Práctica 9
Interfaces

(Rev 1.0)

1 Objetivos

Que el alumno comprenda el concepto de *interfaz* y los aspectos básicos de su implementación en C++.

2 Pre-laboratorio

2.1 Cuestionario

(Las preguntas de los cuestionarios vendrán en sus exámenes parciales del salón.)

1. ¿Qué es una clase base abstracta?
2. ¿En qué se diferencía una clase abstracta de una clase concreta?
3. ¿Qué es un método abstracto y qué es un método concreto?
4. ¿Qué es una interfaz?
5. ¿Cómo se *implementan* las interfaces en C++?
6. ¿Cómo se implementa la herencia múltiple en C++?
7. Cuando una clase concreta hereda de una clase abstracta, ¿qué es lo que realmente se está heredando?
8. ¿Para qué se usa la herencia múltiple en el contexto de las interfaces? *Esto es porque la herencia múltiple se puede utilizar sin meterse en el tema de interfaces*.
9. ¿Cuáles son las dos aplicaciones típicas de las interfaces?

(Artículo relacionado: http://programmers.stackexchange.com/questions/108240/why-are-interfaces-useful)

2.2 Actividades

1.

3 Teoría

3.1 Clases *mix-in*

En muchas ocasiones nos vamos a encontrar con que un cierto comportamiento no pertenece a una clase, pero aún así nos gustaría implementarla. Agregar tal comportamiento rompería con el esquema de la clase original porque estaríamos agregando algo que no es natural a ella. Tome como ejemplo una clase Reloj que debe de llevar la hora, pero no imprimirla. Sin embargo, vamos a necesitar imprimir la hora tarde o temprano. Un método Print() no encaja en lo que se supone van a ser nuestros relojes, porque así está indicado por diseño. Una alternativa sería escribir funciones "sueltas" para imprimir la hora, pero esto rompe el esquema de orientación a objetos. Una mejor opción y que usa programación orientada a objetos y abstracciones es utilizar clases ***mix-in***.

> *"Una clase **mix-in** responde a la pregunta ¿Qué más es capaz de hacer esta clase? Y la respuesta normalmente termina en '-ible' (NT: en inglés la terminación es '-able'). Las clases **mix-in** son una manera en la que el programador puede agregar funcionalidad a una clase sin comprometerse a la relación **es-un**.*
>
> *[…] Las clases **mix-in** se usan frecuentemente en las interfaces de usuario. En lugar de decir que una clase `PictureButton` es una imagen y un botón, uno podría decir que se trata de una `Image` que es `Clickable`. La clase **mix-in** `Clickable` lo único que hace es agregar los comportamientos "ratón abajo" y "ratón arriba"." [KLEPER11, pp. 92]*

En C++ las interfaces se representan con clases base abstractas puras, y se implementan (se proporciona el código de los métodos abstractos) usando la herencia múltiple, aunque para las interfaces la semántica de la herencia –es decir, el significado– es diferente.

En la Actividad 2 de esta práctica Ud. hará que una clase `Clock`, que no se supone debiera imprimir la hora, se convierta en una clase imprimible a través de la herencia y las interfaces.

4 Actividades

Actividad 1: Probando las interfaces

1. Compile y ejecute el archivo **test_device.cpp**.
2. Escriba una interfaz `IChannel` que se encargue de cambiar el canal en una televisión, o de emisora en una radio.
3. Cree una clase concreta `MiRadio` que implemente las interfaces `IDevice` e `IVolume` en forma similar a la clase concreta `CellPhone`. Luego implemente la interfaz `MiRadio` para un rango de emisoras de radio de FM desde 88.1 MHZ hasta 108.0 MHZ en pasos de 200 KHZ.

Actividad 2: Clases Mix-in

1. Cree un nuevo archivo **test_imprimible.cpp**.

2. Escriba una clase `Clock` que lleve la cuenta de los segundos transcurridos desde la media noche. La hora deberá ser puesta en el formato *hh:mm:ss*, cada uno como un entero con el método `SetTime()`. La hora deberá ser devuelta componente por componente con los métodos `GetHours()`, `GetMinutes()` y `GetSeconds()`. Sobrecargue al operador `==` para que indique si la hora de dos relojes son iguales o no. Escriba únicamente al constructor con argumentos. Pruebe su clase en el driver program.

3. Escriba una interfaz `Printable` con un único método abstracto `Print()`.

4. Escriba una clase `UserClock` que tenga un reloj que sea imprimible. La hora deberá ser impresa como cadena en formato de 12 horas, indicando si es AM o PM. Para esto su clase deberá heredar de la clase `Clock` e implementar la interfaz `Printable`. Pruebe su clase `UserClock` en el driver program.

Entregable: Los archivos test_device.cpp y test_imprimible.cpp; el cuestionario y el diagrama de clases de la Actividad 2.