Pauta Lectura 2 CC51H: Programación Orientada a Objetos

Profesora: Nancy Hitschfeld Kahler.
Auxiliar: Alcides Quispe Sanca.
Ayudante: Diego Díaz Espinoza

July 14, 2010

1. Pregunta uno.

   En líneas generales, no existe mucha información ni guías prácticas que ayuden a los programadores y diseñadores computacionales en temas fundamentales de OOP, tales como son: tipos (*types*) y herencia (*inheritance*). Cómo, porqué y cuándo son usados estos terminos en OOP, son informados en términos de lenguajes particulares y no en líneas generales.

   Algunas de las dificultades más comunes son:

   (a) Centrarse en datos más que en procesos: Existe la errada idea de pensar que sólo los datos pueden ser procesados como Objetos en los modelos. Los procesos también pueden ser considerados como Objetos en OOP y en muchas aplicaciones es la única forma de representarlos correctamente. Se acostumbra a dejar las operaciones sobre los datos, como métodos propios de los Objetos. Sin embargo, los procesos por sí mismos pueden ser Objetos.

   Ejemplo: En una aplicación de diseño gráfico, se tienen diferentes Objetos para representar gráficos. Una vez que los gráficos están dispuestos en el área de trabajo, se desea aplicar una transformación a cada uno de los objetos. En este caso las transformaciones deben ser considerados como Objetos, por ejemplo abstractos con una interfaz pública y un método mediante el cual se aplique dicha transformación sobre un Objeto gráfico.

(b) Objetos sin entidades equivalentes en el modelo: Comunmente se traspasan del modelo del problema sólo aquellas entidades que poseían una existencia clara en el mismo. Sin embargo, en OOP se pueden –e incluso a veces se deben– crear nuevos Objetos, que no tienen un equivalente en el modelo del problema o ámbito de aplicación.

Ejemplo:

En un programa de mensajería, existe Objetos que se comunican con Objetos iguales pero de forma remota. Sin embargo, estos Objetos deben comunicarse con el sistema operativo residente para hacer uso del hardware. Los Objetos que se comunican con el hardware, no existen en el modelo teórico, sin embargo son necesarios al momento de implementar dichos sistemas.

(c) Uso adecuado de tipos: El uso de tipos en OOP suele estar limitado por los tipos nativos del ambiente de programación. Sin embargo, en OOP se debe considerar la declaración de nuevos tipos mediante Objetos, versus el uso de operaciones sobre los tipos nativos proveídos por el lenguaje, para representar estos nuevos tipos. Cada una de estas opciones ofrece ventajas y desventajas.

Ejemplo: Un procesador de texto debe considerar Objetos que representen secuencias de cadenas de texto. Se pueden implementar mediante arreglos de strings, arreglos de char o arreglos de bytes. Al mismo tiempo, se puede pensar en crear un Objeto que mantenga las mejores características de cada uno de estas formas, pero sin exponer su implementación. Este nuevo Objeto (ahora un tipo para esta aplicación) podría ser ineficiente en el uso de recursos del hardware respecto a los tipos nativos, pero resultará útil para el diseño e implementación de la aplicación final.

2. Pregunta dos: el criterio de aplicabilidad propone: Mientras más especializado es un comportamiento, más propenso es de ser considerado como un tipo separado.

   Ejemplo: Todos los algoritmos de ordenamiento aplican sobre Objetos mediante una función o método de distancia entre los mismos. Al ser métodos especializados (aplican SÓLO sobre Objetos sobre los cuales se puede definir una función de distancia o de comparación) entonces resulta conveniente crear un tipo especializado para dichos objetos. Así

también existen métodos (no sólo de ordenamiento) que sólo se aplican a tipos numerales (integer, float, double, etc), como por ejemplo todas las funciones matemáticas. En este caso conviene crear un tipo Numeric o Numeral, que englobe todos los Objetos que pueden actuar como tales.

3. Pregunta tres:

   La metodología de diseño basada en *Subtyping*, consiste en desarrollar el software desde un modelo usando tipos complejos que se van descomponiendo en sub-tipos, mediante herencia, especialización, combinación y otros. La principal ventaja es la modularización (que también está presente en otras técnicas de desarrollo); y se basa principalmente en descomponer un problema en partes para luego desarrollar por separado. Sin embargo, a diferencia de otras técnicas, Subtyping tiene pocas dificultades en la integración de las diferentes partes. Se destacan en este esquema:

   (a) Espcialización
   (b) Implementación
   (c) Combinación

4. Pregunta cuatro

   En lineas generales depende de la aplicación misma, y en concreto del o los usuarios finales y los casos de uso del sistema. En algunas ocasiones resulta cómodo modelar con herencia mientras que en otras no.

5. Pregunta cinco

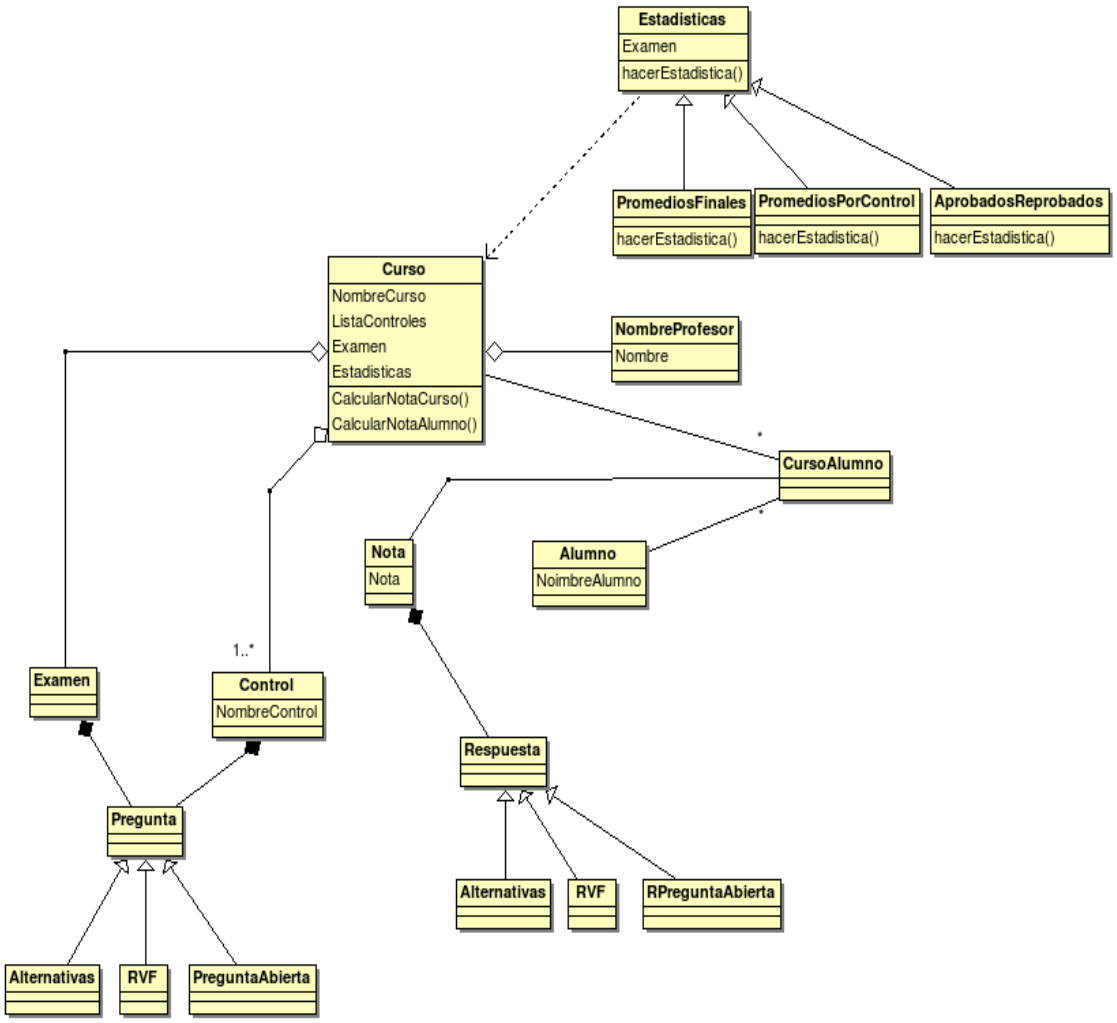

Figure 1: Diagrama de especialización. Conjunto hereda todas las propiedades de una Bolsa. Todo Conjunto es una Bolsa.