

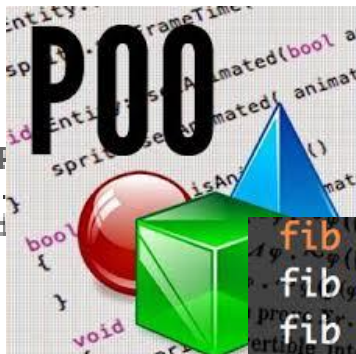
# Paradigmas de Programación

## CYPLP

# PARADIGMAS

Un paradigma de programación es un **estilo de desarrollo de programas**, un modelo para resolver problemas computacionales. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez, a partir del tipo de órdenes que permiten implementar, tiene una **relación directa con su sintaxis**.

```
1 procedure DandC(pbm, sol)
2 local var aux;
3 begin
4   if easy(pbm) then
5     solve(pbm)
6   else
7     begin
8       divide(pbm, subp)
9       parallel i in 1
10        DandC(subpbu[i])
11      combine(subsol,
12    end
13 end;
```



```
?- p_exam(noest,notar).
false.

?- p_exam(est,tar).
true.

?- p_exam(noest,tar).
true.
```

```
Integer -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```



# PRINCIPALES PARADIGMAS

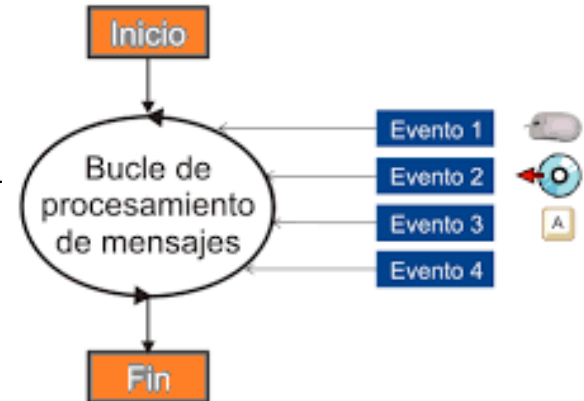
- **Imperativo:** sentencias + secuencias de comandos
- **Declarativo.** Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.
  - **Lógico.** Aserciones lógicas: hechos + reglas, es declarativo
- **Funcional.** Los programas se componen de funciones
- **Orientado a Objetos :** Métodos + mensajes.



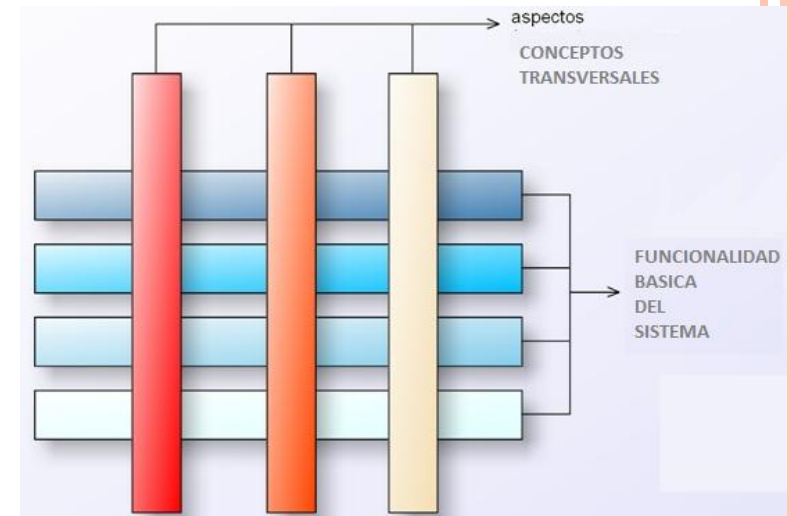
# PRINCIPALES PARADIGMAS

Otra forma de clasificación mas reciente:

- **Dirigido por eventos.** El flujo del programa está determinado por sucesos externos (por ejemplo, una acción del usuario).



- **Orientado a aspectos.**  
Apunta a dividir el programa en módulos independientes, cada uno con un comportamiento y responsabilidad bien definido.



# PARADIGMA APLICATIVO O FUNCIONAL

Basado en el uso de funciones. Muy popular en la resolución de problemas de inteligencia artificial, matemática, lógica, procesamiento paralelo

Ventajas:

- Vista uniforme de programa y función
- Tratamiento de funciones como datos
- Liberación de efectos colaterales
- Manejo automático de memoria

Desventaja:

- Ineficiencia de ejecución



# PARADIGMA FUNCIONAL

## Características de los lenguajes funcionales

- Define un conjunto de datos
- Provee un conjunto de funciones primitivas
- Provee un conjunto de formas funcionales
- Requiere de un operador de aplicación
- Semántica basada en valores
- Transparencia referencial
- Regla de mapeo basada en combinación o composición
- Las funciones de primer orden



# PROGRAMACIÓN FUNCIONAL

## Funciones

- El VALOR más importante en la programación funcional es el de una FUNCIÓN
- Matemáticamente una función es un correspondencia :  
 $f: A \rightarrow B$
- A cada elemento de A le corresponde un único elemento en B
- $f(x)$  denota el resultado de la aplicación de  $f$  a  $x$
- Las funciones son tratadas como valores, pueden ser pasadas como parámetros, retornar resultados, etc.



# PROGRAMACIÓN FUNCIONAL

## Definiendo Funciones:

- Se debe distinguir entre el VALOR y la DEFINICIÓN de una función.
- Existen muchas maneras de DEFINIR una misma función, pero siempre dará el mismo valor, ejemplo:

DOBLE X = X + X

DOBLE' X = 2 \* X

Denotan la misma función pero son dos formas distintas de definirlas

## Tipo de una función

- Puede estar definida explícitamente dentro del SCRIPT, por ejemplo:

cuadrado::num  $\rightarrow$  num

cuadrado x = x + x

- O puede **deducirse/inferirse** el tipo de una función





# PROGRAMACIÓN FUNCIONAL

## Expresiones y valores

- La expresión es la noción central de la programación Funcional
- Característica más importante:
  - “Una expresión es su VALOR”
- El valor de una expresión depende ÚNICAMENTE de los valores de las sub expresiones que la componen.
- Las expresiones también pueden contener VARIABLES, (valores desconocidos)



# PROGRAMACIÓN FUNCIONAL

## Expresiones y valores

- La noción de Variable es la de “variable matemática”, no la de celda de memoria.
- Las expresiones cumplen con la propiedad de  
  
“TRANSPARENCIA REFERENCIAL”: Dos expresiones sintácticamente iguales darán el mismo valor.
- **No** existen EFECTOS LATERALES”



# PROGRAMACIÓN FUNCIONAL

- Ejemplos de expresiones para evaluar

Expresión	Valor	
47	47	
( <b>*</b> 4 7)	28	Se está utilizando una Función primitiva
( <b>+</b> 49 5)	54	

- Definiendo funciones....

cuadrado  $x = x * x$

min  $x y = x$ , if  $x < y$   
 $y$ , if  $x > y$

cube  $(x) = x * x * x$



# PROGRAMACIÓN FUNCIONAL

Un script es una lista de definiciones y

- Pueden someterse a evaluación. Ejemplos:

?cuadrado (3 + 4 )

49

?min 3 4

3

- Pueden combinarse, Ejemplo:

?min(cuadrado ( 1 + 1 ) 3)

3

- Pueden modificarse, ejemplo: Al script anterior le agrego nuevas definiciones:

lado = 12

area = cuadrado lado



# PROGRAMACIÓN FUNCIONAL

- Algunas expresiones pueden **NO** llegar a reducirse del todo, ejemplo:  $1/0$
- A esas expresiones se las denominan **CANÓNICAS**, pero se les asigna **un VALOR INDEFINIDO** y corresponde al símbolo  $\text{bottom}(\wedge)$
- Por lo tanto toda **EXPRESIÓN** siempre denota un **VALOR**



# PROGRAMACIÓN FUNCIONAL

Evaluación de las expresiones:

- La forma de evaluar es a través de un mecanismo de REDUCCIÓN o SIMPLIFICACIÓN

- Ejemplo:

cuadrado  $(3 + 4)$

$\Rightarrow$  cuadrado 7 (+)

$\Rightarrow 7 * 7$  (cuadrado)

$\Rightarrow 49$  (\*)

Otra forma sería:

cuadrado  $(3 + 4)$

$\Rightarrow (3 + 4) * (3 + 4)$  (cuadrado)

$\Rightarrow 7 * (3 + 4)$  (+)

$\Rightarrow 7 * 7$  (+)

$\Rightarrow 49$  (\*)

**“No importa la forma de evaluarla, siempre el resultado final será el mismo”**



# PROGRAMACIÓN FUNCIONAL

Existen dos formas de reducción:

- Orden aplicativo

Aunque no lo necesite  
**SIEMPRE** evalúa los  
argumentos

- Orden normal  
(**lazy evaluation**)

No calcula más de lo necesario  
La expresión NO es evaluada hasta que  
su valor se necesite  
Una expresión compartida NO es  
evaluada más de una vez



# PROGRAMACIÓN FUNCIONAL

TIPOS {  
    Básicos  
    Derivados

**Básicos:** Son los primitivos, ejemplo:

**NUM (INT y FLOAT) (Números)**

**BOOL(Valores de verdad)**

**CHAR(Caracteres)**

**Derivados:** Se construyen de otros tipos, ejemplo:

**(num,char) Tipo de pares de valores**

**(num → char) Tipo de una función**

**TODA FUNCIÓN TIENE ASOCIADO UN TIPO**





# PROGRAMACIÓN FUNCIONAL

- Expresiones de tipo polimórficas:

En algunas funciones no es tan fácil deducir su tipo.

Ejemplo:

**id x = x**

Esta función es la **función Identidad**

Su tipo puede ser de  $\text{char} \rightarrow \text{char}$ , de  $\text{num} \rightarrow \text{num}$ , etc.

Por lo tanto su tipo será de  $\beta \rightarrow \beta$

Se utilizan **letras griegas** para **tipos polimórficos**

Otro ejemplo: **letra x = "A"**

Su tipo será  $\beta \rightarrow \text{char}$



# PROGRAMACIÓN FUNCIONAL

## ○ Currificación:

- Mecanismo que reemplaza argumentos estructurados por argumentos más simples.
- Ejemplo: sean dos definiciones de la Función “Suma”

1.  $\text{Suma}(x,y) = x + y$

2.  $\text{Suma}' x y = x + y \longrightarrow \text{Suma}' x \ y = \text{Suma}'x (y) = x + y$

Existen entre estas dos definiciones una diferencia sutil: **“Diferencia de tipos de función”**

El tipo de Suma es :  $(\text{num}, \text{num}) \rightarrow \text{num}$

El tipo de Suma' es :  $\text{num} \rightarrow (\text{num} \rightarrow \text{num})$

Por cada valor de x devuelve una función

## Aplicando la función:

$$\text{Suma}(1,2) \rightarrow 3$$

$$\text{Suma}' 1 \ 2 \rightarrow \text{Suma}'1 \text{ aplicado al valor } 2 \\ 3$$

Para todo los valores devuelve el siguiente

# PROGRAMACIÓN FUNCIONAL

## Cálculo Lambda

- El un modelo de computación para definir funciones
- Se utiliza para entender los elementos de la programación funcional y la semántica subyacente, independientemente de los detalles sintácticos de un lenguaje de programación en particular.
- Es un modelo de programación funcional que se independiza de la sintaxis del lenguaje de programación



# PROGRAMACIÓN FUNCIONAL

Las expresiones del Lambda cálculo pueden ser de 3 clases:

- Un simple identificador o una constante. Ej:  $x$ ,  $3$
- Una definición de una función. Ej:  $\lambda x.x+1$
- Una aplicación de una función. La forma es  $(e1\ e2)$ , dónde se lee  $e1$  se aplica a  $e2$ .


**Ej: en la funcion cube  $(x) = x * x * x$**

$\lambda x.x * x * x$

$\lambda (x.x * x * x)$

Evaluamos la  
función con 2  
y resulta en 8

# PROGRAMACIÓN LÓGICA

- La programación lógica es un tipo de paradigmas de programación dentro del paradigma de programación declarativa
  - Es un paradigma en el cual los programas son una serie de aserciones lógicas.
  - El conocimiento se representa a través de **reglas** y **hechos**
  - Los objetos son representados por **términos**, los cuales contienen constantes y variables
  - PROLOG es el lenguaje lógico más utilizado.
- 

# ELEMENTOS DE LA PROGRAMACIÓN LÓGICA

- La sintáxis básica es el “**término**”

- **Variables:**

- Se refieren a elementos indeterminados que pueden sustituirse por cualquier otro.

*“humano(X)”*, la X puede ser sustituida por constantes como: juan, pepe, etc.

- Los nombres de las variables comienzan con mayúsculas y pueden incluir números.

- **Constantes:**

- A diferencia de las variables son elementos determinados.

*“humano(juan)”*

- Las constantes son string de letras en minúsculas (representan objetos atómicos) o string de dígitos (representan números).



# ELEMENTOS DE LA PROGRAMACIÓN LÓGICA

Término compuesto:

- Consisten en un "functor" seguido de un número fijo de argumentos encerrados entre paréntesis, los cuales son a su vez términos.
- Se denomina "aridad" al número de argumentos.
- Se denomina "estructura" (ground term) a un término compuesto cuyos argumentos no son variables.

Ejemplos:

**padre**

**Longitud**

**tamaño(4,5)**

→

→

→

**constante**

**variable**

**estructura**



# ELEMENTOS DE LA PROGRAMACIÓN LÓGICA

Listas:

- La constante `[]` representa una lista vacía
- El functor “.” construye una lista de un elemento y una lista. Ejemplo: `.(alpha,[])`, representa una lista que contiene un único elemento que es `alpha`.
- Otra manera de representar la lista es usando `[]` en lugar de `.( )`. Ejemplo anterior la lista quedaría: `[alpha,[]]`
- Y también se representa utilizando el símbolo `|`  
`[alpha | []]`

La notación general para denotar lista es : `[X | Y]`

X es el elemento cabeza de la lista e

Y es una lista, que representa la cola de la lista que se está modelando



# CLÁUSULAS DE HORN

- Un programa escrito en un lenguaje lógico es una secuencia de “cláusulas”.
- Las cláusulas pueden ser: un “Hecho” o una “Regla”.

Hecho:

- Expresan relaciones entre objetos
- Expresan verdades
- Son expresiones del tipo  $p(t_1, t_2, \dots, t_n)$

Ejemplos:

- $\text{tiene}(\text{coche}, \text{ruedas}) \rightarrow$  representa el hecho que un coche tiene ruedas
- $\text{longuitud}([], 0) \rightarrow$  representa el hecho que una lista vacía tiene longitud cero
- $\text{virus}(\text{ithaqua}) \rightarrow$  representa el hecho que ithaqua es un virus.

# CLÁUSULAS DE HORN

Regla:

Sintáxis de Prolog

- Cláusula de Horn
- Tiene la forma: conclusión :- condición.

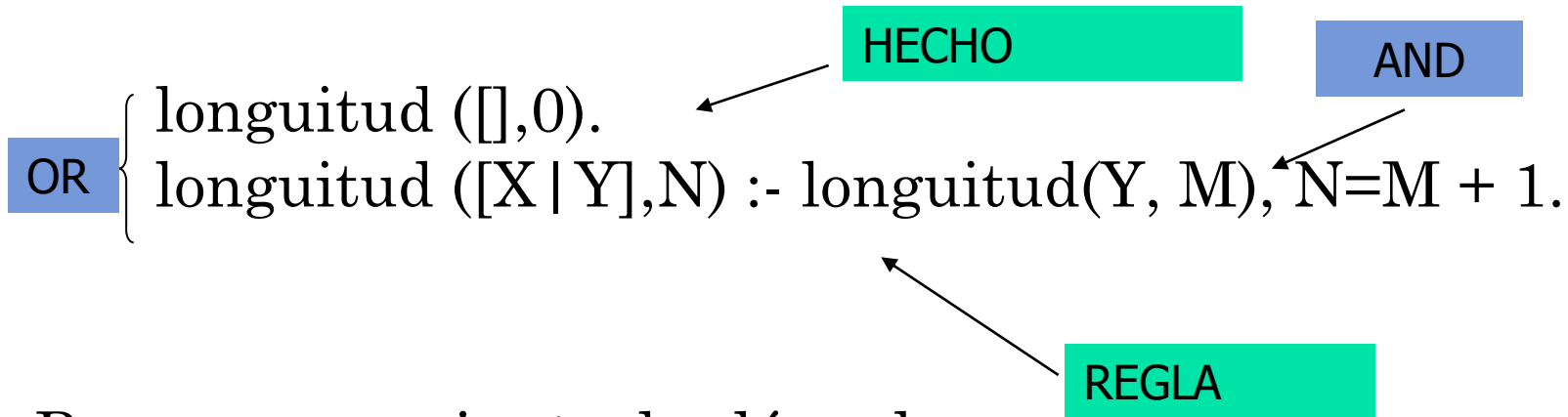
Dónde:

- :- indica “Si”
- conclusión es un simple predicado y
- condición es una conjunción de predicados, separados por comas. Representan un AND lógico
- En un lenguaje procedural una regla la podríamos representar como: if condición else conclusión.



# PROGRAMAS Y QUERIES

## Ejemplo de programa:



Programa: conjunto de cláusulas

?-longuitud([rojo | [verde | [azul | [] ] ] ],X).

**QUERY** (green box) points to the query.

Query: Representa lo que deseamos que sea contestado



# PROGRAMAS Y QUERIES

## Programa:

longitud ([],0).

longitud ([X | Y],N) :- longitud(Y, M), N=M + 1.

?-longitud([rojo | [verde | [azul | [] ] ] ],X).

longitud([verde | [azul | [] ] ],M) y  $X=M+1$

longitud([azul | [] ] ,Z) y  $M=Z+1$

longitud([],T)  $Z=T+1$      $T=0 \Rightarrow Z=1$

$M=2$

$X=3$



# EJECUCIÓN DE PROGRAMAS

- Un programa es un conjunto de reglas y hechos que proveen una especificación declarativa de que es lo que se conoce y la pregunta es el objetivo que queremos alcanzar.
- La ejecución de dicho programa será el intento de obtener una respuesta.
- Desde un punto de vista lógico la respuesta a esa pregunta es “YES”, si la pregunta puede ser derivada aplicando “deducciones” del conjunto de reglas y hechos dados.



# EJECUCIÓN DE PROGRAMAS: EJEMPLO

Programa que describe una relación binaria (rel) y su cierre (clos):

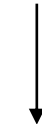
**rel(a,b).**  
**rel(a,c).**  
**rel(b,f).**  
**rel(f,g).**  
**clos(X,Y) :- rel(X, Y).**  
**clos(X,Y) :- rel(X, Z), clos(Z, Y).**

**?-clos(a,f)**

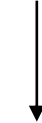
PROGRAMA

QUERY

clos (a,f)



rel(a,f)



falla

clos (a,f)



**rel(a,Z1),** clos(Z1,f)



**rel(a,b),** clos(b,f)

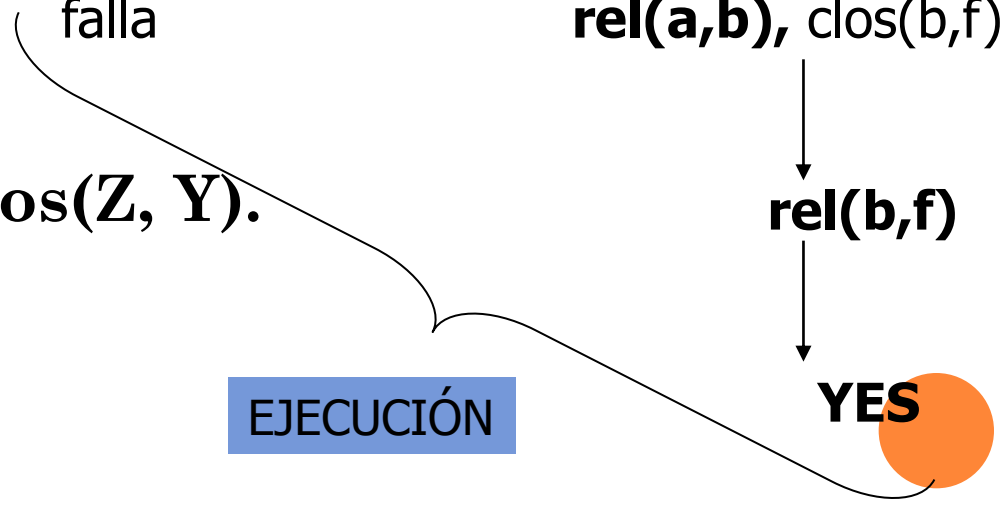


**rel(b,f)**



**YES**

EJECUCIÓN



# PROGRAMACIÓN ORIENTADA A OBJETOS

“Un programa escrito con una lenguaje OO es un conjunto de OBJETOS que **INTERACTÚAN** mandándose **MENSAJES**”

Los elementos que intervienen en la programación OO son:

- Objetos
- Mensajes
- Métodos
- Clases

## Objetos:

- Son entidades que poseen estado interno y comportamiento
- Es el equivalente a un dato abstracto



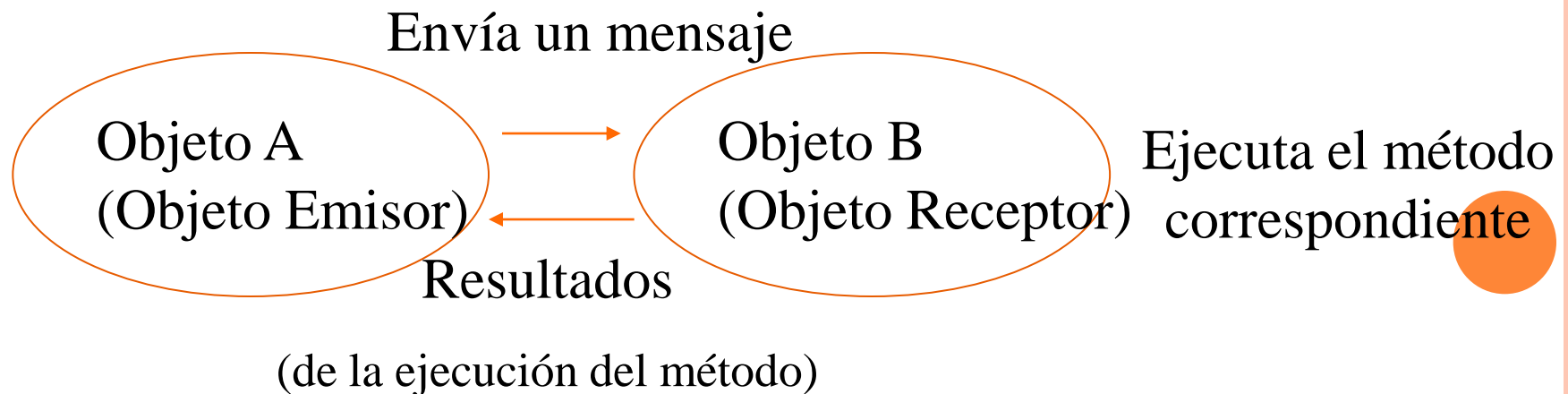
# PROGRAMACIÓN ORIENTADA A OBJETOS

## Mensajes:

- Es una petición de un objeto a otro para que este se comporte de una determinada manera, ejecutando uno de sus métodos
- TODO el procesamiento en este modelo es activado por mensajes entre objetos.

## Métodos:

- Es un programa que está asociado a un objeto determinado y cuya ejecución solo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes






# PROGRAMACIÓN ORIENTADA A OBJETOS

## Clases:

- Es un tipo definido por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo
- Cada objeto pertenece a una clase y recibe de ella su funcionalidad
- Primer nivel de abstracción de datos: definimos estructura, comportamiento y tenemos ocultamiento.
- La información contenida en el objeto solo puede ser accedida por la ejecución de los métodos correspondientes

## Instancia de clase:

- Cada vez que se construye un objeto se está creando una INSTANCIA de esa clase
  - Una instancia es un objeto individualizado por los valores que tomen sus atributos
- 

# PROGRAMACIÓN ORIENTADA A OBJETOS

## Otro aspecto de las abstracciones de datos

### GENERALIZACIÓN/ESPECIFICACIÓN



### HERENCIA

El segundo nivel de abstracción consiste en agrupar las clases en jerarquías de clases (definiendo SUB y SUPER clases), de forma tal que una clase A herede todas las propiedades de su superclase B (suponiendo que tiene una)

Ejemplo: Se tiene definido la siguiente clase

PERSONA

- Nombre
- Edad
- Sexo
- Documento
- Dirección
- Teléfono

**ver-nombre**  
**ver-edad**  
**ver-teléfono**  
**ver-documento**  
**ver-sexo**  
**cambiar-dirección**  
**sacar-documento, etc.**

EMPLEADO  
•Curriculum  
•cuil

# PROGRAMACIÓN ORIENTADA A OBJETOS

## Otros conceptos adicionales

Polimorfismo:

- Es la capacidad que tienen los objetos de distintas clases de responder a mensajes con el mismo nombre

Ejemplo:

$3 + 5$   
“Buenos” + “días”

Se aplica suma entre números  
Se concatenan strings

Binding dinámico:

Es la vinculación en el proceso de ejecución de los objetos con los mensajes



# PROGRAMACIÓN ORIENTADA A OBJETOS

## **C++ (Lenguaje híbrido)** **Algunas características**

- Lenguaje extendido del lenguaje C
- Incorporó características de POO
- Lenguaje compilativo el ambiente de programación es de los lenguajes tradicionales.

Los objetos en C++:

- Se agrupan en tipos denominados clases
  - Contienen datos internos que definen su estado interno
  - Soportan ocultamiento de datos
  - Los métodos son los que definen su comportamiento
  - Pueden heredar propiedades de otros objetos
  - Pueden comunicarse con otros objetos enviándose mensajes
- 