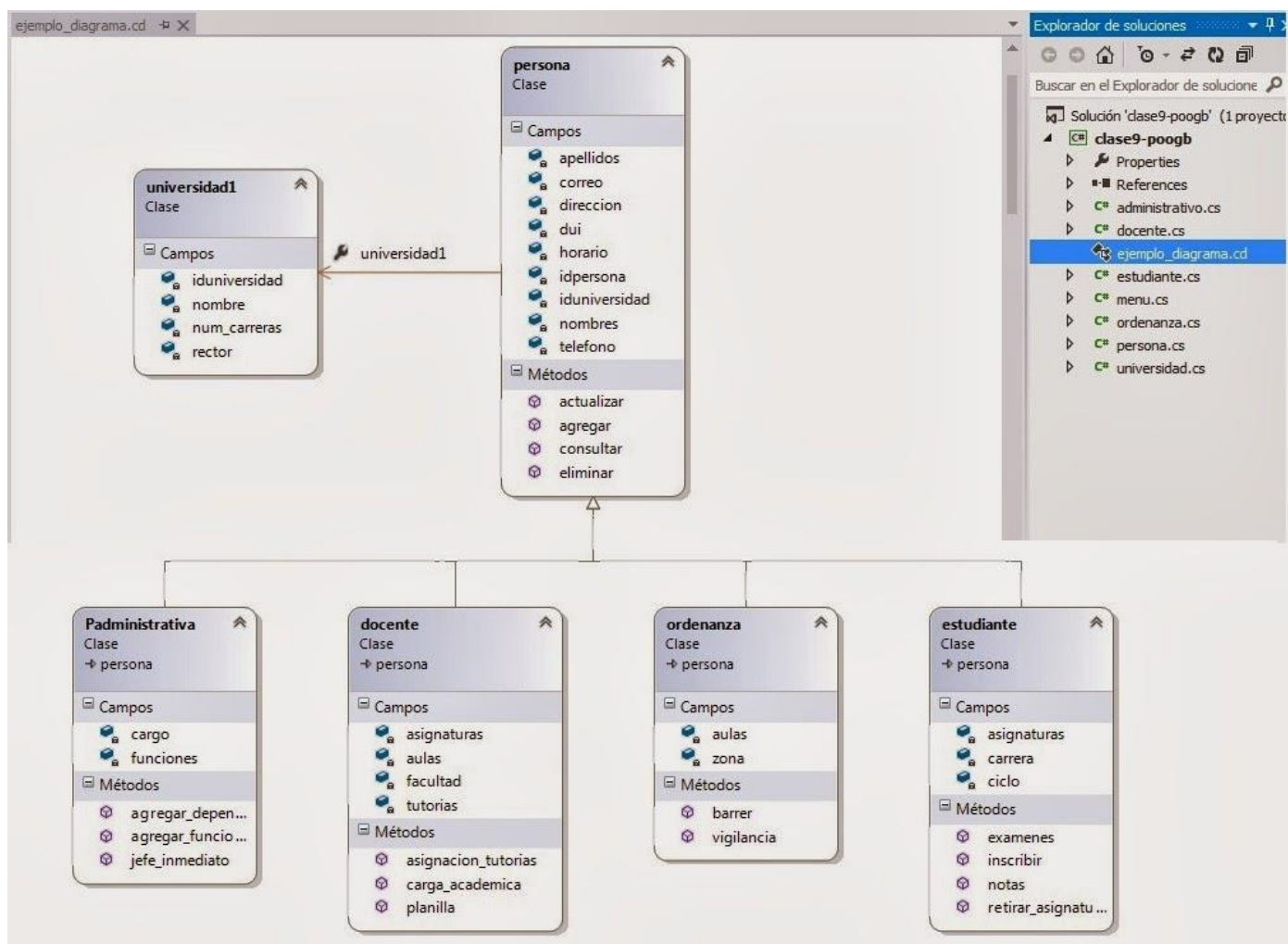
- Una clase es una definición abstracta de un objeto
 - Define la estructura y el comportamiento compartidos por los objetos
 - Sirve como modelo para la creación de objetos
- Los objetos pueden ser agrupados en clases



Ejemplo de una Clase

- Clase: Curso
- Estado (Atributos)
 - Nombre
 - Ubicación
 - Días Ofrecidos
 - Horario de Inicio
 - Horario de Término
- Comportamiento (Métodos)
 - Agregar un Alumno
 - Borrar un Alumno
 - Entregar un Listado del Curso
 - Determinar si está Completo

Ejemplo Diagrama de Clases



Creando una Clase en C#

```
/* By SergioGirado */
namespace PartialClassSample.NoPartial
{
    class Persona
    {
        public string Identificacion { get; set; }
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public int Edad { get; set; }

        public override string ToString()
        {
            /*
             (123123) Sergio Girado : 18
            */
            return string.Format(
                "({0}) {1} {2} : {3}",
                Identificacion, Nombre, Apellido, Edad
            );
        }
    }
}
```