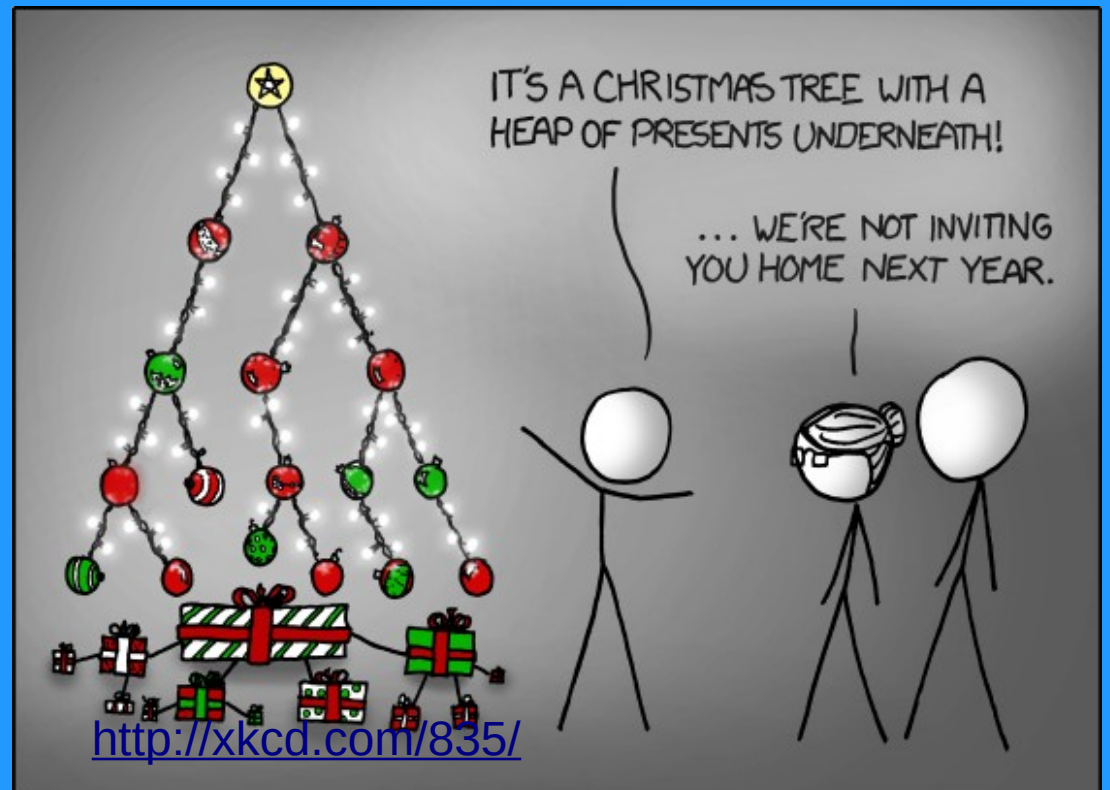


# Estructuras de Datos

## 2020



# **Especificación de TAD**

# TAD. Abstracción

## ¿Qué es una abstracción?

Una abstracción es un proceso mental, mediante el cual se extraen **los rasgos esenciales de algo** para representarlos por medio de un lenguaje gráfico o escrito.

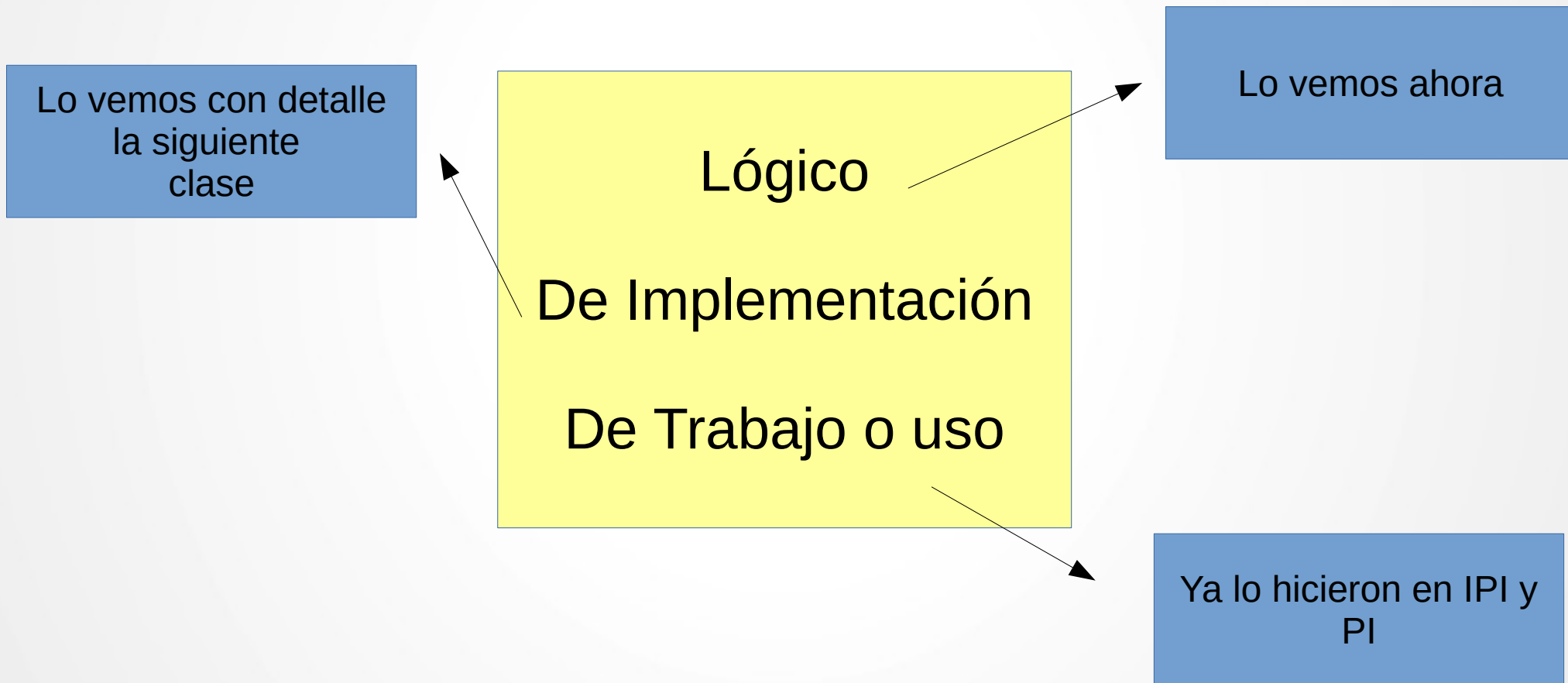
Puesto que es un proceso mental, la abstracción es una acción subjetiva y creativa, esto es, depende del contexto psicológico de la persona que la realiza.

# TAD

¿Qué es un Tipo de Dato Abstracto (TAD)?

La técnica de la abstracción de datos establece que al diseñar una nueva estructura de datos, ésta pasa a ser un Tipo de Dato Abstracto (TAD), **que podrá implementarse en cualquier lenguaje** y aplicarse en cualquier concepto.

# Niveles de Trabajo



# TAD. Especificación

La especificación lógica de un TAD es un documento en el que se plasma la abstracción realizada al diseñar una estructura de datos.

Dicho documento pasará a ser el **mapa o plano mediante el cual se construirá (implementará) la estructura de datos** y en el que se definirá claramente las **reglas** en las que podrá usarse (aplicarse) el TAD.

# TAD. Especificación

La especificación lógica de un TAD debe abarcar en general los siguientes puntos:

- Elementos que conformarán la estructura de datos
- Tipo de organización en que se guardan los elementos: lineal, jerárquica, en red, sin relación
- Dominio de la estructura: se describe si es necesario la capacidad de la estructura en cuanto al rango posible de datos a guardar.
- La descripción de las operaciones de la estructura

# Abstracción de Datos

## Niveles de Trabajo

**Nivel lógico:** que corresponde a la especificación lógica del TAD. En este nivel se define abstractamente la estructura de datos y las operaciones relacionadas con ella. **La descripción que se obtenga en este nivel debe ser *independiente* del lenguaje de programación en el que se implementará o usará la estructura.**



# Abstracción de Datos

## Niveles de Trabajo

**Nivel de implementación:** En este nivel se decide el lenguaje de programación en que se implementará la estructura, qué tipos de datos ya definidos servirán para representarlas (estructura interna) y, finalmente, bajo estas consideraciones, se implementa a cada una de las operaciones del TAD.

# Independencia de Datos Ocultamiento de Información

Hay una **independencia** bien marcada entre **el nivel de implementación y el nivel de aplicación del TAD.**

Esto significa que quien implementa el TAD **no** debe estar influenciado por la aplicación que tendrá la estructura y que quién use la estructura **no** tiene porqué saber cómo se implementaron las operaciones.

# Independencia de Datos Ocultamiento de Información

se dice que la forma en que se almacenan los datos en la estructura es independiente de su aplicación y que para el usuario programador permanece oculto cómo se implementaron las operaciones del TAD.

El implementador del TAD, podrá hacer cambios o mejoras a su implementación, respetando la especificación lógica y **sin afectar las aplicaciones desarrolladas**.

No afecta la sintaxis de las operaciones

¿Puede afectar los Tiempos y Órdenes de Ejecución?

# TAD. Nivel Lógico

Vemos un ejemplo..

**Pensemos en un vector dinámico...**

Estructura: es lineal, homogénea y de acceso directo. Puedo acceder a un elemento directamente por su posición en el vector.

**¿Operaciones?**

# TAD. Nivel Lógico

**Pensemos en un vector dinámico...**

**-Obtener un elemento**

**-Insertar un elemento**

**-Consultar la cantidad de elementos...**

**¿Operaciones?**

**¿Otras?**

# TAD. Nivel Lógico

Algunas Operaciones:

**¿Están bien las operaciones?**

**¿Problemas?**

`InsertarValor(Vector, v)`  
inserta el valor `v` en `Vector`

`ObtenValor(Vector, v)`  
Retorna un valor `v` de `Vector`

`Longitud(Vector)`  
Retorna la cantidad de valores que tiene `Vector`

# Más Estructuras...

**Pilas**

**Colas**

**Listas**

**Listas Circulares**

**Listas Dobles**

**Conjuntos**

¿Otras?

# Pilas y Colas

## **Pilas y Colas:**

Son estructuras de datos lineales con restricciones en cuanto a la posición en la cual se pueden llevar a cabo las operaciones de inserción y eliminación de componentes.



# TAD. Abstracción

**Pensemos en una Pila...**



# Pilas

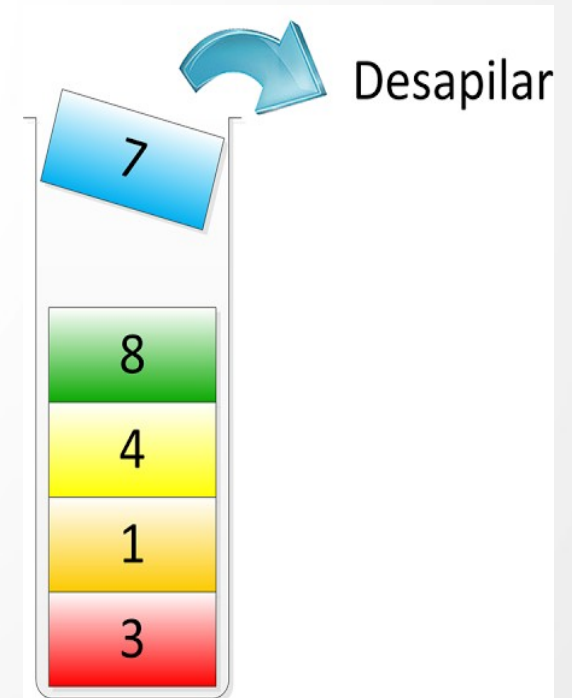
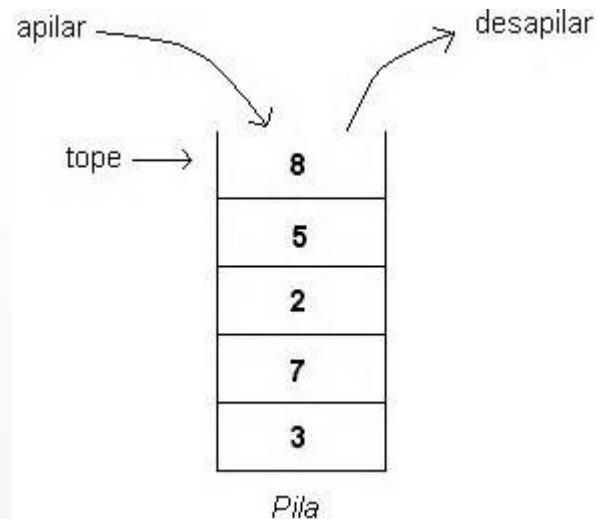
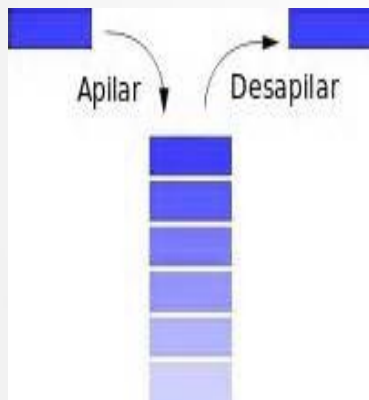
Una pila representa una estructura lineal de datos en la que se puede agregar o quitar elementos únicamente por uno de los dos extremos.

En consecuencia, los elementos de una pila se eliminan en orden inverso al que se insertaron; es decir, el último elemento que se mete en la pila es el primero que se saca.

Una pila es una estructura de datos lineal, como un arreglo; y se define formalmente como una colección de datos a los cuales se puede acceder mediante un extremo, que se conoce generalmente como **tope**.

# Pila

Suele denominarse **lifo** (last in, first out); es decir, el último en llegar es el primero en salir.



# Enlace interesante para Visualizar...

<http://visualgo.net/list.html>

<http://www.cs.usfca.edu/~galles/visualization/Algorithms.html>

# Pila

Pensemos en las operaciones:

Creación

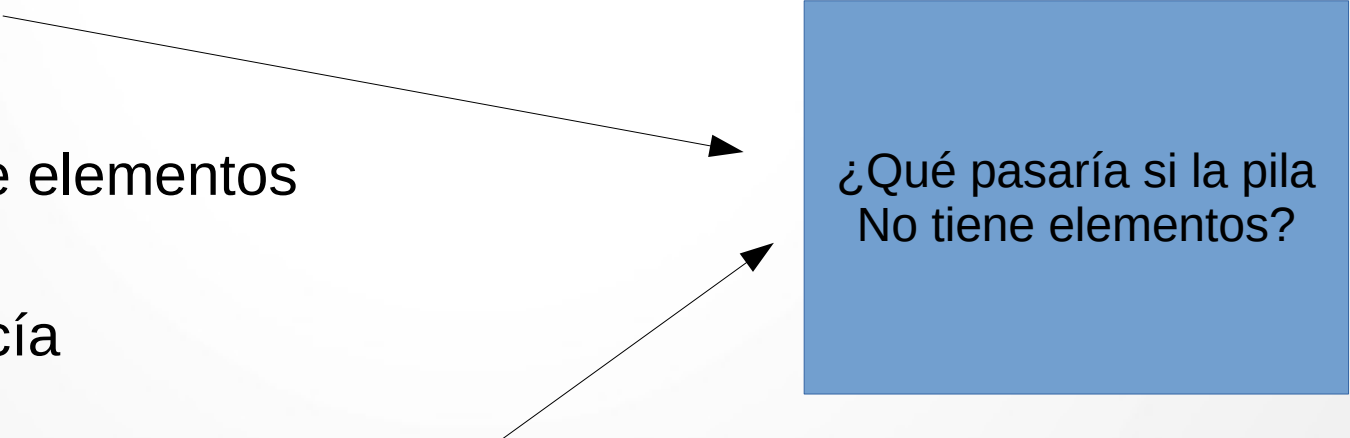
Agregar un elemento

Quitar un elemento

Conocer el número de elementos

Saber si está o no vacía

Conocer el elemento que está por encima en la pila



¿Qué pasaría si la pila  
No tiene elementos?

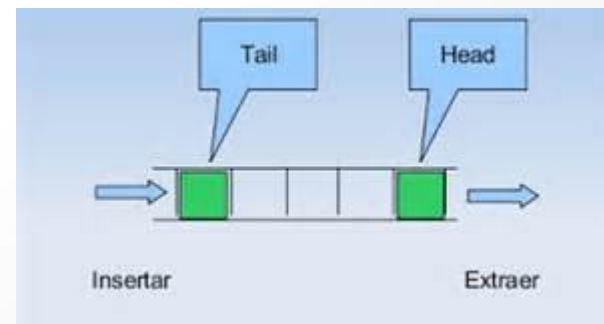
# Cola

**Nos seguimos concentrando en la Parte Visible o Pública (Interfaz)**

**Ahora pensemos en una Cola...**

**Acceso: los elementos se añaden por un extremo (final) y se sacan por el otro extremo (frente)**

**Estructura FIFO (First Input First Output)**



# TAD. Especificación

Como vamos a escribir las operaciones para:

- No olvidar ninguna
- que sean consistentes
- No ambiguas
- Completas
- Cómo indicamos sus restricciones...

# Ejemplo. Cola

**Tomemos el ejemplo de la Cola...**

**Cola: es un grupo ordenado (con respecto al tiempo que llevan en él) de elementos homogéneos (todos del mismo Tipo)**

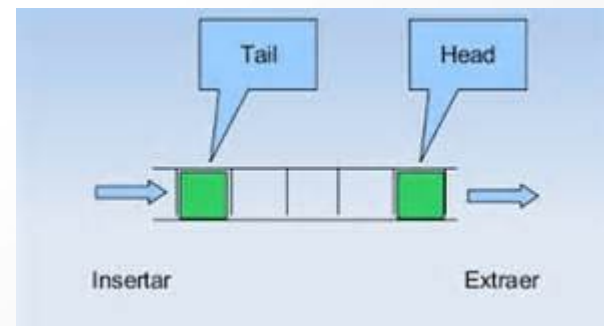




# Ejemplo. Cola

**Acceso:** los elementos se añaden por un extremo (final) y se sacan por el otro extremo (frente)

Estructura **FIFO** (First Input First Output)



# TAD. Especificación

Como vamos a escribir las operaciones para:

- No olvidar ninguna
- que sean consistentes
- No ambiguas
- Completas
- Cómo indicamos sus restricciones...

**Vamos a construir un mapa/plano/molde  
Usando un Lenguaje Formal**

# TAD. Especificación

Al resolver un problema tenemos que  
Comprender el problema:

- Queremos documentar detalles.
- Intercambiar opiniones con otros



**Necesitamos una descripción precisa**



# TAD. Especificación

## Descripción precisa

El Castellano no es bueno para lograrla ya que:

- Su Naturaleza es ambigua
- Es poco preciso

Ejemplo:

Quien es envidioso:  
¿Carlos?  
¿ su tío?

**Carlos no quiere a su tío porque es envidioso**

# TAD. Especificación

## Descripción precisa

El pez está listo para comer.

¿El pez está listo para ser comido o está listo para que le den de comer?

El Barza le ganó al Villarreal en su estadio.

¿En qué estadio? ¿En el del Villarreal o el del Barza?

# TAD. Especificación

## Descripción precisa

El ejemplo no tiene mucha importancia e interpretarlo incorrectamente no parece tener mayores consecuencias...

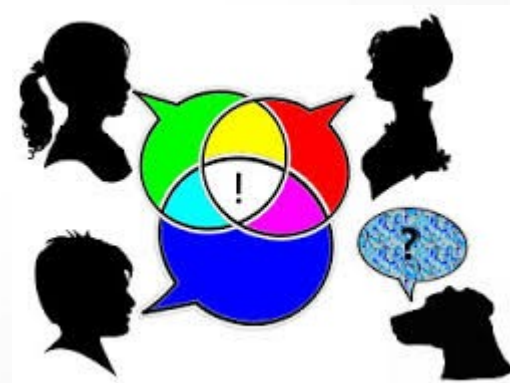
Pero si queremos describir “un Controlador de Tráfico Aéreo” o “El controlador del motor de un auto” pequeños errores podrían tener **consecuencias graves**.

**Los lenguajes formales** tienen propiedades que los hacen aptos para esta tarea.

# **Lenguajes Formales**

# TAD. Lenguajes Formales

Podemos definir en matemáticas, lógica y ciencias de la computación un “lenguaje formal” como un lenguaje cuyos símbolos primitivos y reglas para unir esos símbolos están formalmente especificados.





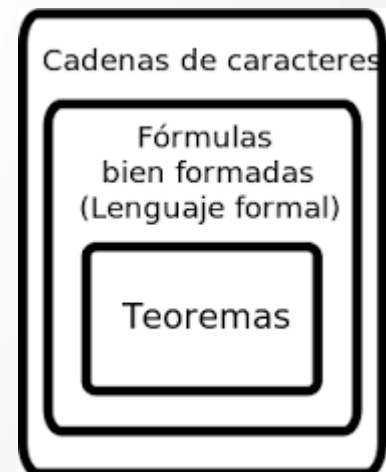
# TAD. Lenguajes Formales

**Alfabeto:** Conjunto de símbolos primitivos

**Gramática formal:** Conjunto de Reglas (describe la sintaxis)

**Formula bien formada:** Una cadena de símbolos formada de acuerdo a la gramática (palabra)

Estrictamente hablando, un lenguaje formal es idéntico al conjunto de todas sus fórmulas bien formadas



# TAD. Lenguajes Formales

Algunas Ventajas:

- No son ambiguos
- Podemos verificar propiedades



# TAD. Lenguajes Formales

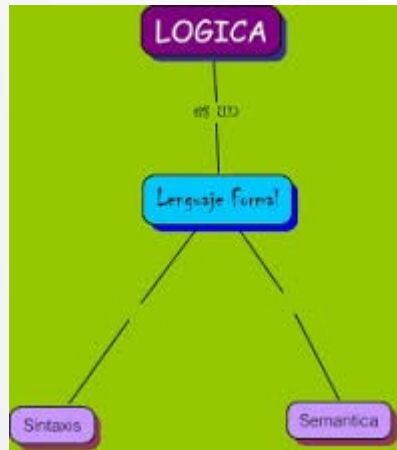
Por ejemplo, un alfabeto podría ser el conjunto  $\{c,g\}$

y una gramática podría definir a las fórmulas bien formadas como aquellas que tienen el mismo número de símbolos  $c$  que  $g$ .

Entonces, algunas fórmulas bien formadas del lenguaje serían:  $cg$ ,  $gc$ ,  $cggc$ ,  $cgcg$ , etc.; y el lenguaje formal sería el conjunto de todas esas fórmulas bien formadas.

# TAD. Lenguajes Formales

Lenguaje Tiene



Sintaxis: Como se escriben  
Las oraciones validas en el  
Lenguaje

Semántica: Como se  
entienden (interpretan)  
Esas oraciones validas del  
lenguaje

# TAD. Lenguajes Formales

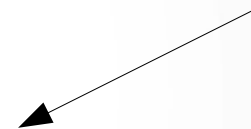
Para algunos lenguajes formales existe una semántica formal que puede interpretar y dar significado a las fórmulas bien formadas del lenguaje.

Sin embargo, **una semántica formal no es condición necesaria para definir un lenguaje formal**

En nuestro ejemplo anterior no les dimos un significado a ggcc o gcgcgc...

# TAD. Lenguajes Formales

Vemos que existen muchos lenguajes formales pero nosotros nos vamos a concentrar en los TAD'S



Modelos matemáticos que se construyen con el fin de exponer los aspectos relevantes del problema bajo análisis

Modelo: determina que elementos del mundo real estarán reflejados en la especificación y que operaciones se permitirá realizar sobre ellos

**Especificamos...**

# TAD. Especificación

Primero vamos a **describir la estructura de datos indicando sus características:**

- Homogénea/Heterogénea
- Si posee restricciones con respecto a la cantidad de elementos
- Lineal/jerárquica, acceso secuencial/indexado/por clave...



# TAD. Especificación

Existen varios enfoques para especificar TADs (Procedural, Funcional, Orientado a Objetos, etc).

Nosotros vamos a usar un enfoque Imperativo para aprovechar la base que tienen (IPI y PI)

Para “hablar todos el mismo idioma” luego de la descripción de la ED en la Práctica vamos a usar el siguiente **“molde” para especificar TADs**

# Molde Especificación TAD

TAD Nombre\_del\_TAD

Igualdad Observacional

Usa

Parámetro Formal

Géneros

observadores básicos

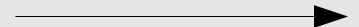
Generadores

otras operaciones

Axiomas

Exporta

Vemos con detalle cada apartado



# Igualdad Observacional

La igualdad observacional “nos dice” cuándo dos instancias del aspecto de la vida real que nuestro TAD está modelando se comportan de la misma manera.

Es un concepto semántico y no sintáctico.

Es como “se ve desde afuera” y no tiene que ver con la implementación

También podemos pensar en cuando “el resultado de comparar usando `==` da True” (OJO sin pensar en la implementación de `==`)

# Igualdad Observacional. Ejemplo: pila

...

**Igualdad Observacional**

**Si  $a$  y  $b$  son dos pilas**

**$a$  es igual a  $b$  si se cumple que:**

**Las longitudes de  $a$  y  $b$  son iguales**

**Y**

**cada elemento en  $a$  es igual al correspondiente  
elemento en  $b$ .**

# Usa

En este apartado incluimos los nombres de otros TADs que necesitemos para poder definir el nuestro.

Por ejemplo:

Ej.: Natural, Bool, Secu(a), LoQueSea, . . .

Para nuestro ejemplo Pila lo completamos al final ya que todavía no sabemos...

# Parámetro Formal

Un parámetro Formal puede entenderse como una “variable del tipo” .

Necesitamos una pila de “algo”..ese algo en el ejemplo de nuestra pila lo vamos a llamar **a**

...

**Parámetro Formal**

**a**

...

# Géneros

Es el nombre con el que se va a hacer referencia a instancias del TAD que estamos especificando.

En nuestro ejemplo le vamos a llamar Pila<a>

...

**Género**

**Pila<a>**

...

# Géneros

Para establecer una analogía,

podríamos pensar que el conjunto  $N$  es el género de los números naturales.

Pensados éstos no sólo como un conjunto de valores, sino también como operaciones entre ellos.



# Observadores Básicos

Son operaciones que **NO** van a cambiar nuestro estado observable.

Sintaxis:

```
Nombre_del-observador (gen_p1, gen_p2 , . . .)→  
género_de_salida  
{Restricción, si la hubiera}
```

Ejemplos:

```
obs1 (elGénero)→ natural  
Obs2 : (loquesea, elGénero)→ bool
```

# Observadores Básicos

Al principio pensamos en estas operaciones:

## **Creación**

Agregar un elemento

Quitar un elemento

Conocer el número de elementos

Saber si está o no vacía

Conocer el elemento que está por encima en la pila

¿Cuales serían  
Observadores?

# Observadores Básicos

Los observadores entonces para  $\text{Pila}\langle a \rangle$  serían:

$\text{tamaño}(\text{Pila}\langle a \rangle) \rightarrow \text{Natural}$

$\text{es\_vacía}(\text{Pila}\langle a \rangle) \rightarrow \text{Bool}$

$\text{tope}(\text{Pila}\langle a \rangle) \rightarrow a$

# Observadores Básicos. Restricciones

Cuando especificamos vamos a encontrar muchos casos donde una operación que estamos definiendo no tiene sentido en ciertos casos..

Por ejemplo: si la operación es “dividir por” tenemos que exigir que el parámetro recibido no debe ser cero

```
tamaño(Pila<a>) → Natural
```

```
es_vacia(Pila<a>) → Bool
```

```
tope(Pila<a>) → a
```

Para los Observadores  
de Pila<a>  
¿Encuentran alguna  
restricción?

# Observadores Básicos. Restricciones

Vamos a escribir las restricciones como Pre y Post condiciones según corresponda:

```
tope(Pila<a>) → a
```

```
{Pre: la pila tiene al menos un elemento}
```

# Restricciones

Las restricciones son una parte fundamental de un TAD:

- Con ellas explicitamos los casos para los cuales ciertas operaciones no tienen sentido

Aportan claridad y coherencia a una especificación.

- Aportan expresividad ya que nos permiten limitar el universo al cual aplican ciertas operaciones de nuestro TAD.

# Relación cercana entre: Igualdad Observacional y Observadores

Si comparamos instancias observacionalmente iguales NO debería pasar que al aplicar un observador a ambas obtengamos resultados observacionalmente distintos.

Por ejemplo:  $a$  y  $b$  son personas y sus edades son observables.

Si sabemos que  $a$  y  $b$  son observacionalmente iguales entonces

Sus edades deberían ser iguales.

# Relación cercana entre: Igualdad Observacional y Observadores

Y si las instancias no son observacionalmente iguales NO debería pasar que al aplicar todos los observadores a ambas obtengamos los mismos resultados.

Si  $a$  y  $b$  tienen:

Documentos iguales

Apellidos iguales y Nombres iguales

Siendo Documentos, apellidos y nombres sus tres Observadores entonces  $a$  y  $b$  deben ser iguales.

Es muy común definir la Igualdad Observacional a partir de la igualdad de los observadores básicos.



# Generadores

Los generadores son un conjunto de funciones que retornan un resultado del género del TAD especificado.

A partir de ellos se pueden construir absolutamente todas las instancias del TAD.

La sintaxis es similar a la de los observadores, con la diferencia de que un generador puede no recibir parámetros, en cuyo caso crea una nueva instancia “de la nada”.

# Generadores

Sintaxis:

**G1() → elGénero**

**G2(loquesea ) → elGénero**

Y en el caso de la pila?

Que generadores podemos tener?

# Generadores

Sintaxis:

```
vacía() → Pila<a>
```

```
{Post: La pila retornada esta vacía}
```

```
a_partir_de(Secuencia<a>) → Pila<a>
```

```
{Post: La pila contiene apilados los elementos  
de la secuencia recibida}
```

# Otras Operaciones

Irían todas las operaciones necesarias (ya sea porque el problema las requiere o porque se usan Como auxiliares), que no sean observadores ni generadores.

En este caso **“vale”** **modificar el estado observable.**

La sintaxis para estas funciones es igual a la de los observadores básicos.

# Otras Operaciones

Ejemplos:

001 (loquesea , elGénero)→ bool

002 (elGénero)→ natural

00<sub>aux</sub> (elGénero , natural)→ natural

En el caso de pila<a>

¿Qué operaciones se les  
ocurren?

# Otras Operaciones

En el caso de Pila<a>:

`apilar(Pila<a>, a ) → None`

`{Pos: la pila no esta vacía}`

`desapilar(Pila<a>) → a`

`{Pre: la pila tiene al menos un elemento}`

`{Pos: la pila perdió el tope que tenía antes de desapilar}`

La operación

`obtener_elemento(natural) → a`

Que retorna el elemento que se encuentra en la posición del parámetro

**¿Tiene sentido en este caso?**

# Axiomas

Explicamos “lo que hacen” los generadores, los observadores básicos y las otras operaciones.

**`vacía()`**: Crea una pila (sin elementos)

**`a_partir_de(Secuencia<a> s)`**: crea una pila que contiene apilados los elementos de la secuencia `s` siendo el último elemento de la secuencia el que quede en el tope de la pila.

**`apilar(Pila<a> p, a elem)`**: apila en el tope de `p` el elemento `elem`.

**`desapilar(Pila<a> p)`**: quita el elemento que se encuentra en el tope de `p`.

**`tamaño(Pila<a> p)`**: Retorna/devuelve la cantidad de elementos de la pila `p`.

# Axiomas

Explicamos “lo que hacen” los generadores, los observadores básicos y las otras operaciones.

**es\_vacíá(Pila<a> p):** Retorna/devuelve verdadero si la pila **p** esta vacía y falso en caso contrario.

**tope(Pila<a> p):** retorna/devuelve el elemento que se encuentra en el tope de la pila **p**.



# Exporta

los géneros, observadores básicos, generadores y todas las operaciones que **puedan ser utilizadas desde afuera** (separados por comas).

```
Pila<a>, vacía, a_partir_de, es_vacía, tope,  
tamaño, apilar, desapilar
```

# Usa

En este apartado incluimos los nombres de otros TADs que necesitemos para poder definir el nuestro.

¿Qué TADs usamos en la especificación?

**Natural, Bool, Secuencia<a>, None**

# **Implementación de TAD**

## **Introducción**

# TAD. Recordamos...

¿Qué es un Tipo de Dato Abstracto (TAD)?

La técnica de la abstracción de datos establece que al diseñar una nueva estructura de datos, ésta pasa a ser un Tipo de Dato Abstracto (TAD), **que podrá implementarse en cualquier lenguaje** y aplicarse en cualquier concepto.

# TAD. Especificación e Implementación

La especificación lógica de un TAD es un documento en el que se plasma la abstracción realizada al diseñar una estructura de datos.

Dicho documento pasará a ser el **plano mediante el cual se construirá (implementará) la estructura de datos** y en el que se definirá claramente las **reglas** en las que podrá usarse (aplicarse) el TAD.

En el TP 1 **Diseñamos** Estructuras para obtener "**Planos**"

En el TP 2 vamos a **Construir** esas estructuras implementándolas en un Lenguaje de Programación

# Tipo de Datos. Recordando...

**Tipo de Datos define:**

los valores que pueden ser representados en un objeto de ese tipo,

como se almacenará en memoria

y que operaciones podrán realizarse sobre él.

# TAD. Algunos conceptos importantes

## **Abstracción**

Posibilidad de reutilizar soluciones que respondan a un Modelo en diferentes problemas

## **Encapsulamiento:**

Definir un nuevo tipo e integrar en él todas las operaciones que se puedan realizar.

## **Ocultamiento de datos:**

Separación de la parte visible (interfaz del TAD) de la implementación

# TAD. Impementación

**Vamos a Definir:**

**Parte visible o pública:** Presenta la “interfaz” necesaria para que cualquiera pueda usar el TAD

**Parte privada:** Oculta los detalles de implementación que NO tiene por que ser conocidos.



# TAD

## Ventajas del uso de TAD

**Vamos a tener representación + comportamiento.**

**Independencia:** Desarrollar en forma aislada. Probar en forma aislada

**Reusabilidad de código :** Utilización tipo “caja negra”.

# TAD

Para “concretar” un TAD necesitamos:

- Encapsulación dentro de una construcción de un lenguaje, tanto para la parte visible como para la privada.
- Poder declarar **tipos protegidos de modo que** la representación interna esté oculta de la parte visible del módulo.
- Manejo de tipos genéricos
- Poder Crear valores de un nuevo tipo definido de esta manera

Python lo soporta  
:-)

# TAD. Implementación



Especificación



Implementación

**¡En el “siguiente capítulo” vamos a Implementar la Pila que especificamos usando Python!**