Práctica 4. Implementación del sistema

1. Introducción

En las prácticas anteriores se han introducido los fundamentos de la orientación a objetos para, a continuación, aplicar estos al diseño orientado a objetos de un sistema ficticio de monitorización. Ahora empezaremos a implementar este diseño orientado a objetos en C++.

Previamente, hemos de repasar concienzudamente el diagrama de clases de la práctica anterior, porque la implementación ha de ser un reflejo/conversión de este en código.

2. Visibilidad

Lo primero a tener en cuenta a la hora de implementar es aplicar convenientemente los especificadores de acceso a los miembros de las clases. Ya hemos visto los especificadores `public` y `private`. Los especificadores de acceso determinan la visibilidad, o accesibilidad, de los atributos y operaciones de un objeto para otros objetos. Antes de empezar a implementar nuestro diseño, debemos considerar qué atributos y operaciones de nuestras clases deben ser `public` y cuáles deben ser `private`.

Ya hemos visto que, por lo general, los miembros de datos deben ser `private`, y que los métodos invocados por los clientes de una clase dada deben ser `public`. Sin embargo, los métodos que se llaman sólo por otros métodos de la clase como *funciones utilitarias* deben ser generalmente `private`. UML emplea marcadores de visibilidad para modelar la visibilidad de los atributos y las operaciones. La visibilidad pública se indica mediante la colocación de un signo más (+) antes de una operación o atributo; un signo menos (-) indica visibilidad

privada.

Actividad 1: Comienza por actualizar el diagrama de clases que habías confeccionado para la Práctica 3 agregando los marcadores de visibilidad.

Ejercicio: Implementación del sistema

Ahora ya sí, estamos listos para empezar a implementar el sistema. Primero convertiremos las clases del diagrama de clases en archivos de cabecera de C++. Este código representará el *esqueleto* del sistema. (Más adelante modificaremos estos archivos para incorporar el concepto orientado a objetos de la herencia.)

Sigue estas pautas para cada clase:

1. Usa el nombre que se localiza en el primer compartimento de una clase en el diagrama de clases para definir la clase en un archivo de encabezado. Recuerda usar las directivas del preprocesador `#ifndef`, `#define` y `#endif` para evitar que el archivo de encabezado se incluya más de una vez en el programa.

2. Usa los atributos que se localizan en el segundo compartimento de la clase para declarar los atributos.

3. Usa las asociaciones descritas en el diagrama de clases para declarar las referencias (o apuntadores, según sea apropiado) a otros objetos. Por ejemplo, observa en la Figura *Diagrama de clases que muestra las relaciones de composición* de la Práctica 3 que un objeto de la clase `SistemaMonitorizacion` está formado por un objeto de la clase `Pantalla`, un objeto de la clase `Teclado`, un objeto de la clase `Microfono` y un objeto de la clase `Interruptor`.

   Así, en la implementación de la clase `SistemaMonitorizacion` habrá un constructor que inicializa estos atributos con referencias a objetos actuales. Además, en esta clase, se deberán incluir (mediante `#include`) los archivos de encabezado que contienen las definiciones de las clases `Pantalla`, `Teclado`, `Microfono` e `Interruptor`, de manera que podamos declarar referencias a objetos de estas clases.

4. Mejora del punto anterior. Al incluir los archivos de encabezado para las clases `Pantalla`, `Teclado`, `Microfono` e `Interruptor`, se hace más de lo necesario. La clase `SistemaMonitorizacion` contiene referencias a objetos de estas clases, no verdaderos objetos; y la cantidad de información requerida por el compilador para crear una referencia difiere

de la que se requiere para crear un objeto. Recuerda que para crear un objeto, el programador debe proporcionar al compilador una definición de la clase que introduzca el nombre de la clase como un nuevo tipo definido por el usuario, y que indique los miembros de datos que determinen cuánta memoria se requiere para almacenar el objeto. Sin embargo, al declarar una referencia (o apuntador) a un objeto, sólo se requiere que el compilador sepa que la clase del objeto existe; esto es, no necesita conocer el tamaño del objeto.

Cualquier referencia (o apuntador), sin importar la clase del objeto al que hace referencia, sólo contiene la dirección de memoria de dicho objeto. La cantidad de memoria requerida para almacenar una dirección es una característica física del hardware de la computadora. Por ende, el compilador conoce el tamaño de cualquier referencia (o apuntador). Como resultado, es innecesario incluir el archivo de encabezado completo de una clase cuando se declara sólo una referencia a un objeto de esa clase; simplemente necesitamos introducir el nombre de la clase, pero no necesitamos proporcionar la distribución de los datos del objeto, ya que el compilador conoce de antemano el tamaño de todas las referencias.

C++ proporciona una instrucción conocida como **declaración anticipada**, la cual indica que un archivo de encabezado contiene referencias o apuntadores a una clase, pero ésta se encuentra fuera del archivo de encabezado. Así, podemos reemplazar las instrucciones `#include` mencionadas de la definición de la clase `SistemaMonitorizacion`, por declaraciones anticipadas de tales clases:

```
class Pantalla; // declaración anticipada de la clase Pantalla
class Teclado; // declaración anticipada de la clase Teclado
class Microfono; // declaración anticipada de la clase Microfono
class Interruptor; // declaración anticipada de la clase Interruptor
```

En vez de incluir (mediante `#include`) el archivo de encabezado completo para cada una de estas clases, sólo colocamos una declaración anticipada de cada clase en el archivo de encabezado para la clase `SistemaMonitorizacion`. Ten en cuenta que si la clase `SistemaMonitorizacion` contara con objetos actuales en vez de referencias, entonces tendríamos que incluir los archivos de encabezado completos.

Usar una declaración anticipada (en donde sea posible) en vez de incluir un archivo de encabezado completo nos ayuda a evitar un problema del preprocesador, conocido como inclusión circular. Este problema ocurre cuando el archivo de encabezado para la clase A incluye el archivo de encabezado para la clase B, y viceversa. Algunos

preprocesadores no pueden resolver tales directivas `#include`, lo cual produce un error de compilación. Por ejemplo, si la clase A sólo utiliza una referencia a un objeto de la clase B, entonces la instrucción `#include` en el archivo de encabezado de la clase A se puede reemplazar por una declaración anticipada de la clase B, para evitar la inclusión circular.

5. Usa las operaciones que se localizan en el tercer compartimento de la clase para escribir los prototipos de los métodos. Si todavía no se ha especificado un tipo de valor de retorno para una operación, declaramos el método con el tipo de valor de retorno `void`.