Pensando Orientado al Objeto

Agustín J. González

ELO-326: Seminario de Computadores I

2do. Sem. 2001

Introducción

- En esta clase exploraremos y explicaremos las bases de la programación orientada al objeto (OOP) y se ilustrarán las siguientes dos ideas:
  - OOP es una idea revolucionaria, totalmente distinta de otras en programación
  - OOP es un paso en la evolución de previos abstracciones de programación

¿Por qué es popular OOP?

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

- La programación de un computador aún es una de las más difíciles tareas 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 que significa hacer cómputos, sobre como 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 solo para los lenguajes naturales (español, inglés, mapuche...) sino también para los lenguajes artificiales como los de programación (C, Pascal, C++, Java...)
- Ejemplos:
  - Esquimales y la nieve: en su lengua existen gran número de términos para referirse a la nieve dependiendo de su contextura. No basta con conocer las palabras, sino que hay saber reconocer los tipos de nieve para usar el lenguaje correctamente. Análogamente, no basta conocer un lenguaje OO, sino que hay que saberlo usarlo "Programas FORTRAN pueden ser escritos en cualquier lenguaje"
  - Un Ejemplo en Lenguajes de Computación: Si se desea saber si un patrón de largo M (fijo y pequeño ~ 10) se repite en una secuencia. Una estrategia es tomar secuencialmente grupos de M símbolos y compararlos otros M símbolos de la secuencia para todas las posibilidades. Sería la forma de un programador Pascal o C.

    Otra forma es hacer una matriz de ancho M y el largo de la secuencia, luego ordenar las filas de la matriz y comparar filas iguales. Sería la forma de un programador APL, donde la operación de ordenar es trivial y los loop son difíciles.

Lenguaje y Pensamiento (Cont.)

- Hipótesis de Whorf: Puede ser posible para un individuo trabajando en un lenguaje imaginar pensamientos o ideas que no pueden ser trasladadas o entendidas por individuos trabajando en otro contexto lengüistico.
  ¿Entendemos los problemas de origen étnico y religioso en algunos países del planeta?
- Conjetura de Church: Cualquier computación para la cual existe un procedimiento efectivo puede ser realizada por una maquina 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.
- Entonces en qué quedamos, estas dos ideas parecen contradictorias.
- Técnicas de orientación al objeto no proveen ninguna capacidad computacional nueva que permita resolver problemas no solubles por otros medios. Pero estas técnicas hacen estas soluciones más fáciles y naturales y favorece la administración de grandes proyectos.

Una nueva forma de ver el mundo

- Supongamos que deseo enviar flores a mi abuelita. Una forma es ir a la florería y pedirselo 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 este método.
- Las acciones son iniciadas en OOP por la transmisión de un mensaje a un agente (un ***objeto***) responsable por la acción.
- Dos ideas: Ocultar de información (dejar saber sólo lo indispensable) y sacarse la idea de tener que se debe escribir todo y no usar servicios de otros (delegar).
- Dos importantes diferencias entre 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 o procedimiento) y el fragmento de código (método) usado para responder es determinada a tiempo de ejecución.

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 tu puedes hacer a tu estructura de datos, sino pregunta lo que tu estructura de datos puede hacer por ti.

Clases e Instancias

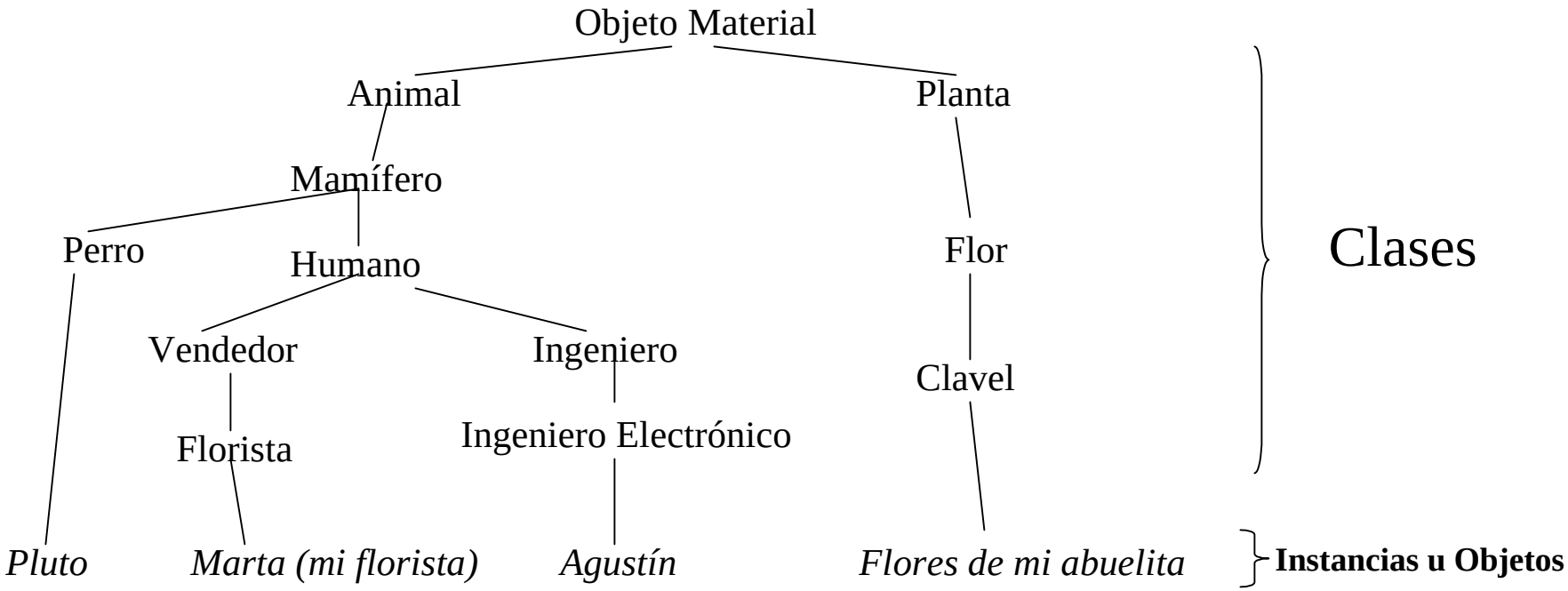
- Nosotros siempre tenemos una idea de las cosas mas allá de ellas mismas. La vendedora es una ***instancia*** de una categoría o ***clase*** (por ejemplo Florista).
- Todos los objetos son ***instancias*** de alguna ***clase***.

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 para la cual no hay instancias directas, es sólo usada para crear subclases. Por ejemplo Mamífero



Ligazón de métodos, sobremontura (override), y excepciones

- Hay situaciones en que un método “general” en una categoría es violado por laguna categoría más específica. Por ejemplo hay un mamífero que pone huevos.
- En este caso se hace una excepción a al regla general redefiniendo el método en la categoría específica.
- La búsqueda de un método parte por las definiciones específicas (en las subclases). Sólo si no se le encuentra se busca en las categorías más generales (clases padres o superclases). Cuando una categoría posee un método con el mismo nombre en una categoría superior, se dice que el método sobremonta el comportamiento heredado.

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, dispersas en la memoria, saca valores desde varias localizaciones, los transforma, y pone resultados en otras localizaciones.
- Este modelo no ayuda mucho para entender como resolver problemas reales. No es la forma de pensar y ver las cosas.
- La visión de la OOP que crea un “universo” 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.
- No todo puede ser delegar, esto resultaría en un loop infinito. Los objetos deben hacer algún trabajo antes de pasar los requerimientos a otro agente.
- En un lenguaje mixto orientado-al-objeto/imperativo, como C++ y Java, esto es hecho por métodos escritos en el lenguaje base (no orientado al objeto).

Tratando con la complejidad

- Los problemas más complejos son comúnmente abordados por un equipo de programadores.
- La tarea que toma a un programador dos meses, no puede ser realizada por dos programadores trabajando un mes.
- La razón es la complejidad. La interconexión entre componentes es tradicionalmente complicada y por ello gran cantidad de información debe ser intercambiada entre los integrantes de un equipo.
- Las comunicaciones entre programadores puede llegar a dominar el tiempo dedicado y la incorporación de más gente **extiende** el proyecto en lugar de acortarlo.
- El principal mecanismo para controlar la complejidad es la abstracción. Ésta es la habilidad de encapsular y aislar información de diseño.
- Los procedimientos son un camino para ocultar información. Pero no resuelven todo, en particular debido a que múltiples programadores pueden coincidir en los mismos nombres.

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 (“protocolo”), también se está dando acceso a sus datos comunes (datatop) , al dar accesso los nombres quedan “tomados”.
- Para resolver los problemas planteados antes se han propuesto otros lenguajes, como Modula. La idea es (dos principios de David Parnas):
  1.- Proveer al usuario (otro programador, por ejemplo) toda la información necesaria para usar correctamente un módulo, y NADA MAS.
  2.- Se debe proveer al implementador con toda la información que él necesita para completar el módulo, y NADA MAS.

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 poner un valor dentro de él?
- Implícito está la idea que la interpretación del mensaje puede variar con diferentes objetos. Los nombres no necesitan ser únicos. Se pueden usar formas más simples que conducen a programas más leíbles y entendibles.
- Herencia le permite a diferentes tipos de datos compartir el mismo código (=> menor tamaño de código y mayor funcionalidad).
- Polimorfismo: permite personalizar el código compartido para satisfacer excepciones a la regla (un mamífero que ponga huevos). Toda figura (cuadrado, círculo) se puede dibujar, pero cada una de ellas debe definir como hacerlo. En otras palabras, la capacidad de dibujarse es común para todas las figuras, pero cada una de ellas debe definirlo.

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 reescrita 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 u buscar los elementos.
- El uso de técnicas de programación orientada al objeto debería propender a generar gran número de componentes de software reusable.