Paradigmas en Programación
Resumen

Pianelli Felipe

Buiatti Pedro

Programación Orientada a Objetos

Clase 1 - El proceso de abstracción

Las aplicaciones de software modelan el mundo real. Los humanos construimos modelos mentales para entender el mundo. Un modelo mental es una visión simplificada de cómo las cosas funcionan y cómo podemos interactuar con ellas. El mundo real es complejo, la abstracción es uno de los mecanismos para combatir la complejidad. Los paradigmas de programación proveen un conjunto de herramientas conceptuales que nos permiten analizar, representar y abordar los problemas.

Durante el **proceso de abstracción tomamos cosas del mundo real y lo modelamos en forma de programación.**

Cada **paradigma de programación** determina la manera de pasar del espacio de los problemas al de la implementación de una solución.

Diferentes paradigmas de programación:

- **Programación imperativa:** La programación imperativa es un enfoque de programación que se centra en describir los pasos específicos y detallados que debe seguir una computadora para ejecutar una tarea. Un programa es, por lo tanto, un conjunto de instrucciones y ordenes que determinan lo que debe hacer la computadora. (Pascal, C, ADA).

- **Programación Orientada a Objetos:** Smalltalk, Java, Python, C++

- **Programación Funcional:** Consiste en escribir un programa basado en funciones matemáticas. No hay un orden específico, no hay un orden secuencial en el que se ejecutan las funciones del programa, si no que se ejecuta la función que es acorde al deseo del usuario. (Lisp, Haskell).

- **Programación Lógica:** Tiene como base la lógica proposicional, es una sucesión de predicados lógicos. Los programas son demostradores de consultas. Los programas tienen un orden de escritura, y cada vez que se ejecuta un programa se recorren todos los predicados hasta encontrar el predicado que coincida con la consulta hecha. Los programas se denominan base de conocimiento. Esta es usada en la inteligencia artificial. (Prolog)

Programación orientada a objetos: El objetivo de la Programación Orientada a Objetos es manejar la complejidad de los problemas del mundo real, abstrayendo su conocimiento y encapsulándolo en objetos. Esta **describe un sistema en término de los objetos involucrados, dichos objetos interactúan entre sí enviándose mensajes para llevar a cabo una tarea.**

Identificar los objetos es un arte y no una ciencia, el resultado dependerá del contexto y punto de vista de quien los modela.

Características:

- Simulación

- Encapsulamiento

- Abstracción

- Clasificación
- Binding dinámico
- Herencia
- Extensión
- Polimorfismo

Programamos por simulación, es decir, se trata de simular a las entidades del mundo real en un programa.

1. Objetos. Estado y comportamiento. Composición

Objeto: Representa una entidad de la vida real. Cada objeto abstrae un dato del programa y lo que puede hacerse sobre él. Pueden ser objetos concretos, que tienen representación física en el mundo real, o **objetos abstractos**, que no tienen representación física en el mundo real. Tienen dos características: estado y comportamiento.

Estado: son los **atributos** que describen al objeto unívocamente, es decir, las propiedades relevantes o que tienen valor del objeto. Estos atributos se almacenan en **variables de instancia.**

Comportamiento: son el conjunto de acciones, operaciones, **funciones o métodos** que lleva a cabo el objeto. Permiten **consultar o modificar el estado** del objeto.

A la representación de un objeto de la vida real en un programa se lo denomina **objeto de software.**

Composición de objetos: Un objeto puede componerse de dos o más objetos, conformando así un objeto compuesto.

2. Clases e instancias

Clase: Una clase es una plantilla o molde para crear objetos. Contiene los atributos y comportamientos que luego tendrán los objetos. Si dos objetos comparten los mismos atributos y comportamientos, estos pertenecen (se agrupan) en la misma clase. Esta encapsula y oculta los atributos y comportamientos de un objeto.

- Vista externa o pública del objeto: para el usuario, dada a través de un protocolo que dice para qué sirve un objeto y qué se puede hacer con él.
- Vista interna o privada: Para el programador, que muestra la implementación de las operaciones y su estructura interna.

Clase abstracta: No puedo crear objetos a partir de dicha clase. No puedo instanciarla, tampoco puedo implementar sus métodos. Si o si necesito crear subclases de dicha clase para poder instanciar objetos e implementar sus operaciones.

Instancia: Una instancia es un objeto en particular de una clase.

La clase es quien **encapsula y oculta la estructura interna y la implementación de las operaciones**. La clase provee el formato de un objeto y especifica su comportamiento compartido.

Toda clase es subclase de otra clase más general denominada superclase. La subclase es una especialización de la superclase porque detalla algún atributo que la distingue de la superclase, mientras que la superclase es una generalización de la subclase y comparte todos sus atributos con la subclase. La subclase hereda los atributos y el comportamiento completo de la superclase (no siempre). Todo lo heredado no se vuelve a escribir, **se hereda por extensión**, es decir, se extiende el código existente solo agregando lo nuevo. Si se desea redefinir un método, este mantiene el mismo nombre pero se cambia su código (**métodos polimórficos).**

3. Mensajes y métodos. Invocación de métodos.

Para que un objeto lleve a cabo una operación especifica, se le debe enviar un mensaje, y el objeto responde ejecutando el método asociado. El objeto al recibir un mensaje busca en su protocolo de instancia el método y ejecuta su operación determinada.

Cuando el objeto recibe un mensaje, el hecho de responder con la ejecución de un método puede involucrar la participación de otros objetos. **La resolución del problema puede requerir la ayuda de otros individuos.**

Cada objeto cumple un rol, ejecuta una acción, que es usada por otros miembros de la comunidad. El **emisor** envía un mensaje al receptor, junto con los argumentos necesarios para llevar a cabo el requerimiento. El **receptor** es el objeto a quien se le envía el mensaje.

El mensaje es la invocación al método. El método es la implementación del mensaje.

El formato de un mensaje es: **OR mensaje,** OR es el objeto receptor, al cual se le solicita que ejecute el método correspondiente.

4. Encapsulamiento y ocultamiento de información

Encapsulamiento: una clase encapsula la **representación privada** y el **comportamiento de un objeto.**

Un objeto no conoce el funcionamiento interno de los demás objetos y no lo necesita para poder interactuar con ellos.

Ocultamiento de la información: Sólo puede modificarse la estructura interna de un objeto, desde fuera de la clase, a través de la interfaz pública del objeto, es decir sólo a través de los mensajes que entiende el objeto. Para consultar o modificar el valor almacenado en variables de instancia de un objeto solo puede hacerse mediante los métodos definidos en la clase para tal fin.

5. Herencia. Polimorfismo. Binding dinámico.

Herencia: Es la propiedad de crear nuevas clases a partir de otras ya existentes, ampliando su estructura y comportamiento.

Las clases están organizadas en una jerarquía de clases, donde las subclases heredan estado y comportamiento de las superclases.

Las subclases agregan nuevos atributos, comportamiento, y además pueden cambiar el comportamiento de los métodos heredados.

Aquí se pone énfasis en la programación por extensión o la reusabilidad del código, para evitar reescribir código previamente desarrollado.

En una jerarquía debe respetarse la relación es un.

La herencia puede ser:

- De estado: se heredan solo los atributos de la superclase.
- De comportamiento: se heredan solo los métodos de la superclase.
- Total, o parcial: se heredan todos los atributos y métodos (total), o solo parte de ellos.
- Simple o múltiple: se hereda de una sola superclase, o se hereda de varias superclases.

Cuando se programa usando herencia, se **programa por extensión**:

- Se extiende el código existente, se rehúsa código. No se vuelve a escribir lo heredado. Sólo se agrega código correspondiente a los nuevos atributos y a las operaciones nuevas.
- Cuando se le envía un mensaje a un objeto, el S.O. lo busca en la clase de dicho objeto. Si no lo encuentra, porque es heredado, sube por el árbol jerárquico de clases aplicando una **BÚSQUEDA LOOK-UP**, hasta encontrarlo y lo ejecuta.

Polimorfismo

Es la capacidad que tienen objetos diferentes de entender y responder a un mismo mensaje de manera diferente.

Dos métodos tienen el mismo nombre, pero pertenecen a diferentes clases y responden de distinta manera (métodos polimórficos). No es necesariamente por herencia, pueden haber métodos polimórficos pertenecientes a clases completamente diferentes que no pertenecen a la misma jerarquía de clases.

Binding dinámico

Esto significa que la ligadura entre el objeto receptor y el método que se va a ejecutar se realizará en tiempo de ejecución. En tiempo de ejecución se va a conocer qué método se va a ejecutar ya que está determinado por el objeto receptor. Por eso pueden haber métodos denominados de la misma forma para distintos objetos, que están implementados de forma diferente, porque esa asociación entre método y objeto se hace en tiempo de ejecución.

Clase 2 – Smalltalk. Mensajes. Estructuras de Control. Jerarquía de clases.

Características de Smalltalk

- Es considerado el primero de los lenguajes orientados a objetos, aunque en realidad el primero en implementar la programación orientada a objetos fue Simula.
- Ha tenido gran influencia sobre otros lenguajes como Java o Ruby, y de su entorno han surgido muchas de las prácticas y herramientas de desarrollo promulgadas actualmente por las metodologías ágiles (refactorización, desarrollo incremental, desarrollo dirigido por tests, etc.).
- Es un lenguaje de programación reflexivo y con tipado dinámico.

Smalltalk es un lenguaje orientado a objetos puro, todas las entidades que maneja son objetos.

- El lenguaje se basa en conceptos tales como objetos y mensajes.
- Algunos de los **entornos** de trabajo son: Smalltalk Express, Pharo, Squeak, VisualWorks, Dolphin.

Se dice que programar en Smalltalk es programar por extensión.

- Diseñar nuevas aplicaciones Smalltalk, requiere de conocimientos sobre las clases existentes en el sistema Smalltalk.
- Las nuevas aplicaciones son construidas por extensión de las librerías de clases de Smalltalk.
- Sus aplicaciones son de alta productividad debido al “reuso” y “extensión del código”.

Mensajes

Los mensajes están compuestos por:

- Objeto receptor.
- Mensaje.
- Selector.
- 0 o más argumentos.
- Valor de retorno (siempre).

Los tipos de mensaje son:

- Unarios.
- Binarios.
- De palabra clave.

Ejemplos:

5 factorial.

3 < 5.

#(4 3 8 1) at: 4 put: 2.

a) **Mensajes unarios:** Los mensajes unarios no tienen argumentos. Interviene un único objeto.

Ejemplo:

5 factorial. -> objeto receptor (5) mensaje, compuesto por el selector (factorial).

El selector es el nombre del método. En este caso: factorial.

Otros ejemplos:

19,76 rounded.

'abcd' size.

a1 verPrecio.

b) **Mensajes binarios:** Tiene un solo argumento. Se utilizan para operaciones lógico-matemáticas. Intervienen dos objetos.

Ejemplo:

3 < 5. -> objeto receptor (3) mensaje, compuesto por el selector (<) y el argumento (5).

El selector es el nombre del método. En este caso factorial. En este caso: <.

Otros ejemplos:

'abc' ~= 'def'. (~= simboliza la palabra distinto/diferente).

True & false. (& simboliza la palabra and, se puede representar también de la siguiente forma: and: [])

7 + 1.

a1 modPrecio: otroPre.

c) **Mensajes de palabra clave:** son mensajes con una o más palabras clave, cada palabra clave tiene un argumento asociado. Estas se reconocen por los dos puntos (:).

Ejemplo:

#(4 3 8 1) at: 4 put: 2. -> objeto receptor (#(4 3 8 1)), mensaje, compuesto por el selector (at: put:) y los argumentos asociados (4, 2).

El selector es el nombre del método. En este caso: at: put:

Otros ejemplos:

5 between: 8 and: 10.

´Hello World´ copyFrom:1 to: 5.

Agencia crear: nom cuil: unCuil dom: unDom.

Orden de ejecución de los mensajes

Los mensajes en Smalltalk se ejecutan en el siguiente orden:

1) Las expresiones que están entre paréntesis ().
2) Las expresiones unarias.
3) Las expresiones binarias.
4) Las expresiones de palabra clave.

Todas de izquierda a derecha.

Ejemplos de mensajes anidados:

2 factorial negated.

3 + 4 * 6 + 3.

5 between: 1 and: 3 squared + 4.

Ejemplos de mensajes en cascada:

En este caso, se envían mensajes sucesivos al mismo objeto receptor.

'sol' size.

'sol' inspect.

- **'sol' size; inspect.**

Ejemplos de polimorfismo:

1) 5 + 100. Valor de retorno: 105

2) (200 @ 200) + 100. Valor de retorno: (300 @ 300)

3) (1/5) + (5/3). Valor de retorno: (28/15)

Variable y asignación

Una variable en Smalltalk representa un puntero a un objeto. Una asignación en Smalltalk tiene el siguiente formato: var := expresión.

Ejemplo:

X := X + 1. (todas las sentencias en Smalltalk terminan en., salvo algunas excepciones).

Smalltalk es un lenguaje no tipado, las variables no se declaran, se asocian a un objeto y a una clase por **binding dinámico.** Hay alocación y eliminación dinámica de objetos (Garbage Collector).

x:= 5.

x:= x+1

(El mensaje +1 es enviado al objeto referido por X. La variable X luego apunta a la expresión resultado de evaluar X+1)

z:= 3. (z apunta al objeto 3 y por lo tanto es de clase integer).

z:= 'hola'. (z ahora apunta al objeto 'hola' y por lo tanto es de clase String).

z:= Array new: 3. (z ahora apunta a un objeto de la clase Array (vector).)

Variables de instancia: propias de cada objeto, representan el Estado Interno del mismo. Es una variable privada del objeto, su valor es unívoco a cada objeto.

Variables de clase: Variable global. Son comunes a todos los objetos de la misma clase, tienen EL MISMO valor para todos los objetos de la clase. Se inicializan una única vez.

Variables temporales: locales a un método o aplicación.

Argumentos: parámetros en los métodos.

Estructuras de control

En Smalltalk no existen las estructuras de control, se simulan, están implementadas en términos de objetos y mensajes.

a) Selección condicional:

· (expresión booleana) ifTrue:[TrueBlock]

ifFalse:[FalseBlock].

- (expresión booleana): ifTrue:[TrueBlock].
- (expresión booleana): ifFalse:[FalseBlock].

Los objetos boolean true y false aceptan los mensajes de palabra clave ifTrue: ifFalse:

La expresión booleana es evaluada, dará un objeto true o false como resultado. Si el resultado es true el mensaje ifTrue: ifFalse: con sus argumentos será enviado al objeto true, sino será enviado al objeto false.

b) Repetición condicional:

- **[expresión booleana] whileTrue:[cuerpo del loop].**
- **[expresión booleana] whileFalse:[cuerpo del loop].**

c) Repetición de longitud fija:

- **valorInicial to: valorFinal do: [:variable del loop | cuerpo del loop].**
- **valorInicial to: valorFinal by: paso do: [variable del loop | cuerpo del loop].**

El mensaje valorInicial to: valorFinal do: unBloque evalúa el argumento unBloque para cada entero que esté en el intervalo dado por el valor inicial hasta el valor final incluido.

Jerarquía de clases (Class Hierarchy Browser)

Las clases están organizadas en forma jerárquica dando origen a un árbol cuya raíz es la clase Object.

Clase char

Los caracteres son objetos que representan los símbolos que conforman un alfabeto. Se los denota con un signo $: $a $c $D $+ $;

Entienden los siguientes mensajes: desarrollado en el PowerPoint de 2da clase.

Clase String

Un objeto de la clase String es una cadena de caracteres encerrados entre comillas simples. Es un objeto indexado y cada uno de sus componentes es un objeto de la clase Char.

Ej. 'hola' es igual a: $h,$o,$l,$a donde la coma concatena.

Además de las operaciones de comparación admite las siguientes operaciones: desarrollado en el PowerPoint de 2da clase.

Clase number

La clase Number encapsula protocolo común para sus subclases: Integer, Float y Fraction.

Alguno de los mensajes que entiende son: desarrollado en el PowerPoint de 2da clase.

Mensajes que entienden todos los objetos:

unaClase new (crea una instancia vacía)

unObjeto class (me devuelve la clase del objeto receptor)

unaClase superClass (me devuelve la superclase de una clase)

unObjeto isKindOff: aClass (me devuelve V o F si unObjeto pertenece a la clase aClass o no)

unObjeto isNil (devuelve V o F si es nil o no)

unObjeto notNil (devuelve V o F si es nil o no)

unObjeto yourself (devuelve el objeto receptor del mensaje)

unObjeto inspect (abre una ventana especial que me permite inspeccionar al objeto receptor para ver su estado y/o modificarlo)

Mensajes para ingresar datos por pantalla

variable:= Prompter prompt: ‘ título del mensaje de ingreso de datos’. //ingresa un string siempre.

resp:=MessageBox confirm: 'desea continuar?’. //ingresa un booleano

x:= (Prompter prompt: ‘ingrese valor') asNumber. //convierte a número entero o real.

Mensajes para imprimir datos por pantalla

MessageBox notify: ‘El nombre y apellido del alumno es:’, nomyape. //la coma concatena.

MessageBox notify: 'la distancia entre los puntos es:', (x displayString). //displayString convierte a string.

Transcript nextPutAll: ‘string’.

Transcript show: ‘string’.

suma inspect. //abre una ventana de inspección y muestra el contenido del OR.

Declaración y asignación de variables

No es necesario declarar las variables, pero se enuncian al principio de la aplicación entre pipes y separadas por espacio:

|var1, var2, var3, …, varN|

La asignación se realiza con :=

var1 := ´Paradigmas de Programación´

var2 := 'Comisión S21'

var3 := Curso crearCurso: var1 comisión: var2 (pasaje de parámetros en smalltalk)

Constructor -> es un método o pseudométodo que permite crear y cargar a la vez un objeto.

Herencia en Smalltalk:

- **Generalización:** se agrupan clases creando superclases que tengan características comunes. Es decir, se abstraen características comunes a dos o más clases formando una clase más abstracta.
- **Especialización:** se agrega un nuevo nivel a la jerarquía, es decir se agrega comportamiento a la jerarquía creando nuevas subclases.
- **Herencia**: Mecanismo mediante el cual se pueden definir nuevas clases basadas en otras ya existentes, a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados.

Hay dos tipos de herencia:

- **Herencia de estructura:** en Smalltalk es total, las subclases heredan todos los atributos de la superclase. No hay forma de no heredar algún atributo.
- **Herencia de comportamiento:** es parcial, se puede no heredar un método de una superclase, redefiniéndolo en la subclase.

Ventajas de la herencia:

1. Se escribe menos código.
2. Favorece el mantenimiento del programa.

Clase 3 – Smalltalk. Diseño de clases simples.

1) Diseño de una nueva clase.

Se debe especificar la clase para definir el protocolo o vista externa de la nueva clase.

Protocolo de clase: es la descripción del protocolo entendido por una clase. Contiene los métodos de creación de objetos y manipulación de variables de clase. Conjunto de métodos de clase, sólo puede llevarlos a cabo la clase. Un método de clase es un Método de palabra clave, me permite crear e inicializar un objeto/instancia.

Protocolo de instancia: es la descripción del protocolo entendido por las instancias de una clase. Contiene los métodos de manipulación de las variables de instancia de un objeto. Métodos que se pueden aplicar sobre los objetos ya creados.

Variables de clase: son variables cuyo valor es compartido por todas las instancias de una clase.

Variables de instancia: denotan la información privada o estado de una instancia de una clase.

2) Especificación de la clase Libro.

1er paso. **Definir qué hace la clase**. La clase Libro permite almacenar los datos de un libro de la biblioteca y el dni del socio que lo retiró.

2do paso**. Definimos el nombre de la clase, de qué clase es subclase, variables de instancia y variables de clase**. Clase Libro, Subclase de: Object, Variables de instancia: isbn titulo autor editorial estado dni.

3er paso**. Definimos el protocolo de clase.** Métodos de clase: crearLibroIsbn: unIsbn tit: unTit aut: unAut edit: unaEdit. Este método retorna una instancia de Libro inicializada (**definimos qué hace el método).**

4to paso. **Definimos el protocolo de instancia.** Métodos de instancia: iniLibroIsbn: unIsbn tit: unTit aut: unAut edit: unaEdit. Este método inicializa una instancia de libro.

5to y 6to paso. **Definimos los ver/consultar y los modificar.**

>>verIsbn

"Retorna el isbn del libro"

>>verTitulo

"Retorna el título del libro"

>> verAutor

"Retorna el autor del libro"

>> verEditorial

"Retorna la editorial del libro"

>> verEstado

"Retorna el estado del libro"

>> verDni

"Retorna el dni del socio que sacó el libro; se le asigna 0 si no está prestado"

>>modiIsbn: unIsbn

“Modifica el isbn del libro”

>>modiTítulo: unTit

“Modifica el título del libro”

>> modiAutor: unAut

“Modifica el autor del libro”

>> modiEditorial: unaEdit

“Modifica la editorial del libro”

>> modiEstado

“Modifica el estado del libro”

>> modiDni: unDni

“Modifica el dni del socio que retiró el libro; se debe asignar un 0 si nadie lo tiene”

3) Implementación de la clase Libro.

Métodos de clase:

>>crearLibroIsbn:unIsbn tit:unTit aut:unAut edit:unaEdit

“Retorna una instancia de Libro inicializada”

^(self new) iniLibroIsbn:unIsbn tit:unTit aut:unAut edit:unaEdit.

El símbolo circunflejo hace referencia a que el método retorna un valor. self new hace referencia a que la clase se va a pedir a si misma que realice una operación (new).

^(super new) iniLibroIsbn:unIsbn tit:unTit aut:unAut edit:unaEdit.

super new hace referencia a que a la clase le pedirá a su super clase que realice una operación (new).

Métodos de instancia:

>>iniLibroIsbn:unIsbn tit:unTit aut:unAut edit:unaEdit

“Inicializa una instancia de Libro”

isbn:=unIsbn.

titulo:=unTit.

autor:=unAut.

editorial:=unaEdit.

estado:=false. “se asigna false cuando no está prestado”

dni:=0. “se asigna 0 en la creación ya que no está prestado”

Implementación de la clase Libro en Smalltalk

>>verIsbn

“Retorna el isbn del libro”

^ isbn.

>>verTitulo

“Retorna el título del libro”

^ titulo.

>> verAutor

“Retorna el autor del libro”

^ autor.

>> verEditorial

“Retorna la editorial del libro”

^ editorial.

>> verEstado

“Retorna el estado del libro”

^ estado.

>> verDni

“Retorna el dni del socio que retiró el libro”

^ dni.

>> modiIsbn:unIsbn

“Modifica el isbn del libro”

isbn:=unIsbn.

>> modiTit:unTit

“Modifica el título del libro”

tit:=unTit.

>> modiAutor:unAut

“Modifica el autor del libro”

autor:=unAut.

>> modiEditorial:unaEdit

“Modifica la editorial del libro”

editorial:=unaEdit.

>> modiEstado

“Modifica el estado del libro”

estado:=estado not. “le asigna el valor opuesto al estado”

>> modiDni:unDni

“Modifica el dni del socio que retiró el libro; asignar 0 en caso de que no lo tenga nadie”

dni:=unDni.

Clase 4 – Smalltalk. Diseño de clases compuestas

1. Especificación de la clase Biblioteca

Clase BIBLIOTECA, es subclase de object.

Variables de instancia: nombre libros (cuando tengo una clase compuesta que tiene una variable de instancia que es un conjunto/colección, no se inicializa pasandole argumentos al método crear).

Métodos de clase:

>>crearBiblioNom:unNom

“Retorna una instancia de Biblioteca inicializada”

Métodos de instancia:

>>iniBiblioNom:unNom

“Inicializa una instancia de Biblioteca”

>>verNom

“Retorna el nombre de la Biblioteca”

>>modiNom:unNom

“Modifica el nombre de la Biblioteca”

>>agregarLibro:unLibro

“Agrega un libro a la Biblioteca”

>>eliminarLibro:unLibro

“Elimina un libro de la Biblioteca”

>>existeLibro:unLibro

“Retorna V si el libro está en la Biblioteca, F en caso contrario(cc)”

>> esVacia

“Retorna V si la Biblioteca no tiene libros, F en cc”

>>buscarLibroIsbn:unIsbn

“Retorna el libro de Isbn unIsbn si existe, nil en cc”

>>todosLosLibros

“Retorna los libros de la Biblioteca” (retorna una ordered collection de libros de la biblioteca)

>> cantidadLibros

“Retorna la cantidad total de libros de la Biblioteca”

>>recuperarLibro:pos

“retorne el libro de la posición pos de la biblioteca”

2. Implementación de la clase Biblioteca

Métodos de clase:

>>crearBiblioNom:unNom

“Retorna una instancia de Biblioteca inicializada”

^(self new) iniBiblioNom:unNom.

Métodos de instancia:

>>iniBiblioNom:unNom

“Inicializa una instancia de Biblioteca”

nombre:= unNom.

libros:= OrderedCollection new.

>>verNom

“Retorna el nombre de la Biblioteca”

^ nombre.

>>modiNom:unNom

“Modifica el nombre de la Biblioteca”

nombre:=unNom.

>>agregarLibro:unLibro

“Agrega un libro a la Biblioteca”

libros add:unLibro.

>>eliminarLibro:unLibro

“Elimina un libro de la Biblioteca”

libros remove:unLibro. (variante: libros remove: unLibro ifAbsent: [^ nil].)

>>existeLibro:unLibro

“Retorna V si el libro está en la Biblioteca, F en caso contrario(cc)”

^ libros includes:unLibro.

>> esVacia

“Retorna V si la Biblioteca no tiene libros, F en cc”

^ libros isEmpty.

```
>>buscarLibroIsbn:unIsbn

“Retorna el libro de Isbn unIsbn si existe, nil en cc”

libros do:[:lib|(lib verIsbn = unIsbn)ifTrue:[ ^ lib]].

^ nil.

>>todosLosLibros

“Retorna los libros de la Biblioteca” (retorna una ordered collection de libros de la biblioteca)

^ libros

>> cantidadLibros

“Retorna la cantidad total de libros de la Biblioteca”

^ libros size

>>recuperarLibro:pos

“retorne el libro de la posición pos de la biblioteca”

^ libros at:pos.
```

Clase Collection

Tipos de colecciones:

- OrderedCollection.
- SortedCollection.
- Array.
- Dictionary.

Una colección es un objeto que puede contener un número variable de objetos de distinto tipo.

Generalmente usamos OrderedCollection (donde importa el orden de inserción: al principio, atrás, en una posición determinada, y los elementos están enumerados: 1°, 2°, 3°, etc) o SortedCollection

(donde importa el orden bajo algún criterio: ascendente o descendente). En ambos casos se accede por índice.

La OrderedCollection se crea con un new, es una colección dinámica, y es heterogénea. Tiene 3 variables de instancia: Contents, startPosition, endPosition.

Otros ejemplos de colecciones son: set (donde importa que no haya elementos repetidos), bag (cuando interesa saber cuántas veces se repite el mismo objeto), etc. En estos dos casos, no se puede acceder por índice y el manejo de los elementos es aleatorio.

Las colecciones pueden responder a los siguientes mensajes:

Operaciones para agregar datos:

add: unObj (agrega unObj al final y modifica punteros de posición).

addFirst: unObj (agrega unObj al principio y modifica punteros de posición).

addLast: unObj (agrega unObj al final y modifica punteros de posición).

at: unaPos put: unObj (pone unObj en la posición unaPos sobrescribiendo lo que ya exista, o sea, ya debe haber un elemento en esa posición, sino dará un error).

Siempre se reenumeran los elementos en caso de incorporación o eliminación de elementos, por lo cual para usar los mensajes at: y at: put: los índices deben ser correctos, sino dará mensaje de error.

Operaciones para eliminar datos:

remove: unObj (borra el objeto unObj si existe, sino da error)

removeFirst (borra el primer elemento si no está vacia la colección, sino da error)

removeLast (borra el último elemento si no está vacia la colección, sino da error)

remove: unObj ifAbsent: unBloque (borra unObj si éste existe, sino ejecuta unBloque)

En todos los casos devuelve el elemento eliminado.

Operaciones para consultar:

at: unaPos (devuelve el objeto que está en la posición unaPos)

size (devuelve la cantidad de elementos)

isEmpty (devuelve true si está vacía o false en caso contrario)

includes: unObj (devuelve true si incluye al objeto unObj y false en caso contrario)

first (devuelve el elemento de la primera posición)

last (devuelve el elemento de la última posición)

occurrencesOf: unObj (devuelve la cantidad de veces que unObj aparece en la colección)

col1:= col asSet (devuelve en col1 una colección de los mismo objetos sin que hayan objetos repetidos)

Ejemplos:

c:= OrderedCollection new. c=[] c size. à 0 c isEmpty. à true

c add: 23. c=[23]

c add: 'vacío' c=[23 'vacío']

c at:1 put: 'hola' c=['hola' 'vacío']

c size. à 2

c isEmpty. à false

c includes: 'hola'. à true

c includes: 'Pepe'. à false

c at: 1. à 'hola'

c remove: 'hola'. c=['vacío']

c remove: 'papa'. à Error

c remove: 'papa' ifAbsent: [Transcript show: 'no existe ese elemento'].

Clase 4 – UML

El **lenguaje de modelado unificado (UML)** es un estándar para la representación visual de objetos, estados y procesos dentro de un sistema.

Por un lado, el lenguaje de modelado puede servir de modelo para un proyecto y garantizar así una arquitectura de información estructurada; por el otro, ayuda a los desarrolladores a presentar la descripción del sistema de una manera que sea comprensible para quienes están fuera del campo.

UML se utiliza principalmente en el desarrollo de software orientado a objetos.

Los diagramas UML se utilizan para representar los siguientes componentes del sistema:

- Objetos individuales (elementos básicos).
- Clases (combina elementos con las mismas propiedades).
- Relaciones entre objetos (jerarquía y comportamiento/comunicación entre objetos).
- Actividad (combinación compleja de acciones/módulos de comportamiento).
- Interacciones entre objetos e interfaces.

1. Diagrama de clases

Un **diagrama de clases** es un diagrama de estructura estática que describe la estructura de un sistema. Permite modelar sus clases, atributos, operaciones y relaciones entre objetos.

Un diagrama de clases describe:

- Los tipos de objetos en el sistema.
- Las relaciones estáticas que existen entre ellos.
- Los atributos y operaciones de las clases.
- Las restricciones a las clases y a sus asociaciones.

Decimos que un diagrama de clases es **un diagrama estático**, ya que no muestra cómo interactúan las clases en tiempo real, sino cómo están definidas y relacionadas en el código fuente o en el diseño de software. Un diagrama de clases se centra en la representación de las clases, sus propiedades (atributos) y las relaciones entre ellas (asociaciones, herencias, agregaciones, composiciones, etc.). Esto permite a los desarrolladores comprender la organización de las clases y cómo se relacionan entre sí en el sistema.

Componentes básicos de un diagrama de clases

En UML las clases se representan mediante un rectángulo que puede estar dividido en tres partes.

- **Sección superior**: Contiene el nombre de la clase.
- **Sección central**: Contiene los atributos de la clase.
- **Sección inferior**: Incluye operaciones de clases (métodos), organizadas en un formato de lista. Cada operación requiere su propia línea.

Relaciones entre clases

Las relaciones existentes entre las distintas clases nos indican cómo se comunican los objetos de esas clases entre sí.

Las relaciones más importantes son:

a) **Asociación:** conexión entre clases.

b) **Herencia:** generalización y especialización.

Asociación: es una relación estructural que describe una conexión entre objetos. Se dice que es estructural porque modifica la estructura del objeto.

Elementos de una asociación:

- Nombre.
- Rol(es).
- Multiplicidad (establece restricciones de existencia para los objetos de las clases asociadas). 1 (uno y sólo uno), 0..1 (cero o uno), m..n (de m a n siendo ambos enteros naturales), * (varios), 0..* (de cero a varios), 1..* (de uno a varios).

Agregación y composición: Son casos particulares de asociaciones. Muestran la relación entre un todo y sus partes. Gráficamente, se muestran como asociaciones con un rombo en uno de los extremos.

Agregación: Las partes pueden formar parte de distintos agregados.

Composición: Las partes sólo existen asociadas al compuesto, sólo se accede a ellas a través del compuesto.

Herencia: Es el proceso en el que una subclase o clase derivada recibe la funcionalidad de una superclase o clase principal, también se conoce como "generalización".

2. Diagramas de secuencias

Diagramas de interacción: Son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento. Hay dos tipos: **Diagramas de secuencia** y **Diagramas de colaboración.**

Diagrama de secuencias: Es dinámico. Despliega la secuencia de acciones a seguir cuando un objeto recibe un mensaje.

- Un objeto se muestra como caja en la parte superior de una línea vertical punteada. Esta línea vertical se llama **línea de vida del objeto** y representa la vida del objeto durante la interacción.
- Cada mensaje se representa mediante una flecha entre las líneas de vida de los objetos. El orden en el que se dan estos mensajes transcurre de arriba hacia abajo.
- Cada mensaje es etiquetado por lo menos con el nombre del mensaje. Pueden inclusive agregarse los argumentos y alguna información de control.

- Un objeto puede enviarse un mensaje a sí mismo, eso se llama autodelegación. Se representa con una flecha que sale de la línea de vida del objeto y vuelve a la línea de vida del mismo objeto.

Información de control:

- **Condición**: el mensaje se envía sólo si la condición es verdadera. Se simboliza con [condición].
- **Marcador de iteración:** un mensaje se envía muchas veces a varios objetos receptores, como sucedería cuando se itera sobre una colección. Se simboliza con *[mensaje].
- El diagrama puede incluir el regreso de un mensaje, este regreso se representa con una línea punteada. Pueden no representarse.

Clase 5 – Colecciones

Jerarquía de magnitud -> desarrollado en el cuaderno.

Clases abstractas -> son aquellas de las que no podemos tener instancias. Es decir, no podemos instanciarlas. Ej: Magnitud, Number, Integer.

Colecciones abstractas à Collection, IndexedCollection, FixedSizeCollection.

Tipos de colecciones

Bag -> son colecciones no ordenadas (no indexables), dinámicas, que pueden contener elementos repetidos. Guarda el objeto una sola vez, acompañado de la cantidad de veces que se encuentra repetido en la colección.

Set -> Son colecciones no ordenadas (no indexables), que no admiten elementos repetidos.

Interval -> Representan rangos o intervalos de números. Por ejemplo, el intervalo de números de 1 a 100 se define de la siguiente manera: Interval from: 1 to: 100.

Algunas colecciones en Smalltalk pueden ser heterogéneas. Por ejemplo: Array, OrderedCollection, SortedCollection. String es homogéneo, está formado sólo por caracteres.

a) **Array** -> (subclase de FixedSizeCollection) es una colección homogénea, indexada, de tamaño fijo. El array no admite el add: y tampoco el remove: ya que es una estructura estática. Permite utilizar el at: put: El mensaje size siempre me da el tamaño fijo (no el corriente). El mensaje isEmpty da falso siempre porque genera un vector con elementos creados en nil.

b) **OrderedCollection ->** (subclase de IndexedCollection) es una colección heterogénea, indexada, dinámica. Tiene 3 variables de instancia: contents (contenido o elementos), startPosition y endPosition. La operación new crea un objeto vacío. Siempre se reenumeran los elementos en caso de incorporación o eliminación de elementos, por lo cual para usar los mensajes at : y at : put : los índices deben ser correctos sino da mensaje de error. Para simular pila y cola basta con combinar operaciones de esta clase. Con addFirst - removeFirst o

addLast - removeLast simulamos pila (último en llegar, primero en salir, entonces guardo y saco del mismo extremo). Con addLast - removeFirst o addFirst - removeLast simulamos la cola (primero en llegar primero en salir, entonces guardo en un extremo y saco del otro).

La OrderedCollection admite el at: y el at: put:, con ellos podemos manejar la colección en forma indexada. Solo pueden usarse si la posición ya tiene contenido asignado.

c) **SortedCollection** à (subclase de OrderedCollection) es una colección indexable, heterogénea, dinámica, sus elementos están ordenados bajo algún criterio específico. Están anuladas las operaciones addLast:, addFirst:, at: put:

sc:=SortedCollection new. /*crea una colección vacía cuyos elementos por defecto serán ordenados de menor a mayor cuando se agreguen, solo en caso de ser objetos de clases simples como números y strings*/

El método add agrega un elemento a la SortedCollection en forma ordenada.

Si se necesita que los elementos estén en orden descendente hay que usar el método de clase **sortBlock[ ]** para crear la colección con un criterio de orden distinto. Este método de clase ejecuta el new y le establece el criterio de ordenamiento que esté definido entre los corchetes [].

En caso de que sean objetos compuestos, como libros, personas, productos, etc. debe usarse el método de clase **sortBlock[ ]** e indicar el criterio de ordenamiento:

s:=SortedCollection sortBlock [:x :y| x verNombre <= y verNombre].

En la clase SortedCollection también existe el método de instancia **sortBlock: [ ]** que permite reordenar una colección ya cargada bajo otro criterio.

s:= s sortBlock:[:x :y| x verPrecio >= y verPrecio].

Este método cambia el criterio de ordenamiento y reordena todos los elementos de s.

d) Dictionary -> (subclase de Set) Almacena pares de valores clave:valor. La clave es única. La clave puede ser cualquier objeto, y su valor lo mismo. Son heterogéneas. Agrega al final, y se accede por clave. Se almacenarán la clave junto a su valor siempre y cuando la clave no se repita. Si defino a una clave que ya existe y le pongo otro valor, se sobrescribe el valor nuevo sobre el anterior. Puedo también agregar una Asociación creada previamente. Una Asociación está formada por una clave y un valor. La forma de agregar una asociación es: d add: unaAsociación, previamente habiendo creado dicha asociación.

Testeo y Conversión de colecciones.

Testeo de colecciones:

- Size
- isEmpty
- Includes: unObjeto
- ocurrencesOf: unObjeto

Conversión de colecciones:

- asSet
- asBag
- asArray
- asOrderedCollection
- asSortedCollection

Nota: Los métodos de conversión de colecciones retornan una nueva colección del tipo que se especifique. No modifican la colección original.

Enumeradores de colecciones

do: itera sobre cada elemento de la colección (trabaja con el puntero).

to: do: repite un número conocido de veces un proceso (trabaja con el índice)**.**

No se debe agregar ni remover un elemento dentro de un do: Porque trabaja internamente con el tamaño de la colección.

Uso del select, reject, collect y detect

Todos ellos están implementados internamente con un do:

select à recorro con condición, genera una colección nueva con elementos que cumplen con la condición.

col1:= alumnos select:[:alu| alu verNota=8].

Retorna en otra colección los alumnos con nota igual a 8.

reject -> crea una nueva colección con los elementos que no cumplen con la condición dentro de los corchetes.

col2:= alumnos reject:[:alu| alu verNota=8].

Retorna en otra colección los alumnos con nota distinto a 8.

collect -> Colección del mismo tamaño que la original, aplica una operación sobre los elementos de la colección. Crea una nueva colección únicamente con el objeto resultante de la operación dentro de los corchetes.

col3:= alumnos collect:[:alu| alu verNombre].

Retorna en otra colección los nombres de los alumnos.

detect -> devuelve el primer objeto que encuentre que cumpla con la condición dentro de los corchetes. Si no lo encuentra devolverá un error. Para evitar eso se utiliza el ifNone:

a:= alumnos detect:[:alu| alu verNota=8] ifNone:[nil].

Retorna el primer alumno con nota igual a 8 o nil si no hay ninguno.

Clase 6 – Herencia

Jerarquías de clases

En POO un programa puede concebirse como una red de objetos que interactúan enviándose mensajes.

Cuando se diseña un sistema orientado a objetos se debe tener en cuenta las relaciones que existen entre las clases de objetos ya que existe una jerarquía de objetos y no todos las clases se encuentran en el mismo nivel. La jerarquía puede tener varios niveles.

En una jerarquía puede haber clases genéricas o abstractas y clases concretas.

De las clases abstractas no se pueden tener instancias, de las clases concretas sí.

Una clase es subclase de otra cuando especializa los conceptos definidos en ésta.

Las superclases generalizan a las subclases agrupando comportamiento común de las mismas.

Al programar se buscan definir conceptos abstractos y luego los subconceptos, subclases concretas.

Podemos decir que definir jerarquías de clases consiste en definir un árbol donde más abajo pongo las clases más concretas y más arriba las clases más abstractas.

Mecanismo de herencia en POO

Generalización -> se agrupan clases creando superclases que tengan características comunes. Es decir, se abstraen características comunes a dos o más clases formando una clase más abstracta.

Especialización -> se agrega un nuevo nivel a la jerarquía, es decir se agrega comportamiento a la jerarquía creando nuevas subclases.

Herencia -> la herencia es el mecanismo mediante el cual se pueden definir nuevas clases basadas en otras ya existentes, a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados.

La herencia puede ser simple o múltiple.

Simple -> una subclase puede tener una sola superclase (Smalltalk).

Múltiple -> una subclase puede tener 2 o más superclases (C++, Python).

Hay dos tipos de herencia: de estructura y de comportamiento.

Herencia de estructura -> **en Smalltalk es total**, las subclases heredan todos los atributos de la superclase. No hay forma de no heredar algún atributo.

Herencia de comportamiento -> **es parcial**, se puede no heredar un método de una superclase, redefiniéndolo en la subclase.

Ventajas de la herencia

1) Se escribe menos código, se reutiliza el código.
2) Favorece el mantenimiento del programa. Lo poco que agrego es lo que tengo que chequear. La modularidad me facilita el trabajo de mantenimiento.

Para programar usando herencia, se utilizan las pseudo variables **self** y **super.**

- self: hace referencia al objeto receptor del mensaje.
- super: hace referencia a la superclase inmediata de la clase que contiene el método en el cual aparece dicha pseudo variable.

Clase 7 – C++

Características del lenguaje de programación C++:

- Es un lenguaje compilativo híbrido.
- Todo programa en C++ consta de **objetos**.
- Todo objeto tiene un **estado**, dado por sus **datos miembros**, y un **comportamiento**, dado por sus **funciones miembros**.
- Todo objeto es una **instancia** de una **clase**.
- La clase **encapsula** el estado y el comportamiento.
- Los objetos se comunican vía **mensajes**. P.e. si p es un objeto de la clase Persona puedo escribir el siguiente comando: p.verEdad, donde le envío el mensaje verEdad al objeto P.
- El envío del mensaje desencadena la ejecución de la función miembro asociada.
- C++ tiene 2 funciones miembro especiales: Constructor y destructor.

Constructor -> se llama igual que la clase, se invocan al definir el objeto, admite parámetros, no retorna valores, si no se define, el compilador lo crea automáticamente.

Ejemplo: class Persona à Persona p (= new en smalltalk)

Destructor -> El nombre es la negación del nombre de la clase, no admite parámetros, no retorna valores.

Ejemplo: -Persona p.

Definición de una clase

Se deben especificar sus dos componentes:

- Una zona de declaración: contiene la lista de datos miembro (variables de instancia de smalltalk).
- Una zona de implementación: define e implementa las funciones miembro (métodos de smalltalk).

Una clase puede contener una parte pública y una privada. Por defecto la clase es **privada.**

Toda clase tiene una **visibilidad**: privada, pública o protegida. La **visibilidad** define qué se puede acceder y cuál es el tipo de acceso: Lo público lo accedo directamente, lo protegido y lo privado sólo a través de funciones miembro.

Herencia

C++ presenta herencia simple y también múltiple. La subclase hereda datos miembro y funciones miembro de su/s superclase/s. No hereda el constructor ni el destructor. Agrega datos miembro y funciones miembro.

Definición de una subclase

Se debe determinar de quién es subclase y el tipo de derivación: **pública, protegida y privada.** El tipo de derivación defina el acceso que tiene la subclase a los datos miembro de la superclase. Los datos miembros NO se heredan.

Subclase:

- **Publica** -> hereda la parte pública y protegida de la superclase, con igual visibilidad.
- **Protegida** -> hereda la parte pública y protegida, ambas como protegidas.
- **Privada** -> hereda la parte pública y protegida, ambas como privadas.

Clase 7 – Python

Características de Python

Python es un lenguaje de programación interpretado. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje dinámico y multiplataforma. Python, al igual que C++, presenta herencia simple y múltiple.

Definición de una clase

En Python para definir una nueva clase se coloca la palabra class seguida de su nombre:

- Class Auto:
- Class Mascota:

Por convención, los nombres de clases **deben comenzar en mayúscula.**

Dentro de la clase se especifican los métodos y los atributos comunes a todos los objetos de dicha clase. Todo lo que va dentro de una clase se escribe con sangría. La primera sentencia que no esté en sangría, no pertenece a la clase.

Constructor -> Los constructores se ejecutan automáticamente justo después de crear o instanciar un objeto. Los constructores son métodos especiales mediante los cuales los programadores pueden inicializar una instancia. Los constructores en Python se definen codificando un método especial llamado **_init_.**

En el **_init_** se inicializan las variables de instancia del objeto, dejándolas disponibles para comenzar a operar con ellas a través de los métodos.

Nota: el guion bajo sirve para indicar que los atributos son privados.

Uso de self

La palabra reservada **self** sirve para **referirse al objeto actual**, que es el que recibe el mensaje. Es utilizada dentro del método para señalarse a si mismo y sirve para poder acceder a los atributos y métodos del objeto actual. En todos los métodos de una clase el primer parámetro **es siempre self.**

Creación de objetos

Para crear un objeto hay que instanciarlo a partir de una clase.

Ejemplo: from clamascota import *

```
M = Mascota('beto', 'perro', 'Jorge', 5)
```

La forma en que los métodos son invocados difiere en la forma de invocar a las funciones. Se debe utilizar la notación de punto para invocar los métodos.

Ejemplo:

```
m.modNom('lolo')

print(m.verEdad())
```

Creación de una subclase con herencia simple

En Python, para definir una subclase, se coloca la palabra class seguida de su nombre y entre paréntesis el nombre de la superclase:

- **Class Auto(Vehiculo):**
- **Class Mascota(Animal):**

Hereda todos los atributos y métodos.

Creación de una subclase con herencia múltiple

- **class Base1: pass**
- **class Base2: pass**
- **class Multiderivada(Base1, Base2): pass**

En este caso, cuando busca un método o atributo lo hace en la clase Multiderivada primero, luego va a Base1 y finalmente a Base2 (de izq a derecha según definición de derivación).

Paradigma Funcional

Definiciones de este paradigma:

- **Programa:** conjunto de definiciones de funciones o ecuaciones matemáticas, cuya evaluación permite obtener el resultado de una consulta.
- **Sistema Operativo:** tomando el rol de "calculadora", evalúa las funciones y devuelve un único resultado.
- **Paradigma Declarativo:** paradigma donde se dice qué acciones ejecutar, pero no como ni en qué orden ejecutarlas. El paradigma funcional pertenece a esta agrupación.
- **Función:** para cierto argumento existe un único resultado posible, sin importar el orden en que se evalúen las expresiones matemáticas pertenecientes al programa, siempre se obtendrá el mismo resultado (principio de unicidad).
- **Variable:** no es una celda de memoria cuyo contenido puede cambiar durante la ejecución, el valor de la variable matemática se liga al argumento de la función y no cambia.
- **Transparencia Referencial:** propiedad que asegura que durante la ejecución del programa no habrá efectos laterales que modifiquen el valor final resultante. Cosa que el Paradigma Imperativo no cumple.
- **Script:** ambiente donde se detalla el tipo de función (dominio, codominio) y la ecuación que la describe junto con su nombre, argumento y valor. Donde entre el nombre y el valor se define un binding.
- **Efecto lateral:** cuando modifico el valor de una variable cuando se está ejecutando una función, cosa que no se puede en funcional.

Definición de funciones

Se debe definir un **script o ambiente** en el que se detalla:

· Tipo de la función (dominio y codominio).

· La ecuación que describe a dicha función.

Tipos de Datos:

- **Standard:** num (Int y Float), bool, char, string, a (alpha).
- **Derivado:** par ordenado, lista, función.

Particularidades del Paradigma:

- No existen estructuras de control. Sino recursividad.
- Las funciones pueden incluir condiciones.

Formas de Definir Funciones:

- Por ecuaciones simples.
  cuadrado(x):= x*x

- Por casos o Guardas:
  min(x, y):= x, if x < y
  := y, otherwise

- Con definiciones locales:
  Cuarta:= cuad(x)*cad(x)
  where cuad(x):= x*x

- Por patrones:
  and(x, true):=x
  and(x, false):=false.

- Recursivas:
  fact(0):=1
  fact(x):= x* fact(x-1)

- De alto orden (composición de funciones): map, filter, take, take while.

Reducción: consiste en sustituir la función por su definición y simplificar, a la forma más simple o forma normal (canónica). El orden de reducción es la evaluación de la función, y no importa el orden, siempre debe dar el mismo resultado.

Evaluación de Funciones:

- **Orden normal:**
  ?cuad(5 + 4)
  (5+4)* (5+4)
  9 * (5+4)
  9 * 9
  81

- **Orden aplicativo:**
  ?cuad(5 + 4)
  cuad(9)
  9 * 9
  81

Lista: conjunto de elementos homogéneos que puede ser…

- Vacía: [ ]
- Unitaria: [X]
- Simple: (x:xs); x->cabeza xs->cuerpo [2, 4, 5, 5]
- De listas: (xs:xss); xs->cabeza xss->cuerpo [[3, 5 ,6], [3, 4], [8]]

Operaciones sobre Listas:

- Concatenación.
  - [ ] : Xs = Xs : [ ] = Xs
  - (Xs :Ys) : Zs = Xs : (Ys : Zs)
- Longitud.
  - #[ ] =0
  - # (Xs : Ys)= #Xs + #Ys
- Head.
  - hd(X : Xs)= X hd(Xs: Xss)= Xs
- Tail.
  - tl(X : Xs)= Xs tl(Xs:Xss)= Xss
- map.
  - cubo(X):=X*X*X
  - cuboL(X:Xs):= map (cubo(X)) Xs
- filter.
  listaPares(X:Xs):= filter(par(X)) Xs
  where par(X) :=true, if X mod 2=0
  := false, otherwise

- take.
  - take(0, [21 ,3 ,19]) → [ ]
  - take(2, [21, 3, 19]) → [21, 3]
- takewhile.
  - takewhile(par(X), [24, 6,3,18,25])

Preguntas Típicas de Parcial:

- ¿Por qué pertenece al declarativo?
- ¿Cómo está escrito un funcional?
- ¿Cómo se define una función?
- ¿Cuáles son las distintas maneras de definir ecuaciones matemáticas: patrones, etc…?

Paradigma Lógico

Características:

- El paradigma trabaja en **forma descriptiva,** es decir, no indica cómo resolver el problema sino que establece las relaciones entre las entidades del mismo, las cuales describen qué hacer.
    - Al trabajar en forma descriptiva se lo encuadra en la programación declarativa (definición ya dada en funcional). Se describe qué hacer pero no cómo.
- El programa lógico no posee un algoritmo que lo oriente para llegar a una resolución. Sino que está formado por **expresiones lógicas** que declaran o describen la solución y es el sistema interno quien proporciona la secuencia de control que se utilizan esas expresiones.
- Usa **reglas lógicas** o **predicados de 1er orden** para inferir (deducción de una respuesta) o derivar conclusiones a partir de los datos.
- *En síntesis,* conociendo los datos y las condiciones del problema, la ejecución de un programa consiste en demostrar un objetivo a partir de las relaciones declaradas.
- Las variables en lógico son matemáticas.

Un algoritmo lógico se construye:

- Especificando una **Base de Conocimiento.**
- Se realizan **consultas** sobre dicha base.
- Se aplica un **mecanismo de inferencia o deducción** sobre dicha base.
- Con dicho mecanismo **se infieren** conclusiones.

Base de conocimiento (BC):

- Se construye con predicados de 1er orden (si p -> q) llamados ***Cláusulas de Horn*** las cuales, a partir de un conjunto de datos del problema, permiten la deducción de conclusiones (está constituído por hechos y reglas, se le hace consulta y a través de un motor de inferencias y saca una conclusión):

c :=p1, p2, p3, p4,…….pk

c: conclusión o consecuente/consecuencia.
pi: premisas o antecedentes.

- "c" es verdadera si todas las premisas "pi" son verdaderas.
- "c" está formada por hechos y reglas.
- **Hechos:** enunciados o predicados (parte de la BC) siempre verdaderos que muestran relaciones entre los datos del problema (axiomas).
- **Reglas:** muestran relaciones entre los datos, los hechos y otras reglas, y permiten deducir o demostrar objetivos.

• Las reglas muestran relaciones entre los datos, los hechos y otras reglas, y permiten deducir o demostrar objetivos.

• Hechos → hombre(juan).
hombre(tomás).
mujer(ana).
progenitor(ana, juan).

• Reglas → mortal(X):- humano(X).
cabeza cuerpo
que significa: X es mortal si X es humano
Si X es humano entonces X es mortal.
madre(X,Y):- mujer(X), progenitor(X,Y).

Consultas: objetivo a ser demostrado por **comparación** o **pattern machine** contra la *base de conocimiento.* Donde se trata de aparear o emparejar la consulta con algún hecho o con la cabeza de una regla. Puede ser de dos tipos:
• Validación (cuya respuesta es Sí o No).
• Una Búsqueda (que retorna los valores que hacen verdadero al objetivo).

- **Si hay variables entonces es un proceso de búsqueda.**

Si puede demostrar la consulta u objetivo, responde:
• Yes (si es una validación).
• O el/los valor/es que satisface/n la consulta (en caso de búsqueda).

Si no puede demostrar el objetivo, responde **No,** que tiene dos significados:
• Que es falso
• O que no puede deducirlo con la información contenida en la base de conocimiento

- Ejemplo:

?- hombre(juan)
Yes
?- hombre(luis)
No
?hombre(X)
X=juan;
X=tomás;
No

BC

hombre(juan).
hombre(tomás).
mujer(ana).
mujer(pilar).
progenitor(ana, juan).

Observación:
Si colocamos un ; después de un resultado, sigue mostrando otros resultados posibles hasta agotarlos todos.
Si colocamos un . el proceso de búsqueda se corta.

Unificación: si la consulta posee variables, las mismas se asocian o ligan con los valores de los hechos o con los argumentos de las cabezas de las reglas (con los predicados).
Transparencia Referencial: a las variables no se les asigna un valor, ni una celda de memoria. Sino que como una variable matemática, o en este caso, **lógica** se ligan un valor que no cambia, no hay efectos laterales.
Backtracking: método por el cuál se realiza una búsqueda generando un árbol de búsqueda por una rama y de no encontrarse va por otra, eso de forma recursiva.
Resolución (proceso de demostración): al emparejar las variables de consulta con la cabeza o predicado de una regla las variables se ligan y luego se aplican una regla. Se usa su definición y debe demostrarse cada sub-objetivo de la misma.

Clasificación por Variables: otra forma de clasificar las consultas es a través de la relación o la no relación con las variables lógicas:

- *Sin Variables:* ?gusta(maria, juan) a maria le gusta juan Validación.
- *Con Variables:* ?gusta(maria, X) quién le gusta a maría Búsqueda.
- *Con Variable Anónima:* ?gusta(maria,_) hay alguien que le gusta a maría. La respuesta es Sí o No.
-

“_”: es una variable anónima, es como un existe.

Motor de Inferencias: es quien controla la ejecución del programa lógico a partir de la consulta. Recorre la base de conocimiento desde el principio y permite derivar o deducir un objetivo en base a dos mecanismos: *recursión y backtracking.* Aplicando pattern machine, unificación y reemplazo.

```
eshombre(juan).
eshombre(pedro).
eshombre(luis).
quiere(juan,maria).
quiere(juan, elena).
quiere(pedro, raquel).
quiere(pedro, belen).

?-eshombre(X),quiere(X,Y)

X=juan Y=maría;
X=juan Y=elena;
X=pedro  Y=raquel;
X=pedro  Y=belen;
No
```

Proceso de resolución: evalúa de izq a derecha.
Toma 1er objetivo: eshombre(X):
Liga X=juan y reemplaza en el segundo objetivo
?quiere(juan,Y) busca todos los posibles valores de Y
X=juan Y=maría;
X=juan Y=elena; No hay más alternativas posibles
Por backtracking retrocede al objetivo previo: eshombre(X)
Liga X=pedro y reemplaza en el segundo objetivo
?quiere(pedro,Y)
X=pedro Y=raquel;
X=pedro Y=belen; No hay más alternativas posibles
Por backtracking retrocede al objetivo previo: eshombre(X)
Liga X=Luis y reemplaza en el segundo objetivo
?quiere(Luis,Y)
No encuentra ninguna cláusula en la BC que satisfaga el segundo objetivo, termina el proceso y muestra No.

Reversibilidad:

- Se pueden introducir valores en la consulta esperando que el motor de inferencias encuentre las variables de salida que apareen o unifiquen con la consulta.

```
?- progenitor(ana, X)
X=juan;
No
```

De quién es progenitor ana?

```
BC
mujer(ana).
hombre(juan).
progenitor(ana, juan).
```

- Recíprocamente también puede introducir un valor de salida y esperar que el motor encuentre qué valores de entrada son adecuados para satisfacer esa salida.

```
?- progenitor(Y, juan)
Y=ana;
No
```

quién es el progenitor de juan?

¿Cómo se construye la base del conocimiento? Así:

Tipos de Reglas:

- Simples
  - notrabaja(belen).
  - bueno(pedro).

  } hechos
  - cuida(belen, pedro):- notrabaja(belen), bueno(pedro).

- Con variables
  - humano(juan).
  - humano(ana).

  } hechos
  - mortal(X):- humano(X).

- Recursivas
  - antepasado(X,Y):- progenitor(X,Y).
  - antepasado(X,Y):- antepasado(X,Z), progenitor(Z,Y).

Pasos de la Programación Lógica:

1. Modelar el problema mediante una Base de Conocimiento formada por hechos que muestren las relaciones de los datos de dicho problema.
2. Formular reglas que se cumplan en el universo del problema modelado.
3. Formular consultas al sistema que demuestre o deduzca la respuesta a partir de la base de conocimiento usando resolución.

Usos y Aplicación del Paradigma Lógico:

- Inteligencia Artificial.
- Sistemas expertos.
- Robótica.
- Procesamiento de Lenguaje Natural.
- Compiladores.
- Bases de datos deductivas.
- Acceso a bases de datos desde páginas web.

PROLOG (Programmation in Logique): lenguaje interpretado que consta de un compilador, un intérprete, un shell y una biblioteca. Es un híbrido.

- Sintaxis:
  - Constantes y predicados comienzan con minúscula
  - Variables comienzan en mayúscula
  - Hechos y predicados terminan en punto
  - Se usa ; en una consulta para ver más resultados
  - El <enter> o el . finaliza la consulta
  - Los comentarios se empiezan con %

- Predicados predefinidos:
  - true
  - fail
  - var(X) devuelve V si X es una variable sin instanciar
  - nonvar(X) devuelve T si X tiene valor asignado
  - integer(X) devuelve V si X es una variable entera
  - float(X) devuelve V si X es una variable real
  - number(X) devuelve V si X es una variable numérica
  - write(X) imprime el valor de X
  - read(X) ingresa un valor a X
  - == compara si son iguales
  - \== compara si son distintos

  Operadores: + - / * X mod Y

- **Sin instanciar:** no está unificada, no tiene un valor asignado.
- **Hipótesis del mundo cerrado:** lo que no está escrito no posee valor. Surge de la teoría de bases de datos. Esta suposición establece que todo lo que es relevante en el mundo ha sido especificado. Por lo tanto, esto permite a un agente asumir de forma “segura”que un hecho es falso si no puede inferir que es verdadero.

```
posit(X,Z):- X>0, Z is false.  %se usa la hipótesis de mundo cerrado
```

Listas en Prolog: conjunto de elementos homogéneo:

- Se definen: [1, 2, 3, 4]
- Están divididas en cabeza (x) y cuerpo (xs) -> [X | Xs]
- Lista vacía: [ ]
- Se recorren recursivamente.
- Contienen términos (variables, constantes o estructuras) en general. Es posible anidar listas.

Ejemplos:
• [1,2,3,4]
• [a,b,c,d]
• [1,’a’, X,[1,2,3]]
• [[1,2],[3,4],[5,6]]

- Los ejemplos están dados en la filmina de la clase “9 Continuación” al ser esto un resumen no se ha agregado, pero se informa por las dudas.

longitud de una lista

```
long([], C):- is 0.
long([X|Xs], C):- long(Xs, C1), C is C1+1
```

suma de una lista

```
sum([], 0). //igualmente válido
sum([X|Xs], S):- sum(Xs, S1), S is S1 + X
```

suma([2, 7, 3], su)
x->2
Xs-> [7, 3]
s -> su

suma([7, 3], su)
x->7
Xs-> [3]
s -> su

suma([3], su)
x -> 3
xs-> []
s-> su

suma([], su)
su = 0

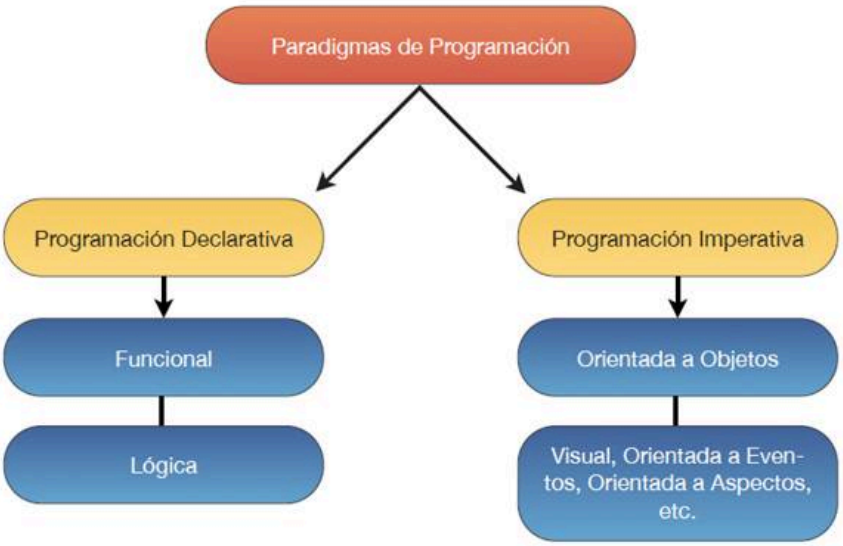
se llaman todas igual pero estan en cajas de registros de activación diferentes de distintas

0 + 3 + 7 + 2 con los su

- ¿Cómo es que la base de conocimiento demuestra los predicados? A través de un proceso de resolución 😀.

La hipótesis de mundo cerrado es booleano, validación. Todo lo que no está en la Base de conocimiento es falso.

Fin de Paradigmas en Programación


Paradigmas de Programación
Programación Declarativa
Programación Imperativa
Funcional
Orientada a Objetos
Lógica
Visual, Orientada a Even-
tos, Orientada a Aspectos,
etc.