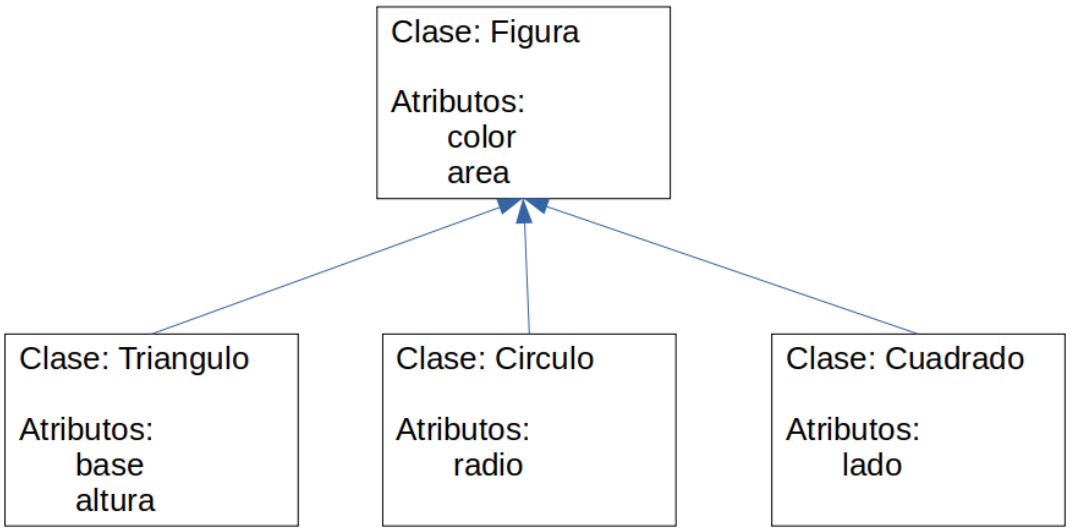
Crea un repositorio llamado Examen_Ordinaria_22_23_nombre_apellido y comparte el link con el profesor.

Recuerda que en ese repositorio deben al menos estar todos los ejercicios en diferentes ficheros y el archivo Readme.md

Duración: 2 horas.

1. (3 puntos) Dada la relación entre clases representada en la siguiente grafica



en la que la clase Figura es la clase base de la que heredan el resto de las clases.

- (0.5) Crea un objeto de cada subclase (*Triangulo*, *Circulo* y *Cuadrado*) y añádelos a una lista llamada *figuras*.
- (0.5) Realiza una función llamada **catalogar()** que reciba la lista *figuras* y recorra sus elementos mostrando el nombre de su clase y sus atributos.
- (0.5) Realiza una función llamada **mostrar_areas()** que reciba la lista *figuras* y recorra sus elementos mostrando su area con el siguiente formato: “La figura *xxxxx* tiene un area de *yyyyy* m2”, donde *xxxxx* representa el nombre de figura e *yyyyy* el área de la figura.

- (0.5) Realiza una función llamada **comparar_areas()** que reciba dos figuras y escriba los siguientes mensajes en función del área de la figura:

    1. “Las figuras xxxx y yyyyy tienen el mismo area zzzzz m2”
    2. “Las figura xxxx tiene un area mayor que la yyyyy: zzzzz > wwwww m^2”
    3. “Las figura xxxx tiene un area menor que la yyyyy: zzzzz < wwwww m^2”

- (0.5) Realiza una función llamada **comparar_areas_v2()** que implimente la misma funcionalidad que la anterior funcion, pero haciendo uso directo de los operadores matematicos >, < y = sobre cada los objetos de entrada tipo Figura, basandose en el area de cada figura. Es decir, para las clases anteriormente definidas (*Triangulo*, *Circulo* y *Cuadrado*) definir las funciones necesarias para que estos operadores se puedan usar basandose en el area de cada una de ellas. Por ejemplo, para dos figuras xxxx e yyyy el siguiente operador

    xxxxx > yyyyy

    tenga el mismo resultado que

    xxxxx.area > yyyyy.area

    Realiza los cambios necesarios en la definición de las anteriores clases.

- (0.25) Realiza una función llamada **guardar_figuras()** que guarde en un fichero una lista *figuras* de entrada. Esta función tendrá como parámetros la lista de figuras y el nombre del fichero.

- (0.25) Realiza una función llamada **cargar_figuras()** que devuelva una lista *figuras* guardadas previamente en un fichero. Esta función tendrá como parámetro el nombre del fichero y devuelva la lista de figuras.

2. (2.0 punto) Dada la lista

[18, 50, 210, 80, 145, 333, 70, 30]

- (1.0) Realiza una función que imprima el número en caso de que sea múltiplo de 10 y menor que 200. El programa se debe detener si se llega a un número mayor que 300.
- (1.0) Organizar la lista mediante el método de ordenamiento *merge sort* y ejecuta la función usando la lista ordenada. Discutir las diferencias observadas.

3. (1.0 punto) Implementar una función que calcule la potencia de un numero mediante recursión, es decir sin usar el operador **.

Por ejemplo, la operación 2^5 (o 2 x 2 x 2 x 2 x 2) se puede calcular sin el uso del operador **, mediante la relación 2^(m * 2) == 2^m * 2^m.

4. (2.0 punto) Desarrollar un algoritmo que permita comprimir mensajes contemplando los siguientes requerimientos: implementar un algoritmo que pueda crear un árbol de Huffman a partir de la siguiente tabla y desarrollar las funcionas para comprimir y descomprimir un mensaje.

Símbolo	Frecuencia
A	0.2
B	0.21
1	0.13
5	0.17
3	0.05
J	0.15
Z	0.09

5. (2.0 punto) Dada una cantidad monetaria de entrada y un numero infinitos de monedas y billetes {1, 2, 5, 10, 20, 50, 100, 500, 1000}, diseñar un algoritmo que encuentre el mínimo número de monedas y/o billetes necesarios para proporcionar cambio.

Ejemplos:

entrada: V = 70
salida: 2
Explicación: se necesita un billete de 50 y un billete de 20.

entrada: V = 121
salida: 3
Explicación: se necesita un billete de 100, un billete de 20, y una moneda de 1.