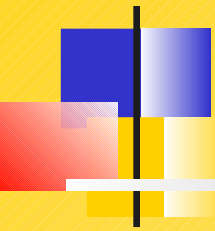
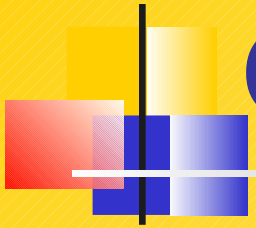


Pensando OO

ELO329: Diseño y programación orientados a objetos

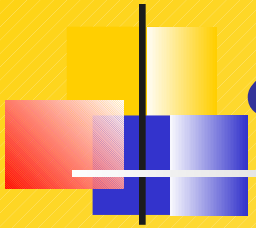
Agustín J. González

1s09



Orientación a Objetos

- OOP (Object Oriented Programming) es una idea distinta de otras en programación
- OOP es un paso en la evolución de previos abstracciones de programación



¿Por qué OOP es popular?

- La esperanza que rápidamente y fácilmente conducirá a aumentar la productividad y mejorar confiabilidad. De la mano viene Diseño Orientado a Objetos.
- El deseo de una transición simple de lenguajes existentes
- La similitud con técnicas de pensamiento sobre problemas en otras áreas

¿Por qué OOP es popular?

- La programación de un computador aún es una de las tareas más difíciles enfrentadas por el hombre;
- Llegar a ser hábil en programación requiere talento, creatividad, inteligencia, lógica, la habilidad de construir y usar abstracciones, y experiencia.
- Programación Orientada al Objeto es una nueva forma de pensar sobre qué significa hacer cómputos, sobre cómo podemos estructurar información al interior de un computador.

Lenguaje y Pensamiento

- En otras palabras la forma como hablamos influye en la manera como vemos el mundo (y viceversa).
- Esto es válido no sólo para los lenguajes naturales (español, inglés, mapuche...) sino también para los lenguajes artificiales como los de programación (C, Pascal, C++, Java...)
- Ejemplo:
  - En MATLAB los loops son lentos de procesar, pero las matrices son muy rápidas. Se sugiere ver las soluciones operando matrices y no vía ciclos for.
- Corolario: los nombres de objetos e identificadores son relevantes. Después de un tiempo no hay pensamiento sólo nombres en el código.

Lenguaje y Pensamiento (Cont.)

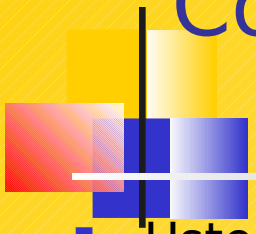
- Hipótesis de Whorf: Trabajando en un lenguaje, es posible que un individuo imagine pensamientos o ideas que no pueden ser trasladadas o entendidas por individuos trabajando en otro contexto lingüístico.
  ¿Entendemos los problemas de origen étnico y religioso en algunos países del planeta?
  Blanco, blanco, blanco, blanco ¿Qué líquido toma la vaca?
  => nuestro conocimiento da forma a nuestras soluciones
- Conjetura de Church: Cualquier computación para la cual existe un procedimiento efectivo puede ser realizada por una máquina de Turing. (ésta dispone de una máquina de estados y una cinta donde se puede escribir y borrar). En los 60s se demostró que esta máquina podía ser emulada por cualquier lenguaje con la sentencia condicional.
  => Una máquina muy simple puede resolverlo todo

Lenguaje y Pensamiento (Cont.)

- Entonces en qué quedamos, estas dos ideas parecen contradictorias. Hay ideas que no son entendidas en otros contextos lingüísticos v/s la máquina de Turing con sólo sentencia if es capaz de hacerlo todo. ¿Para qué aprender otra cosa?

- Técnicas de orientación al objeto no proveen ninguna capacidad computacional nueva que permita resolver problemas no solubles por otros medios. Pero estas técnicas conducen a soluciones más fáciles y naturales (para el hombre) y favorecen la administración de grandes proyectos.

Computación como Simulación



- Ustedes pueden estar acostumbrados al modelo proceso-estado. Al estilo de la máquina de Von Newman. El computador sigue un patrón de instrucciones, organizadas en la memoria, saca valores desde varias localizaciones, los transforma, y pone resultados en otras localizaciones.
- Este modelo no ayuda mucho para entender cómo resolver problemas reales. No es la forma de pensar y ver las cosas.
- La visión de la OOP es crear un “universo” y es en muchas formas similar al estilo de la simulación llamado “simulación conducida por eventos”
- En OOP, tenemos la visión de computación como simulación.
- Cuando los programadores piensan en los problemas en términos de comportamientos y responsabilidades de objetos, ellos aprovechan su gran intuición, ideas, y entendimiento ganado con la experiencia diaria.
- Ver ejemplo líneas y puntos

Tratando la complejidad

- Los problemas más complejos son comúnmente abordados por un equipo de programadores.
- La interconexión entre componentes es tradicionalmente complicada y por ello gran cantidad de información debe ser intercambiada entre los integrantes de un equipo.
- La incorporación de más gente puede **alargar** el proyecto en lugar de acortarlo.
- **El principal mecanismo para controlar la complejidad es la abstracción**. Ésta es la habilidad de encapsular y aislar localmente información de diseño.

Una nueva forma de ver el mundo

- **Supongamos que deseo enviar flores a mi abuelita. Una forma es ir a la florería y pedirlo a la vendedora. Le doy el tipo de flores y la dirección donde enviarlas.**
- **Yo busco un "*agente*" (la vendedora) y le paso un *mensaje* con el requerimiento. Es la *responsabilidad* de la vendedora el enviar las flores. Hay un *método*, conjunto de operaciones o algoritmo, usado por a vendedora para hacer esto. Yo no necesito conocer el detalle de este método.**
- **Las acciones son iniciadas en OOP por la transmisión de un mensaje (invocación de un método) a un agente (un *objeto*) responsable por la acción.**
- **Dos ideas: *Ocultar información* (hacer saber sólo lo indispensable o principio de "Information Hiding") y Reutilización. Sacarse la idea de tener que escribir todo y no usar servicios de otros (*delegar*).**
- **Dos importantes diferencias entre invocar Procedimientos y Mensajes**
  - **En mensajes hay un *receptor* designado, en procedimientos no.**
  - **La interpretación del mensaje (método) depende del receptor. Por ejemplo, mi esposa no actuaría igual si le pido enviar las flores.**
- **Usualmente el receptor de un mensaje no se conoce hasta tiempo de ejecución. Decimos que la *ligazón* entre mensajes (nombre de función, procedimiento o método) y el fragmento de código (implementación) usado para responder es determinada en tiempo de ejecución.**

Reutilización del software

- La gente se ha preguntado con frecuencia por qué el software no puede semejarse a la construcción de objetos materiales. Éstos normalmente son construidos a partir de otros elementos ya existentes y depurados.

- Por ejemplo: El acceso a una tabla indexada para buscar objetos es una operación común en programación; sin embargo, esta operación es re-escrita en cada nueva aplicación. Normalmente esto pasa porque los lenguajes tradicionales tienden a relacionar muy fuertemente el tipo del elemento con el código para insertar o buscar los elementos.

- El uso de técnicas de programación orientada al objeto debería conducir a generar gran número de componentes de software re-utilizables.

Ejemplo: Un Stack

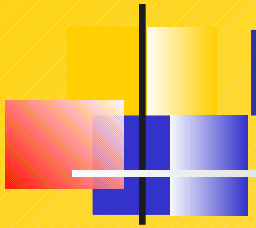
- Este ejemplo ilustra las limitaciones de algunos lenguajes para ocultar información y desacoplar tareas.

```
int datastack[100];
int datatop = 0;
void init() {
  datatop=0;
}
void push (int val) {
  if (datato <100)
        datastack [datatop++] = val;
}
void top () {
  if (datatop >0 )
        return (datastack [datatop-1]);
}
int pop() {
  if (datatop >0)
        return (datastack [--datatop]);
  return 0;
}
```

- Los datos del stack no pueden ser locales a cada función
- Sólo hay dos opciones: Locales o Globales -> Globales
- Globales -> no hay forma de limitar la visibilidad de esos nombres.
- El nombre datastack debe estar en conocimiento de los otros programadores.
- Los nombre init, pus, top, pop, ya no pueden ser usados.

Ejemplo: Un Stack

- Definiendo el alcance dentro de un bloque (como en Pascal)
- begin
  var
    datastack: array [1..100] of integer;
    datatop: integer;
  procedure init; ....
  Procedure push(val: integer); ....
  Procedure pop : integer; ....
  .....
  end;
- Al dar acceso a la interfaz (o “protocolo” o nombre de funciones y profedimientos), también se está dando acceso a sus datos comunes (datatop) , al dar acceso los nombres quedan “tomados”.
- La idea está contenida en los dos principios de David Parnas:
  1.- Proveer al usuario (otro programador, por ejemplo) toda la información necesaria para usar correctamente un módulo, y NADA MÁS.
  2.- Se debe proveer al implementador con toda la información que él necesita para completar el módulo, y NADA MÁS.



Responsabilidades

- Un concepto en OOP es describir comportamientos en términos de **responsabilidades**. Esto permite aumentar el nivel de abstracción y mejora la independencia entre agentes (importante para resolver problemas complejos).
- La colección de responsabilidades asociadas con un objeto es descrita por el término **protocolo**.
- No pregunte por lo que tú puedes hacer a tu estructura de datos, sino pregunta lo que tu estructura de datos puede hacer por ti.

Clases e Instancias

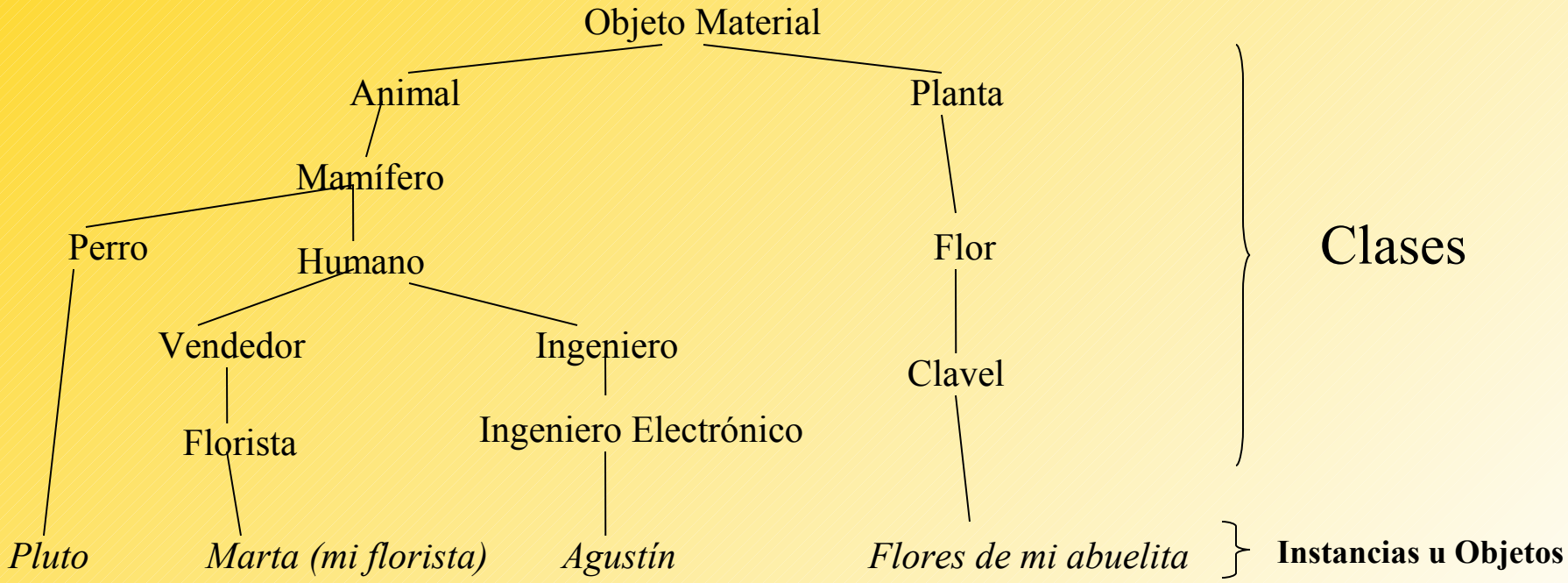
- Nosotros siempre tenemos una idea de las cosas más allá de ellas mismas. La vendedora es una instancia de una categoría o clase (por ejemplo Florista).
- Todos los objetos son instancias de alguna clase.
- Los objetos son entes que tienen nombre, comportamiento y estado.

Jerarquía de clases y herencia

- El hecho que **el conocimiento de una categoría más general es también aplicable a una categoría específica** se conoce como **Herencia**.
- Decimos que la clase Florista hereda los atributos de la clase Vendedor, y ésta hereda de la clase Humano, y ésta hereda de la clase Mamífero .... Se establece así una **Jerarquía de clases.**

Jerarquías de Clases

- Las clases pueden ser organizadas en estructuras de herencia jerárquicas.
- Una clase hijo (O subclase) hereda atributos de la clase padre. Una clase abstracta no posee instancias directas y es sólo usada para crear subclases. Por ejemplo Mamífero



Objetos-Mensajes, Herencia, y Polimorfismo

- Paso de mensajes: En OOP las acciones son iniciadas por un requerimiento a un objeto específico. (analogía: invocación de procedimiento es a procedimiento como mensaje es a método)

- Lo previo es nada más que un cambio de énfasis. ¿Qué es más natural: llamar a la rutina push con stack y dato como parámetros o pedirle a un stack hacer un push de un dato?
- Implícito está la idea que la interpretación del mensaje puede variar con diferentes objetos (polimorfismo). Los nombres no necesitan ser únicos. Se pueden usar formas más simples que conducen a programas más leíbles y entendibles.
- La Herencia permite a diferentes tipos de datos compartir el mismo código (=> menor tamaño de código y mayor funcionalidad).
- Polimorfismo: Hay varias formas de él. La idea básica es usar el mismo nombre o mensaje para referirse a cosas muy similares. ¿Por qué debería darle un nombre distinto a la función push cuando insertamos un real -float- o insertamos un carácter -char?