**Modelos Finales - Introducción a Objetos**

///////////////////////////////////////////////////////////////////////////////////////////////////////

FINAL 1

1 - Definir variable de instancia. Ejemplificar.

- Definir variable de clase. Ejemplificar.

- Tipos de métodos, desarrollar.

2 - Modelado de patrones. Modele UML del patrón Observer y dé un ejemplo (que no esté en las teorías y prácticas) de su posible uso, explicando cómo hará el patrón para resolver el problema.

3 - Colecciones - Características, comportamiento, tipos de colecciones. Explique tres operaciones que tengan en común todas las colecciones.

4 - Proponga un enunciado creativo que contenga 6 clases como mínimo y por cada clase 4 atributos y 4 responsabilidades o métodos. Tener en cuenta el uso de herencia simple, composición y también redefinir un método de una super clase.

///////////////////////////////////////////////////////////////////////////////////////////////////////

FINAL 2

Fue un final accesible. Tomó lo siguiente, parafraseando:

- Pseudovariables. Constantes y no constantes.

- Definir herencia simple y múltiple.

- Definir patrones de diseño. Enumerar 3. Dar un ejemplo de uso de cada uno. Y dar 5 características que deben cumplir los patrones de diseño.

- No tomaron código (increíble). Había un último ejercicio que decía: Desarrolle un enunciado de ejercicio para resolver modelando con UML. Debe tener 6 clases mínimo, con 4 métodos únicos y 4 atributos únicos cada uno. O sea, tenías que armar un enunciadito sobre el ejercicio, y después implementarlo en UML.

Se tomaron preguntas de finales que ya se venían tomando. Y, a mi entender, se simplificó un poco la complejidad.

Quizás alguna más que me esté olvidando. El ejercicio del UML / Enunciado, obviamente era el que más puntos valía.

///////////////////////////////////////////////////////////////////////////////////////////////////////

FINAL 3

Teoría:

- ¿Qué es un programa orientado a objetos?

- ¿Qué elementos hay en un programa orientado a objetos?

- Características de la programación orientada a objetos.

- Herencia. ¿Qué relación define la herencia entre objetos? ¿Qué tipos de herencia existen? Ventajas y desventajas.

- Pseudovariables constantes y no constantes, ejemplos.

- Definir patrones. Dar 5 características que deben cumplir. Desarrollar 3 patrones.

Práctica:

Una empresa tiene que liquidar sueldos. La empresa tiene un máximo de vendedores en su plantel. Los vendedores pueden ser fijos o a comisión. Los fijos cobran un básico + $100 por cada año de antigüedad. Los de comisión cobran un básico + un porcentaje del precio de cada artículo que vendieron. Cada vendedor conoce los artículos que vendió. Los artículos pueden ser nacionales o importados. Los importados tienen un recargo del 21% por impuestos.

Implementar métodos para: agregar un vendedor si no se sobrepasó el máximo de vendedores en el plantel, devolver el total de artículos vendidos por todos los vendedores, obtener el total de sueldos a pagar.

///////////////////////////////////////////////////////////////////////////////////////////////////////

FINAL 4

1) ¿Que es un programa orientado a objetos? Definir todos los componentes

¿Cuáles son los principios básicos de la POO?

2) Definición de patrones. Dar 5 características...Escribir 3 ejemplos de patrones, y que funcionalidad tiene cada uno.

Tomar uno de los patrones y realizar un ejemplo en UML.

3) Definir self/super. Ejemplo.

Definir variable de instancia. Ejemplo.

Definir variable de clase. Ejemplo.

4) Realizar un diagrama UML con 8 clases como mínimo. Cada clase tiene q tener 4 características únicas, y 4 comportamientos únicos. O sea 4 métodos únicos.

El ejercicio no puede ser parecido a los de la teoría o práctica.

RESPUESTAS:

***- ¿Qué es un programa orientado a objetos?***

La programación Orientada a Objetos es un método de implementación en el cual los programas están organizados como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas vía relaciones de herencia.

***- ¿Qué elementos hay en un programa orientado a objetos?***

En el paradigma orientado a objetos, el marco

conceptual es el modelo de objetos. Hay cuatro

elementos principales en este modelo:

**• Abstracción :** Una abstracción es una descripción simplificada de un sistema que enfatiza algunos de sus detalles o propiedades mientras suprime otros. Una buena abstracción es la que enfatiza detalles que son significantes al lector, y suprime los que no lo son.

La abstracción enfoca las características esenciales de algún objeto, relativa a la perspectiva del observador

Una abstracción se enfoca sobre una vista externa del objeto, y sirve para separar su comportamiento esencial de su implementación.

**• Encapsulación:** El encapsulamiento (ocultamiento de la información) previene que los clientes vean la parte interna, donde el comportamiento de la abstracción es implementada.

Según Booch: "El encapsulamiento es el proceso de ocultar todos los detalles de un objeto que no contribuyen a sus características esenciales.

**• Modularidad:** Si bien el acto de particionar un programa en componentes individuales reduce en algún grado la complejidad, una razón más poderosa para realizarlo es que ésto crea un número de contornos bien documentados dentro del programa.

**• Jerarquía:** La abstracción es sin duda algo bueno, pero salvo en aplicaciones triviales, encontramos que hay mas abstracciones diferentes que las que podemos comprender a un dado momento. El encapsulamiento nos ayuda a manejar esta complejidad ocultando la parte interna de estas abstracciones. La modularidad también ayuda, dándonos una manera de agrupar lógicamente abstracciones relacionadas. Esto no es suficiente. Un conjunto de abstracciones frecuentemente forman una jerarquía, e individualizándolas en nuestro diseño, simplificaremos el entendimiento del problema.

La jerarquía es un ordenamiento de abstracciones.

Las dos jerarquías más importantes son: estructura de clases (jerarquía tipo de) y la estructura de objetos (jerarquía parte de).

***- Características de la programación orientada a objetos.***

**Abstracción :** Una abstracción es una descripción simplificada de un sistema que enfatiza algunos de sus detalles o propiedades mientras suprime otros. Una buena abstracción es la que enfatiza detalles que son significantes al lector, y suprime los que no lo son.

La abstracción enfoca las características esenciales de algún objeto, relativa a la perspectiva del observador

Una abstracción se enfoca sobre una vista externa del objeto, y sirve para separar su comportamiento esencial de su implementación.

**Encapsulación:** El encapsulamiento (ocultamiento de la información) previene que los clientes vean la parte interna, donde el comportamiento de la abstracción es implementada.

Según Booch: "El encapsulamiento es el proceso de ocultar todos los detalles de un objeto que no contribuyen a sus características esenciales.

**Modularidad:** Si bien el acto de particionar un programa en componentes individuales reduce en algún grado la complejidad, una razón más poderosa para realizarlo es que ésto crea un número de contornos bien documentados dentro del programa.

**Principio de ocultación**: cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos. (SACADO DE WIKIPEDIA)

**Polimorfismo:** Es aquella cualidad que poseen los objetos para responder de distinto modo ante el mismo mensaje. Ejemplo: las clases hombre, vaca y perro, si a todos les damos la orden -enviamos el mensajeCome, cada uno de ellos sabe cómo hacerlo y realizará este comportamiento a su modo.

**Herencia:** la herencia es el mecanismo por el cual una clase X puede heredar propiedades de una clase Y ( X hereda de Y) de modo que los objetos de la clase X tengan acceso a los atributos y operaciones de la clase, sin necesidad de redefinirlos.

**Recolección de basura**: la recolección de basura o *garbage collector* es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando.  
En la mayoría de los lenguajes híbridos que se extendieron para soportar el Paradigma de Programación Orientada a Objetos como C++ u Object Pascal, esta característica no existe y la memoria debe desasignarse expresamente. (SACADO DE WIKIPEDIA)

***- Pseudovariables constantes y no constantes, ejemplos.***

**“Pseudo-variables no-constantes”**

**• Self/ this** . Hace referencia al receptor del mensaje

**• super.** “Referencia al receptor del mensaje, pero indica que no debe usarse la clase del receptor en la búsqueda del método a evaluar. Se usa, sobre todo, cuando se especializa un método en una subclase y se quiere invocar el método de la superclase.” Al contrario que this, super permite hacer referencia a miembros de la clase padre (o a los ancestros anteriores, que no hayan sido ocultados por la clase padre) que se hayan redefinido en la clase hija. Si un método de una clase hija redefine un miembro –ya sea variable o método– de su clase padre, es posible hacer referencia al miembro redefinido anteponiendo super.

**“Pseudo-variables constantes”**

**• Nil / null.** “Referencia a un objeto usado cuando hay que representar el concepto de „nada‟ o de „vacío‟. Las variables que no se asignaron nunca, referencian a nil”

**• true.** “Referencia a un objeto que representa el verdadero lógico.”

**• false.** “Referencia a un objeto que representa el falso lógico.

***- Definir herencia simple y múltiple.***

Simple: Una clase sólo es subclase de una superclase

Múltiple: una subclase admite más de una superclase.

***- Definir self/super. Ejemplo.***

**“Pseudo-variables no-constantes”**

**• Self/ this** . Hace referencia al receptor del mensaje

**• super.** “Referencia al receptor del mensaje, pero indica que no debe usarse la clase del receptor en la búsqueda del método a evaluar. Se usa, sobre todo, cuando se especializa un método en una subclase y se quiere invocar el método de la superclase.” Al contrario que this, super permite hacer referencia a miembros de la clase padre (o a los ancestros anteriores, que no hayan sido ocultados por la clase padre) que se hayan redefinido en la clase hija. Si un método de una clase hija redefine un miembro –ya sea variable o método– de su clase padre, es posible hacer referencia al miembro redefinido anteponiendo super.

***- Definir variable de instancia. Ejemplo.***

Las variables de instancia representan el estado del objeto y perduran durante toda la vida de éste. Dos objetos diferentes, aunque pertenezcan a la misma clase, pueden tener valores diferentes en sus variables de instancia.

***- Definir variable de clase. Ejemplo.***

Las variables de clase son compartidas por las instancias de una clase y sus subclases, manteniendo el mismo valor para todas las instancias. Estas variables se declaran en la definición de la clase. Ej: Mayoria de edad, cant limite de tarjeta de credito.

***- Herencia. ¿Qué relación define la herencia entre objetos? ¿Qué tipos de herencia existen? Ventajas y desventajas***

**Problemas de Herencia Múltiple**

• Herencia Repetida: A hereda de B y C, que a su vez heredan de D, por lo tanto, la clase A hereda 2 veces de D.

• Conflicto de Nombres: Si A hereda simultáneamente de B y C aparece un conflicto si usan el mismo nombre para algún atributo o método

La principal **ventaja de la Herencia** es que permite reutilizar código

**Los tipos de Herencia que existen son**:

Simple: Una clase sólo es subclase de una superclase

Múltiple: una subclase admite más de una superclase.

**La relación que define la herencia** entre objetos son las características comunes y propias de cada objeto

***- Definir patrones de diseño. Enumerar 3. Dar un ejemplo de uso de cada uno. Y dar 5 características que deben cumplir los patrones de diseño.***

* Observer
* Composite
* Singleton

Caracteristicas de patrones de diseño:

– Solucionar un problema reiterativo

– Ser un concepto probado

– La solución no es obvia

– Describe participantes y relaciones entre ellos

– Los patrones indican repetición, si algo no se repite, no es posible que sea un patrón

***- Tipos de métodos, desarrollar.***

***- ¿Cuáles son los principios básicos de la POO?***