**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:**

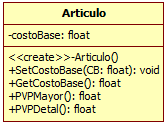
1. Comentar la Siguiente Expresión: “***El origen de la revolución informática ocurrió dentro de una máquina. Por tanto, el origen de nuestros lenguajes de programación tiende a parecerse a esa máquina.”***
2. Comentar la Siguiente Expresión: ***La programación orientada a objetos es parte de este movimiento hacia un uso del ordenador como medio de expresión***
3. ¿Qué se entiende por Clase?, dar un ejemplo?
4. Que se entiende por Objeto, qué relación tiene con la Clase
5. ¿Qué diferencia Plantea este paradigma, respecto al Imperativo?
6. Que es la Abstracción? Que 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?, que 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. Que se entiende Por Reutilizar la implementación
17. ¿Que 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 que comentamos que los Objetos son intercambiables gracias al Polimorfismo?
20. Dar ejemplos de Ciclo de Vida Objetos
21. Relacionar Creación y destrucción de objetos con ciclo de Vida=

**Marco Practico:**

**Tener en Cuenta:**  
. Modularizar el Programa.  
. Proteger contra Inclusiones Múltiples.  
. Aplicar Espacios de Nombres.  
. Aplica 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.  
.Reusa 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:

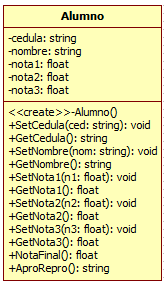
**1.** *Se conoce de un****artículo****su costo base y se sabe que se ofrecen 2 precios de venta: al****detal****(30% de incremento del costo base) y al****mayor****(15% de incremento del costo base). Se desea conocer los distintos PVP del artículo, al detal y al mayor.*

***Diagrama de clases:***

**2.** Planteamiento ejercicio 2:

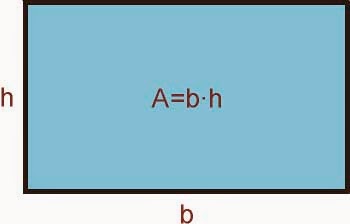
Se conoce de un alumno: cédula, nombre y tres notas parciales (nota1, nota2, nota3). El programa debe imprimir: cédula, nombre, nota final e indique con un mensaje si el alumno aprobó (nota final >= 48) o no aprobó (nota final < 48) la asignatura.

Diagrama de clases

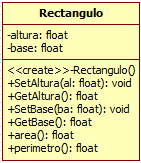
: 

**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).



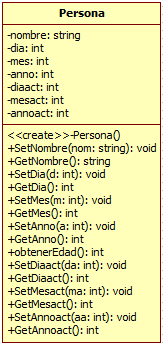
**Diagrama de clases:**



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:**



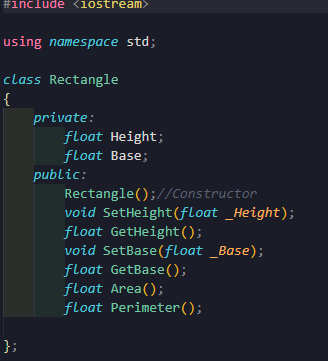
Lic. Oemig José Luis.

**Marco Teórico:**

1-Estoy de acuerdo ya que los lenguajes de programación tienden a simular la manera en que trabaja la maquina pero de una manera más entendible para los seres humanos.

2-Es cierto porque con la programación orientada a objetos ya no solo expresamos la manera en que resolvemos el problema, sino que también expresamos el problema en si, de esta forma nos expresamos para que cualquier persona puedan entender qué y cómo tratamos de resolver determinados problemas.

3-Una clase son tipos de datos abstractos,que en ella tienen características y comportamientos comunes.



4- Los objetos son entidades que combinan un estado (es decir, datos) y un comportamiento (esto es, procedimientos o métodos).Lo que tienen en común con las clases es que Los objetos con características comunes se agrupan en clases

5-La diferencia de este paradigma con respecto al imperativo es que el POO nos permite representar la realidad de una manera más cercana.

6--La abstracción es la serie de pasos para comprender y resolver un problema, este omite detallesy se centra solamente en “¿Qué hace?” dicho sistema a estudiar.Representan un objeto en general.Aporta que nos centramos únicamente en lo general y no en detalles.

7-La abstracción nos permite construir nuestras clases con sus estados y métodos,y con está podemos representar objetos comunes a esa clase ,de manera que no nos limitamos a un objeto en particular.

8-Lenguajes de Poo: Smalltalk, C++,Delphi (Object Pascal), Java y C#.Para Alan Key todo es un objeto de manera que este almacena datos pero además se le pueden hacer peticiones para que realice operaciones. Un programa es un grupo de objetos enviando mensajes a otros para decirles qué hacer. Cada objeto tiene su propia memoria constituida por otros objetos. Visto de otra manera, puede crear un nuevo tipo de objeto haciendo un paquete que contenga objetos existentes. Por consiguiente, puede hacer cosas complejas en un programa ocultando la complejidad de los objetos. Cada objeto tiene un tipo. Todos los objetos de un tipo particular pueden recibir los mismos mensajes.

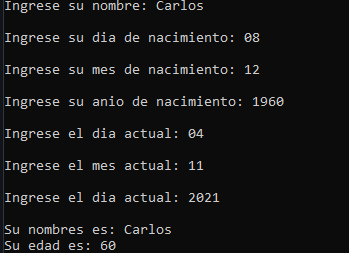
9-La interfaz es aquella función que los usuarios consumen sin saber cómo realiza la operación.Cumple la función de ocultar como se realiza la operación.

10-La interfaz nos dice “Qué hace nuestro programa” y la implementación es quien nos dice “Cómo hace la operación nuestro programa”.

11-Cada objeto tiene una interfaz ya que estos están dentro de unas class y está define sus interfaces.

12- En Programación modular, y más específicamente en programación orientada a objetos, se denomina **encapsulamiento** al ocultamiento del estado, es decir, de los datos miembro, de un objeto de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

13-La implementación oculta es aquella donde al cliente solo le exponemos la interfaz, mientras que la implementación la mantenemos oculta y en esta podemos realizar cualquier modificación sin que el cliente lo note.



14-El ámbito de visibilidad nos permite decidir que tipo de visibilidad le daremos a nuestros atributos y métodos. Con ello podremos tener una implementación oculta.

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

Private, por otro lado, significa que nadie puede acceder a estas definiciones excepto el creador del tipo, es decir, los métodos internos de la clase. Private es una pared entre el creador de la clase y el programador cliente. Si alguien intenta acceder a un miembro privado, obtendrá un error al compilar.

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

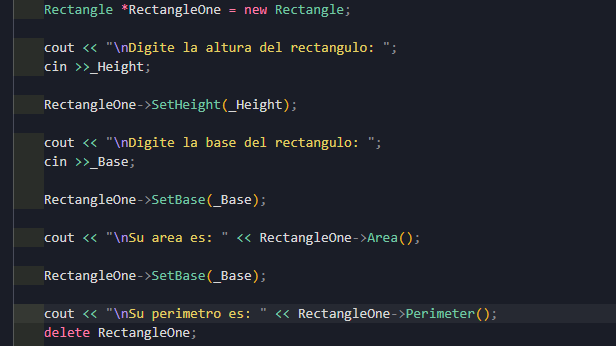
16- vez que una clase se ha creado y probado, debería constituir (idealmente) una unidad útil de código.Sin embargo, esta reutilización no es tan fácil de conseguir como muchos esperarían; producir un buen diseño requiere experiencia y conocimientos. Pero una vez que lo tiene, pide ser reutilizado. El código reutilizado es una de las mejores ventajas de los lenguajes para programación orientada a objetos. La forma más fácil de reutilizar una clase es precisamente utilizar un objeto de esa clase directamente, pero también puede colocar un objeto de esta clase dentro de una clase nueva. Podemos llamarlo «crear un objeto miembro».

17- La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código.Porque nos permite reutilizar la interfaces de la clase padre.

18- Hay cierta controversia que puede ocurrir con la herencia: ¿la herencia debería limitarse a anular sólo funciones de la clase base (y no añadir nuevos métodos que no estén en la clase base)? Esto puede significar que el tipo derivado es exactamente el mismo tipo que la clase base dado que tiene exactamente la misma interfaz. Como resultado, se puede sustituir un objeto de una clase derivada por un objeto de la clase base. Se puede pensar como una sustitución pura, y se suele llamar principio de sustitución. En cierto modo, esta es la forma ideal de tratar la herencia. A menudo nos referimos a las relaciones entre la clase base y clases derivadas en este caso como una relación es-un, porque se dice «un círculo es una figura». Un modo de probar la herencia es determinar si se puede considerar la relación es-un sobre las clases y si tiene sentido.

Hay ocasiones en las que 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; el nuevo tipo tiene la interfaz del viejo tipo, pero también contiene otras funciones, por lo que se puede decir que es exactamente el mismo.

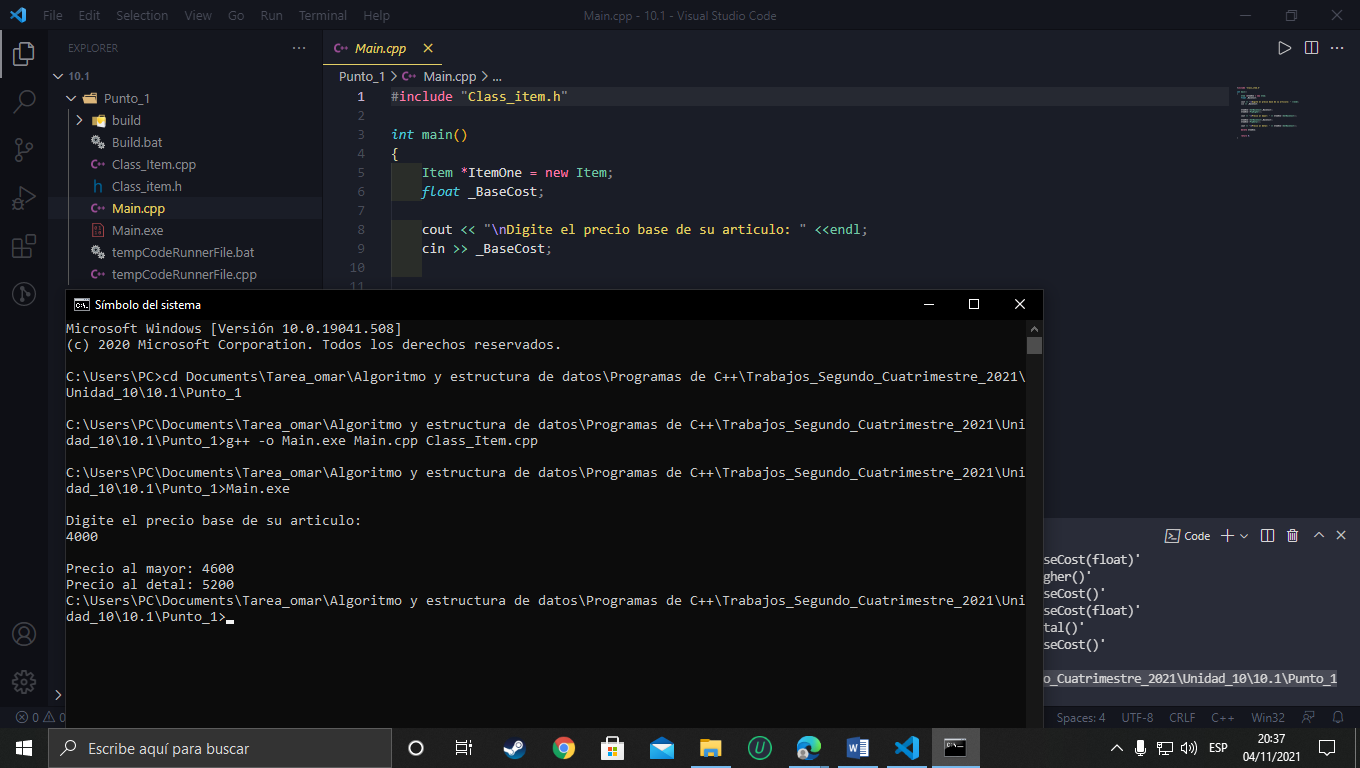
19- Cuando se manejan jerarquías de tipos, se suele tratar un objeto no como el tipo específico si no como su tipo base. Esto le permite escribir código que no depende de los tipos específicos. En el ejemplo de la figura, las funciones manipulan figuras genéricas sin preocuparse de si son círculos, cuadrados, triángulos, etc. Todas las figuras se pueden dibujar, borrar y mover, pero estas funciones simplemente envían un mensaje a un objeto figura, sin preocuparse de cómo se las arregla el objeto con cada mensaje.

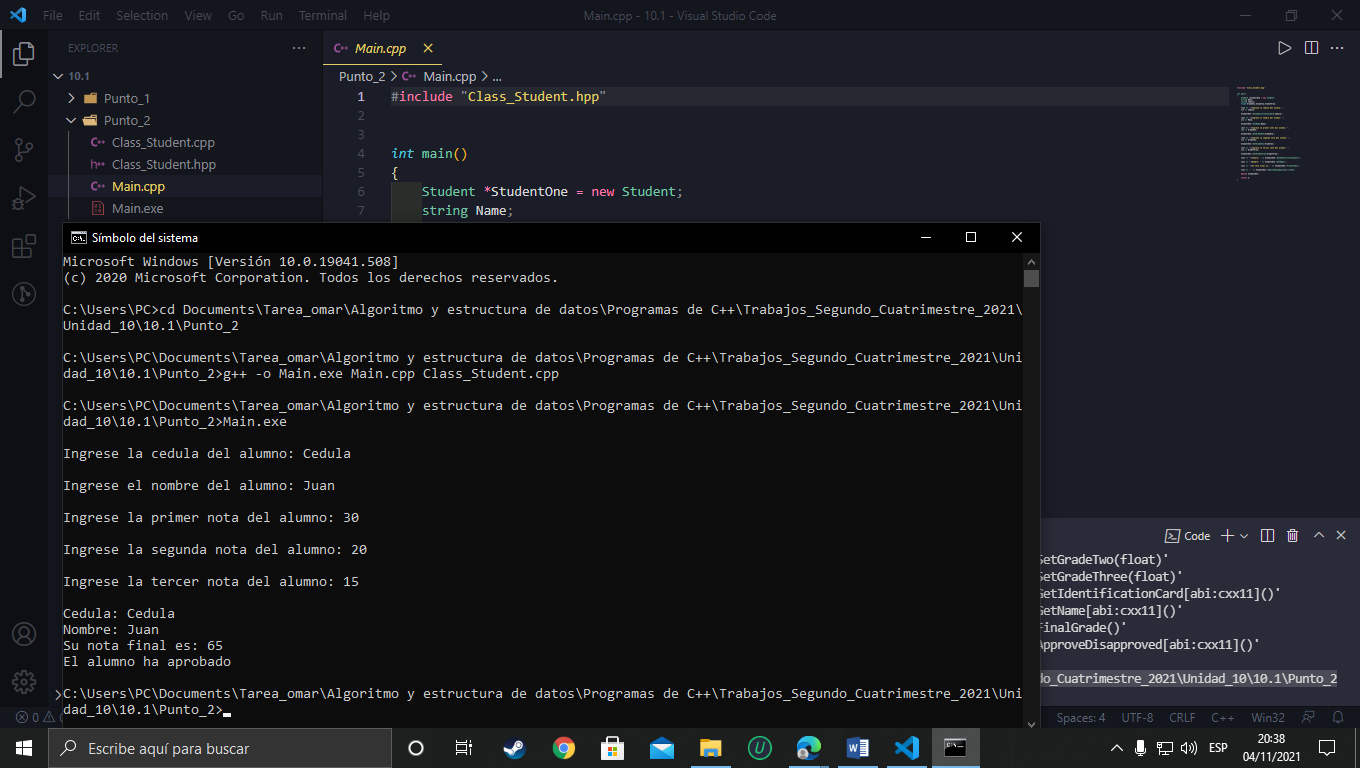
20-

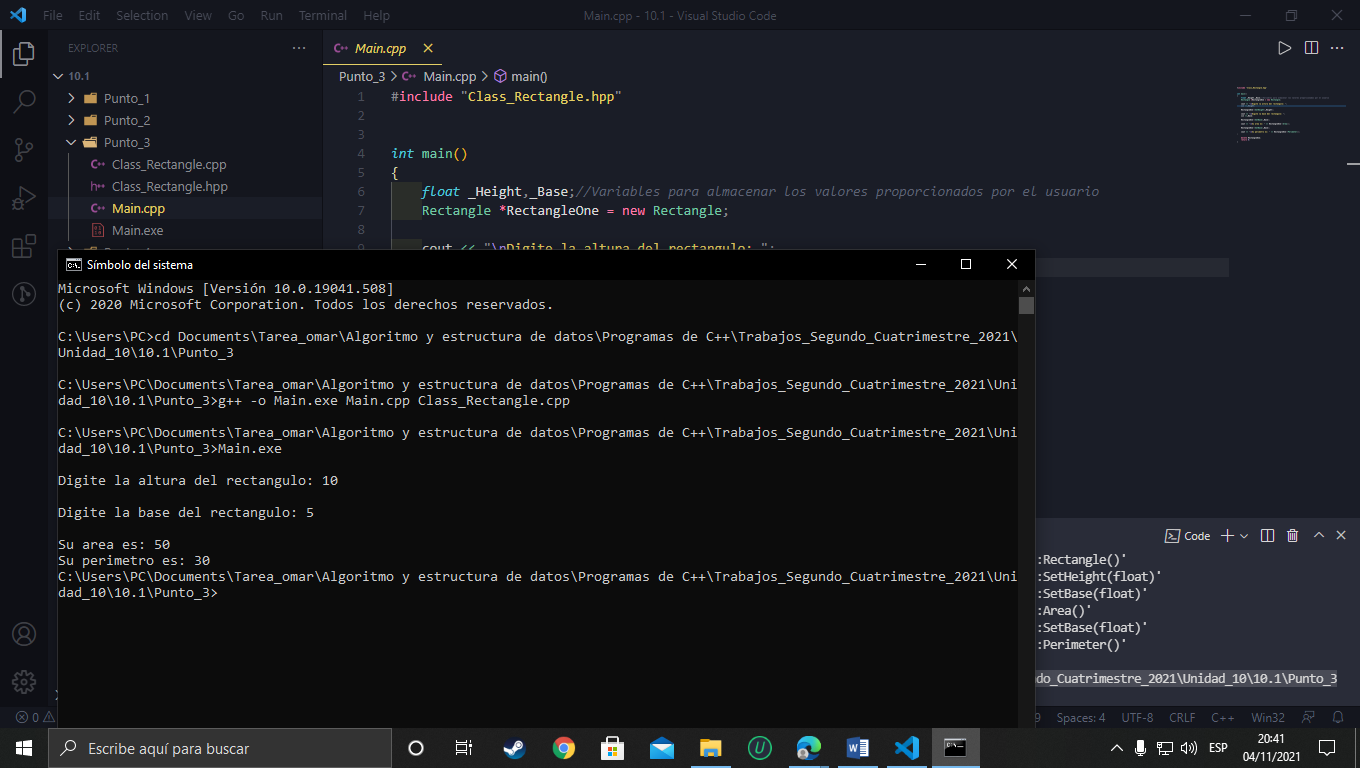
21-El ciclo de vida de nuestro objeto está entre su creación y su destrucción.

**Marco Practico:**

**1-**



2- 

3- 

4- 