Tarea Integradora C++

Después de ver el curso de Programación Orientada a Objetos con C++, programar un proyecto en C++ que cumpla con los siguientes requisitos:

1. Desarrolla una clase llamada ***Figure*** que tenga algunos métodos virtuales y otros no virtuales. Separa las declaraciones de los métodos en Figure.h y las definiciones de los métodos en Figure.cpp (los métodos virtuales no se definen aquí, sino en cada una de las clases hijas):
    a. public:
        i. Método virtual `virtual double getArea()= 0;`
        ii. Método virtual `virtual double getPerimeter()= 0;`
        iii. Método virtual `virtual void printMembers()= 0;`
        iv. Método `void setColor(const std::string& color);`
            1. En Figure.cpp define este método para que actualice la variable color.
        v. Método `std::string& getColor();`
            1. En Figure.cpp define este método para que retorne la variable color.
    b. protected:
        i. Variable protegida `std::string color;`
2. Responde las siguientes preguntas.
    a. ¿Qué son los métodos virtuales en las clases?
        i. Es una función miembro de una clase base que puede redefinirse en una clase derivada, en el caso se establece que el método será implementado por las clases que hereden de Figure
    b. ¿Para qué crees que se utilicen los métodos virtuales?
        i. Se utiliza para hacer polimorfismo y es que el comportamiento real de un objeto se determine en tiempo de ejecución , para realizar como vinculación dinámica y permitir un código reutilizable y extensible.
    c. **¿Por qué crees que el método setColor reciba un `const std::string&` en lugar de simplemente un `std::string`?**
        i. La utilizamos como una optimización, para evitar copias innecesarias del mismo string, para también tener en mente que la función no modifique el valor original.
3. Investiga sobre qué se necesita para que una clase sea estática (static) y crea una clase estática llamada Logger que cumpla con lo siguiente y sepárala en Logger.h y Logger.cpp:
    a. public:
        i. `static void log(const std::string& message);`
            1. Imprime el mensaje en el formato:
            [timestamp] message
        ii. `static void logMember(const std::string& member);`
            1. Imprime el mensaje en el formato:
            –> [timestamp] message
        iii. `static void logArea(const std::string& area);`

1. Imprime el mensaje en el formato:
   <A> [timestamp] message

iv. `static void logPerimeter(const std::string& perimeter);`

1. Imprime el mensaje en el formato:
   <P> [timestamp] message

b. private

i. `static int getTimestamp();`

1. Retorna el tiempo desde que se inició el programa.

4. Responde las preguntas:
   a. **¿Qué es una clase estática?**
      i. Una clase estática, es establecida por los atributos y métodos son todos estáticos, haciendo que sus métodos y variables pertenecen a la misma clase y no puedan crear objetos de esa clase.
   b. **¿Para qué crees que se usen? Menciona 2 casos.**
      i. Pueden ser utilizados para configuraciones globales, donde cualquier parte del programa pueda acceder, como una clase que gestione la configuración
      ii. También puede ser utilizadas como helpers, ejemplo sería las funciones matemáticas

5. Desarrolla 3 clases de las figuras que tú quieras, sepáralas en su .h y su cpp.
   a. Todas las figuras tienen que ser hijas de Figure (por lo tanto definir los métodos virtuales).
      i. Para el método de printMembers(), llama la función de la clase Logger llamada `logMember()` para imprimir cada miembro (lado, radio, base, altura, etc) de la figura.
   b. Agrega como miembros privados todas las variables necesarias para calcular el área y perímetro de esas figuras. Recibe esos miembros en el constructor de la clase.
6. Investiga lo siguiente.
   a. **¿Qué son los Smart Pointers y cuál es la ventaja de utilizarlos?**
      i. Son clases que manejan automáticamente la administración de memoria para objetos dinámicos, liberando la memoria cuando ya no es necesaria, así para evitar fugas de memoria.
   b. **Menciona la diferencia entre Unique Pointer y Shared Pointer**
      i. El Unique Pointer, posee la propiedad de un objeto, únicamente teniendo un objeto a la vez, teniendo como regla no copiarse y automáticamente al ser destruido libera la memoria
      ii. Shared pointer permite que múltiples punteros compartan una misma propiedad de un objeto, el objeto es liberado hasta que el ultimo shared_ptr es destruido
   c. **¿Para qué utilizarías un Unique Pointer y para qué utilizarías un Shared Pointer?**
      i. Un unique pointer sería para garantizar la propiedad única de un objeto, recursos que no deben compartirse , por el otro lado un

shared pointer es cuando múltiples partes acceden a compartir la propiedad de un objeto , como nodos en un grafo.

7. En tu main.cpp, incluye todas las figuras, así como la clase Logger y:
    a. Crea un Unique Pointer para cada una de las tres figuras.
    b. Pregunta al usuario por los lados, base, altura, radio, etc. de cada figura y crea cada una de ellas (utilizando la función del Unique Pointer para crear variables). Para imprimir en pantalla, utiliza el método `log()` de la clase Logger.
    c. Pídele después al usuario que le asigne un color a la figura y asígnaselo.
    d. Después de haber creado las 3 figuras y de haberle asignado sus datos a cada una, imprime el área, perímetro y los miembros de cada una utilizando los métodos de la clase Logger.
8. **Subir a un repositorio de GitHub.**