Deusto
Facultad de Ingeniería

EXAMEN PROGRAMACIÓN IV

(Final C/C++, 10 de junio de 2014)

Nombre y apellidos: _____

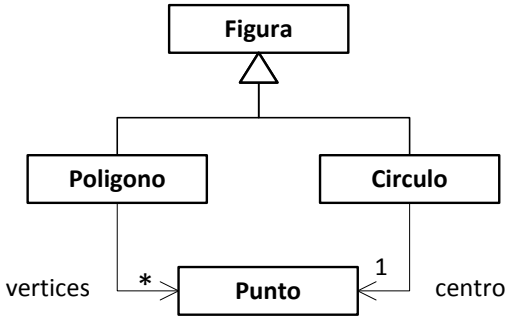
Código individual: _____

INSTRUCCIONES PARA EL EXAMEN

- La entrega del examen resuelto se realiza a través de la plataforma ALUD2. Existe una tarea para la entrega del examen llamada **Entrega examen 10/06/2014**
- El alumno debe crear un único **archivo ZIP** que contenga TODO el proyecto C++ y los ficheros requeridos, y subirlo a la plataforma. Solamente se corregirán los archivos entregados. Comprobar el contenido del ZIP.
- Es imprescindible que en el fichero correspondiente al programa principal se incluya en la parte superior, como comentario, el nombre del alumno y el **código individual** que se entregará en el examen.
- La duración del examen es de **1 hora y 30 minutos**.
- Se puede hacer uso de la referencia de la librería de C/C++ contenida en ALUD2
- Se permite el uso de cualquier material en el examen (prácticas y apuntes).

0. LOS PRELIMINARES…

Se parte de una **clase** `Punto` ya creada (con su fichero de cabecera **"Punto.h"** y de implementación **"Punto.cpp"**) y un fichero **"main.cpp"** que contiene el programa principal (que inicialmente no compila). El objetivo del examen es implementar una pequeña parte de un programa que hace uso de figuras geométricas. Tras la fase de diseño se ha llegado a la conclusión de que es necesario contemplar las clases `Punto`, `Figura`, `Circulo` y `Poligono`, en base a las siguientes relaciones:



1. CLASE FIGURA (0,25 puntos)

1.1 Figura. Crea una nueva clase `Figura` (con su correspondiente fichero de cabecera y de implementación) que sirva como base de otras figuras geométricas. Esta clase deberá ser **abstracta**.

1.2 Atributos de la clase. Esta clase contendrá simplemente una cadena de caracteres con el nombre de la figura geométrica.

Adicionalmente debe declararse un *atributo estático* privado que permita contabilizar cuántas figuras geométricas se encuentran instanciadas en un momento dado en el programa.

1.3 Métodos de la clase. Añade dentro de la clase `Figura`, al menos, los siguientes métodos:

> `constructores(…)`: necesitarás al menos uno que inicialice el nombre de la figura.
>
> `float getPerimetro()`:se encuentra definido en la clase pero no tiene implementación (método *abstracto*).
>
> `void imprimir()`: su cometido es visualizar por pantalla el nombre de la figura. Podrá ser redefinido por las clases hijas para modificar su funcionalidad.

2. CLASE CIRCULO (0,25 puntos)

2.1 Circulo. Crea una nueva clase `Circulo` (con su correspondiente fichero de cabecera y de implementación). Esta clase deberá **heredar** de la clase `Figura`.

2.2 Atributos de la clase. Esta clase, además de los atributos heredados, deberá incluir el radio del círculo y el punto que determina su centro (deberás hacer uso de la clase **`Punto`** proporcionada).

```
float radio
Punto centro
```

2.3 Métodos de la clase. Añade dentro de la clase **`Circulo`**, al menos, los siguientes métodos:

`constructores(…)`: necesitarás al menos uno que, además de los atributos heredados (nombre), reciba el radio y el punto central del círculo.

`float getPerimetro()`: debe implementar el método abstracto de la clase **`Figura`** para calcular el perímetro del círculo (2 x π x radio). Siendo π = 3.1416

`void imprimir()`: debe redefinir el método correspondiente de la clase **`Figura`**. Su cometido es visualizar por pantalla el nombre del círculo, su centro y su radio. Podría ser interesante reutilizar la funcionalidad ya implementada en la clase padre. Puedes emplear un formato similar al siguiente:

```
NOMBRE: Circulo1
Centro: (1, 2)
Radio: 3
```

3. CLASE POLIGONO (0,75 puntos)

3.1 Polígono. Crea una nueva clase **`Poligono`** (con su correspondiente fichero de cabecera y de implementación). Esta clase deberá **heredar** de la clase **`Figura`**.

3.2 Atributos de la clase. Esta clase, además de los atributos heredados, deberá incluir el número de vértices que conforman el polígono, así como todos los puntos que conforman estos vértices (deberás hacer uso de la clase **`Punto`** proporcionada).

```
int numVertices
Punto *vertices
```

Ten presente que el número de vértices no es un dato conocido a priori. Podemos estar hablando de un triángulo (3 vértices y 3 puntos), un cuadrado (4 vértices y 4 puntos) o cualquier otro polígono.

3.3 Métodos de la clase. Añade dentro de la clase **`Poligono`**, al menos, los siguientes métodos:

`constructores(…)`: necesitarás al menos uno que, además de los atributos heredados (nombre), reciba el número de vértices del polígono y un array con los puntos de sus vértices. Fíjate en el programa principal proporcionado para saber cómo se espera que sea este constructor.

`float getPerimetro()`: debe implementar el método abstracto de la clase **`Figura`** para calcular el perímetro del polígono. El perímetro se corresponde con la suma de la longitud de todos los lados del polígono. Por tanto, usando la función que calcula la distancia entre dos puntos (implementada dentro de la clase **`Punto`**), puedes ir sumando las distancias entre los distintos vértices del polígono. Todo punto (vértice) debe considerar la distancia con su adyacente y ten en cuenta que el último punto lo debe hacer con el primero.

`void imprimir()`: debe redefinir el método correspondiente de la clase **`Figura`**. Su cometido es visualizar por pantalla el nombre del polígono y cada uno de sus vértices. Podría ser interesante reutilizar la funcionalidad ya implementada en la clase padre. Puedes emplear un formato similar al siguiente:

```
NOMBRE: Triangulo
Vertice 0 = (1, 1)
Vertice 1 = (1, 2)
Vertice 2 = (2, 1)
```

4. PROGRAMA PRINCIPAL (0,75 puntos)

Partir del fichero **"main.cpp"** proporcionado y completar las secciones indicadas a continuación (éstas también se encuentran señaladas con comentarios en el propio fichero).

Verás que lo primero que hace este programa es crear dos círculos ("circulo1" y "circulo2") y un polígono de tres vértices ("triangulo"). **Atención:** NO se puede modificar el código que ya se proporciona implementado en este fichero. Sí puedes añadir nuevo código y puedes ir comentando algunas partes del mismo con el objetivo de que compile e ir probando y avanzando gradualmente en el examen.

4.1 Visualiza por pantalla el número de figuras geométricas creadas en este punto del programa. Deberás hacer uso del atributo estático que incluiste en la clase **`Figura`**.

4.2 Implementa una función que se corresponda exactamente con la siguiente cabecera:

```
void imprimirDetalleFigura(Figura &f)
```

Esta función deberá visualizar por pantalla el detalle de cualquier tipo de figura geométrica. Se entiende por detalle el nombre de la figura, su perímetro y sus características específicas (centro y radio en el caso de los círculos y los vértices en el caso de los polígonos). Naturalmente se espera que reutilices las funciones que ya tienes implementadas para ello en las clases.

Haz uso de esta función para visualizar los detalles de las tres figuras de tu programa. Se espera que la llames tres veces, una vez con cada una de ellas.

4.3 Implementa otra función que se corresponda exactamente con la siguiente cabecera:

```
float sumarPerimetros(Figura * figuras[], int numFiguras)
```

Esta función deberá devolver la suma de los perímetros de las figuras geométricas contenidas en el array que se pasa como parámetro. Naturalmente se espera que reutilices la función que ya tienes implementada para ello en las clases.

Llama a esta función desde el programa principal y visualiza la suma de los perímetros de los dos círculos y el triángulo que tienes creados. Se espera que construyas un array con ellos, acorde a lo que espera recibir la citada función.

La ejecución del código principal, una vez completo, debe imprimir el siguiente resultado:

```
NUMERO FIGURAS = 3
-----
NOMBRE: Circulo1
Centro: (1, 2)
Radio: 3
Perimetro: 18.8496
-----
NOMBRE: Circulo2
Centro: (2, 1)
Radio: 4
Perimetro: 25.1328
-----
NOMBRE: Triangulo
Vertice 0 = (1, 1)
Vertice 1 = (1, 2)
Vertice 2 = (2, 1)
Perimetro: 3.41421
-----
SUMA PERIMETROS = 47.3966
```