**Instituto Superior de Formación Técnica Nº 151 logo151-trans.gif  
Carrera: Analista de Sistemas  
1 Año. Algoritmos y Estructuras de Datos I**

|  |  |
| --- | --- |
| **Trabajo Práctico Nº 10** | **Unidad 10** |
| **Modalidad:** Semi-Presencial | **Estratégica Didáctica:** Trabajo Individual |
| **Metodología de Desarrollo:** acordar | **Metodología de Corrección:** acordar docente |
| **Carácter de Trabajo:** Obligatorio – con Nota | **Fecha Entrega:** A confirmar por el Docente. |

**Paradigma Orientado a Objetos – Programación Orientada a Objetos**

**Marco Teórico:**

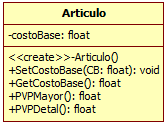
1. Comentar la Siguiente Expresión: “***El origen de la revolución informática ocurrió dentro de una máquina. Por tanto, el origen de nuestros lenguajes de programación tiende a parecerse a esa máquina.”***
2. Comentar la Siguiente Expresión: ***La programación orientada a objetos es parte de este movimiento hacia un uso del ordenador como medio de expresión***
3. ¿Qué se entiende por Clase?, dar un ejemplo?
4. ¿Qué se entiende por Objeto? ¿qué relación tiene con la Clase
5. ¿Qué diferencia Plantea este paradigma, respecto al Imperativo?
6. ¿Qué es la Abstracción? ¿Qué representa? ¿Qué aporta?
7. Relacionar la Abstracción con Clases y Objetos.
8. Dar ejemplos de Lenguajes POO. Explicar las Ideas de Alan Kay.
9. ¿Qué entiende por Interfaz?, ¿Qué función Cumple?
10. ¿Cómo podés separar la Interfaz de la Implementación?
11. ¿Por qué decimos que Cada objeto tiene una interfaz?
12. ¿Cuál es el concepto de Encapsulamiento?
13. ¿Qué entiendes por La implementación oculta? Dar ejemplo
14. ¿Relacionar lo Anterior con el Ámbito de Visibilidad, que representa?
15. Explicar los Alcances, Private, Public & Protected.
16. ¿Qué se entiende Por Reutilizar la implementación?
17. ¿Qué es la Herencia? ¿Por qué decimos que tenemos reutilización de interfaces’?
18. Explicar el Principio de Sustitución - Relaciones es-un vs. es-como-un
19. ¿Por qué comentamos que los Objetos son intercambiables gracias al Polimorfismo?
20. Dar ejemplos de Ciclo de Vida de Objetos.
21. Relacionar Creación y destrucción de objetos con ciclo de Vida.

**Marco Práctico:**

**Tener en Cuenta:**  
. Modularizar el Programa.  
. Proteger contra Inclusiones Múltiples.  
. Aplicar Espacios de Nombres.  
. Aplicar apropiadamente los conceptos de abstracción, encapsulación y ocultamiento de información.  
. Realiza una apropiada distribución de responsabilidades entre las entidades del espacio de la solución.   
. Desarrolla para reusar.  
. Reusar apropiadamente las entidades desarrolladas en el espacio curricular.  
. Demuestra un uso apropiado de la sintaxis y semántica del lenguaje de programación C++.

**Desarrollar un Programa** que:

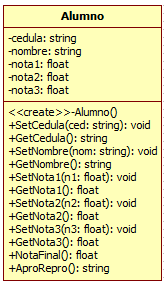
**1.** *Se conoce de un****artículo****su costo base y se sabe que se ofrecen 2 precios de venta: al****detal****(30% de incremento del costo base) y al****mayor****(15% de incremento del costo base). Se desea conocer los distintos PVP del artículo, al detal y al mayor.*

***Diagrama de clases:***

**2.** Planteamiento ejercicio 2:

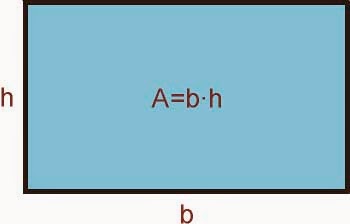
Se conoce de un alumno: cédula, nombre y tres notas parciales (nota1, nota2, nota3). El programa debe imprimir: cédula, nombre, nota final e indique con un mensaje si el alumno aprobó (nota final >= 48) o no aprobó (nota final < 48) la asignatura.

Diagrama de clases

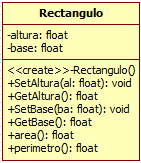
: 

**3.** Ejercicio Número 3 **Planteamiento:**

Dado la **altura** y la **base** de un **rectángulo**, calcule su área (base \* altura) y perímetro (2\*base + 2\*altura).



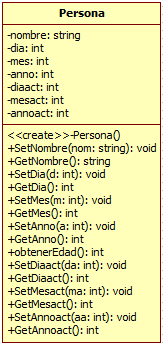
**Diagrama de clases:**



4. **Planteamiento:**

Se conoce de una **persona**el **nombre**, el **día**, **mes**y **año**de nacimiento, escriba un programa que lea la fecha de hoy, como **día**, **mes**y **año**e imprima el nombre de la persona y su edad.

**Diagrama de clases:**



Lic. Oemig José Luis.

**Marco Teórico:**

1. Comentar la Siguiente Expresión: “***El origen de la revolución informática ocurrió dentro de una máquina. Por tanto, el origen de nuestros lenguajes de programación tiende a parecerse a esa máquina.”***

Justamente la primera “capa de abstracción” del binario con que trabajan los ordenadores fue el lenguaje ensamblador (una pequeña abstracción). Y los primeros lenguajes que se popularizaron (Fortran, C, Basic) eran abstracciones del lenguaje ensamblador. De allí sus parecidos con las máquinas.

1. Comentar la Siguiente Expresión: ***La programación orientada a objetos es parte de este movimiento hacia un uso del ordenador como medio de expresión***

Justamente, uno de los puntos fuertes de este paradigma está puesto en su expresividad: si observamos el mundo que nos rodea, vemos que está lleno de objetos. Cada objeto tiene sus propias características (propiedades) y su propio comportamiento (métodos). Como todo esto está contemplado en este paradigma, nos permite un nivel de expresión mucho mayor al paradigma imperativo: hay un menor gap semántico entre el problema y la solución que se plantea a través de este paradigma.

1. ¿Qué se entiende por Clase?, dar un ejemplo?

Una clase es como un bastidor: es una suerte de molde para crear objetos.

1. ¿Qué se entiende por Objeto? ¿qué relación tiene con la Clase?

Un objeto es una estructura que combina datos y comportamientos. Es lo que se denomina una **instancia** de la clase.

1. ¿Qué diferencia Plantea este paradigma, respecto al Imperativo?

Una de las diferencias con respecto a la programación imperativa radica, se comentó en el punto 2, en una mayor expresividad del paradigma orientado a objetos, una reducción en el gap semántico existente entre el problema y solución. El proceso de **abstracción** se vuelve más natural, así como también es más fácil restringir el acceso a los datos sensibles y hacer que los cambios se efectúen sólo a través de métodos propios del objeto (**encapsulación** y **ocultamiento de información**)

1. ¿Qué es la Abstracción? ¿Qué representa? ¿Qué aporta?

La abstracción, como proceso, señala la selección de los detalles esenciales acerca de una cosa o grupo de estas, mientras que ignora aquellos detalles que son considerados no esenciales.

La abstracción, como entidad, señala un modelo, una vista o alguna otra representación simplificada de una cosa o grupo de cosas.

1. ¿Relacionar la Abstracción con Clases y Objetos?

Una clase es justamente una abstracción de un objeto. Por ejemplo: un plano de una casa es una abstracción del objeto ‘casa’. Es decir, la casa (objeto) es una instancia del plano (clase) Asimismo, el mismo plano (la misma clase) nos sirve para crear (instanciar) múltiples objetos.

1. Dar ejemplos de Lenguajes POO. Explicar las Ideas de Alan Kay.

Lenguajes que soportan el Paradigma Orientado a Objetos: C++, Object Pascal, Smalltalk, Java, Eiffel, Ruby, Python.

Alan Kay resumió la POO en 5 características básicas:

1. Todo es un objeto.
2. Un programa es un grupo de objetos enviando mensajes unos a otros para decirles qué hacer.
3. Cada objeto tiene su propia memoria constituida por otros objetos.
4. Cada objeto tiene un tipo.
5. Todos los objetos de un tipo particular pueden recibir los mismos mensajes.
6. ¿Qué entiende por Interfaz? ¿qué función Cumple?

Una interfaz es un enlace de conexión que permite a sistemas independientes (objetos) comunicarse entre sí.

1. ¿Cómo podés separar la Interfaz de la Implementación?

Los datos o funciones, o ambos, declarados a continuación de la palabra *public* constituyen la interfaz pública; los clientes pueden tener acceso directamente a estos miembros de clase. Los miembros de clase declarados después de la palabra *private* son considerados información privada y son inaccesibles para los clientes. Es decir que la parte public funciona como la interfaz de la clase.

La implementación de los métodos que figuran en la parte public puede realizarse luego, utilizando el operador de resolución de ámbito ‘::’.

1. Por qué decimos que Cada objeto tiene una interfaz

Porque, en general, cada objeto tendrá una parte privada que contiene los datos, y una parte pública donde se definen las funciones (métodos) que manejan esos datos.

1. ¿Cuál es el concepto de Encapsulamiento?

**Encapsulación:** Ocultar la ejecución de un módulo en un bloque separado con una interfaz formalmente especificada.

1. ¿Qué entiendes por La implementación oculta? Dar ejemplo

La **ocultación de información** evita que el usuario de una clase tenga que conocer todos los detalles de la implementación. Asimismo, la ocultación de información asegura al encargado de una clase que el usuario no pueda entrar directamente a ningún código privado o datos y comprometa la exactitud de la implementación.

1. Relacionar lo Anterior con el Ámbito de Visibilidad. ¿Qué representa?

El ámbito de visibilidad está provisto por la interfaz: lo que es visible “desde afuera”.

Mientras que “lo que no se ve” es lo que está encapsulado, ocultando así la implementación.

1. Explicar los Alcances, Private, Public & Protected.

Para implementar los conceptos de Encapsulación, ocultamiento de información y visibilidad, C++ provee de distintos niveles de acceso a las clases:

Public: significa que las definiciones posteriores están disponibles para cualquiera.

Private: significa que nadie puede acceder a estas definiciones (salvo el creador). La única forma de que un programador cliente pueda acceder a los datos declarados private es a través de métodos definidos en public.

Protected: actúa como private, con la excepción de que las clases derivadas tienen acceso a miembros protegidos, pero no a los prvados.

1. ¿Qué se entiende Por Reutilizar la implementación?

Una vez que una clase se ha creado y probado, debería constituir una unidad útil de código. De estar bien diseñada y tener la flexibilidad adecuada, dicha clase podrá utilizarse en muchos contextos. Asimismo, esa clase puede utilizarse para formar parte de otra clase (composición).

1. ¿Qué es la Herencia? Por qué decimos que tenemos reutilización de interfaces’

**Herencia**: Mecanismo por medio del cual una clase adquiere las propiedades —datos y operaciones— de otra clase.

**Clase base (superclase)**: Clase de la cual se hereda.

**Clase derivada (subclase)**: Clase que obtiene la herencia

1. Explicar el Principio de Sustitución - Relaciones es-un vs. es-como-un

Principio de Sustitución: Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

Cuando una subclase hereda de la clase base los métodos (sin añadir ni modificar los de la superclase), eso hace que el tipo derivado se vea exactamente como el tipo de la clase base, dado que tienen la misma interfaz. Como resultado, se puede sustituir un objeto de la clase derivada por un objeto de la clase base. (Relación es-un)

En otras ocasiones se deben añadir nuevos elementos a la interfaz de un tipo derivado. De esta manera se amplía la interfaz y se crea un tipo nuevo. El nuevo tipo todavía puede ser sustituido por el tipo base, pero la sustitución no es perfecta porque sus nuevas funciones no son accesibles desde el tipo base. Esta relación se conoce como es-como-un.

1. ¿Por qué comentamos que los Objetos son intercambiables gracias al Polimorfismo?

El ***polimorfismo*** permite que el mismo nombre de método invoque una operación en  
objetos de una clase padre y una operación diferente en objetos de una clase derivada.

El problema para el compilador, al querer tratar a un tipo derivado como sus tipos base genéricos es entonces saber qué pieza de código debe ejecutar. La ligadura dinámica de operaciones a objetos permite que objetos de muchos diferentes tipos derivados respondan a un solo nombre de función, cada uno en su propia manera (polimorfismo). Es decir, la ligadura dinámica permite determinar en tiempo de ejecución qué función se deberá llamar para un objeto particular. Para implementar esta ligadura tardía se utiliza la palabra reservada **virtual**. Esto es, se antepone la palabra **virtual** a las funciones miembro polimórficas de la superclase.

1. Dar ejemplos de Ciclo de Vida Objetos

Pueden verse ejemplos de ciclo de vida de los objetos en los ejercicios de la Parte Práctica.

En esta etapa la creación y destrucción de los objetos se ha realizado de manera estática.

1. Relacionar Creación y destrucción de objetos con ciclo de Vida

Los objetos se crean mediante un tipo de método especial llamado ***constructor*** (es una función propia de la clase sin tipo de retorno) y se destruyen mediante otro método especial llamado ***destructor***. Cuando el programador no incluye estos dos métodos, el compilador lo hace por él. Esto es lo que sucede en el caso de la creación de objetos estáticos.

Existe también la posibilidad de crear objetos de manera dinámica. En este caso el almacenamiento se hará en el *heap*, a diferencia de los objetos estáticos cuyo almacenamiento se produce en el *stack*. En el caso dinámico, la gestión de la memoria está absolutamente a cargo del programador.