Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Introducción a la Programación Orientada a Objetos
Taller # 3

"Herencia, polimorfismo e interfaces gráficas de usuario en Qt/C++ y Java"

Introducción

La usabilidad es un factor determinante en el éxito de un proyecto de software. Con ese propósito, brindar al usuario la posibilidad de interactuar con las aplicaciones mediante una interfaz, que sea gráfica, pero además intuitiva, completa y clara, logra facilitar las tareas y procesos del usuario.

En los lenguajes orientados a objetos, una manera sencilla y rápida en la que el programador puede desarrollar interfaces gráficas de usuario (GUI) es mediante la propiedad ó relación de herencia. En otras palabras, la herencia permite la reutilización de diferentes componentes, y entre ellos obviamente los componentes gráficos, ya existentes en las librerías de clases. En este contexto, surge el patrón de diseño que se observa en la Figura1.

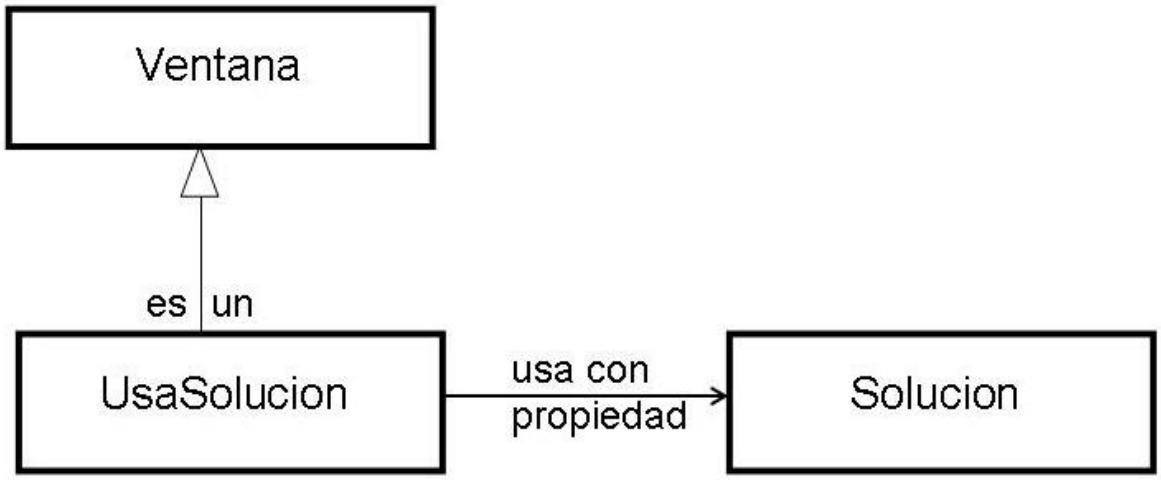

Figura 1. Patrón de diseño en aplicaciones con GUI.

La herencia ofrece otras ventajas además de la reutilización. Una de ellas es el polimorfismo: un solo mensaje, y diferentes formas de respuesta. En las clases unidas por una relación jerárquica de generalización a especialización es posible definir comportamientos polimórficos que agrupen bajo una misma interfaz de diferentes maneras de respuesta. Por ejemplo, el área y el perímetro son dos características inherentes a las figuras de dos dimensiones, pero la forma en que esta se calcula para diferentes instancias de figuras debe ser también diferente. El polimorfismo es entonces una propiedad que permite modelar tales situaciones.

El objetivo de este taller es que el estudiante aborde los conceptos de herencia y polimorfismo, entendiendo su importancia y las ventajas que estos ofrecen al momento de programar.

La fecha de entrega será el día 19 de noviembre de 2007 y pueden participar hasta 2 (dos) estudiantes, como máximo, en un mismo grupo. El fraude, entendido como la presentación de código fuente con alta similitud entre grupos diferentes, se penalizará con la nota de 0.0, para todos los grupos involucrados.

Cada grupo debe entregar, en el CampusVirtual, un archivo comprimido, llamado IPOO_Taller3.zip, que contenga:

- Lo solicitado en cada punto organizado en una carpeta independiente y cuyo nombre corresponda al punto abordado.
- Los códigos fuente. Estos archivos deben estar debidamente etiquetados, de acuerdo con lo discutido en clase, e incluir el nombre y código de cada miembro del grupo.
- Los diagramas de clases generados como imagen.
- Las imágenes generadas salvadas en un formato comprimido (PNG ó JPEG).

I) Aplicaciones con GUI en Qt/C++

En Qt los archivos que componen un proyecto de software se organizan al interior de una carpeta cuyo nombre se asocia con el nombre del proyecto. Ubicado en dicha carpeta desde la línea de comandos (tanto en los S.O. Linux y Windows) se deben ejecutar los siguientes comandos:

```
qmake   –project
qmake   –makefile
make
```

Esto con el propósito de que se generen de manera semi-automática y transparente para el programador código fuente auxiliar, *scripts* de compilación y la compilación misma. El nombre del ejecutable generado coincide con el nombre de la carpeta, es decir, con el del proyecto.

Con base en la aplicación `NeoNato.zip`, que construye un reporte con base datos solicitados al usuario, y de acuerdo a lo discutido en clase, resuelva:

1.1 [10%] En su estado actual, la aplicación no está validando adecuadamente que los datos de entrada estén de acorde a lo que se espera. Adicionalmente, el programa exhibe ciertas prácticas de programación que no son recomendables, como el uso `#define` para ajustar valores usados en la clase, y la asignación de nombres de atributos poco dicientes, ó que no se ajustan al estándar de programación discutido en clase. Modifique el programa de manera tal que, mediante una ventana emergente, se notifique al usuario de las omisiones de valores en el formulario, o de valores suministrados que no estén acorde a la lógica de lo esperado.
Una vez modificado el código fuente, para agregar la funcionalidad descrita en el punto anterior, adicione la documentación al estilo `javadoc` de manera tal que cubra todos los métodos y atributos.

1.2 [5%] Desarrolle un diagrama de clases detallado que ilustre, de acuerdo al lenguaje de implementación, Qt/C++ en este caso, las clases que componen la aplicación, así como las relaciones entre ellas.

II) Aplicaciones con GUI en Java
Con base en la aplicación `HinchaGUI.zip`, y de acuerdo a lo discutido en clase, resuelva:

2.1 [5%] La documentación que presenta la aplicación no emplea *tags* y está incompleta pues no cubre todos los métodos y atributos. Modifique los códigos fuente adicionando la documentación al estilo `javadoc` de manera tal que cubra todos los métodos y atributos. Una vez que estén modificados los archivos genere la documentación API mediante el uso del comando respectivo.

2.2 [5%] Desarrolle un diagrama de clases detallado que ilustre, de acuerdo al lenguaje de implementación, Java en este caso, las clases que componen la aplicación, así como las relaciones entre ellas.

III) Desarrollo de aplicaciones con GUI en Qt/C++
Desarrolle una aplicación con GUI en Qt/C++ que se ajuste al patrón de diseño de la Figura 1 y que permita al usuario calcular el promedio de la nota del curso de IPOO tomando como entrada las notas finales de talleres, proyecto y examen.

3.1 [15%] Desarrolle en Qt/C++ las clases: `UsaPromedioGUI` que actúa como interfaz gráfica para la clase `Promedio`, la cual satisface la funcionalidad arriba descrita. La interfaz debe controlar que los datos suministrados por el usuario se ajusten a lo esperado e informar al usuario cuando esto no sea así. Incorpore en el código fuente la documentación al estilo `javadoc` de manera tal que cubra todos los métodos y atributos.

3.3 [5%] Desarrolle un diagrama de clases detallado que ilustre, de acuerdo al lenguaje de implementación, Qt/C++ en este caso, las clases que componen la aplicación, así como las relaciones entre ellas.

IV) Desarrollo de aplicaciones con GUI en Java
Desarrolle una aplicación con GUI en Java que satisfaga la misma funcionalidad que la aplicación *ReporteNeoNato* del literal **I** de este taller y que se ajuste al patrón de diseño de la Figura 1.

4.1 [15%] Desarrolle en Java las clases: `NeoNatoGUI` que actúa como interfaz gráfica para la clase `ReporteNeoNato`, la cual satisface la funcionalidad arriba descrita. La interfaz debe controlar que los datos suministrados por el usuario se ajusten a lo esperado e informar al usuario cuando esto no sea así. Incorpore en el código fuente la documentación al estilo `javadoc` de manera tal que cubra todos los métodos y atributos, y genere la documentación API mediante el uso del comando respectivo.

4.2 [5%] Desarrolle un diagrama de clases detallado que ilustre, de acuerdo al lenguaje de implementación, Java en este caso, las clases que componen la aplicación, así como las relaciones entre ellas.

V) Polimorfismo
Con base en la clase `Figura2D` y de manera semejante a las clases `Rectángulo` y `Cuadrado` implemente las clases `Rombo` y `Circulo`, implementado en ellas las respuestas polimorficas respectivas.

5.1 [7.5%] Desarrolle en C++ la clase `Rombo`, que es una `Figura2D`, incorporando en el código fuente documentación mediante *tags*.

5.2 [7.5%] Desarrolle en C++ la clase `Circulo`, que es una `Figura2D`, incorporando en el código fuente documentación mediante *tags*.

5.3 [15%] Desarrolle en Qt/C++ o en Java la clase `Figuras2DGUI`, de manera libre (e.j. a su gusto o criterio), que es una interfaz para usar un número mínimo y máximo de 9 objetos que correspondan a subclases de `Figura2D` implementadas en C++ (e.j. no se debe generar una versión en Java). La interfaz debe permitir acceder en cualquier momento crear, enviar mensajes o liberar (destruir) objetos en particular, es decir al objeto k-esimo, para un valor k especificado por el usuario. Incorpore en el código fuente documentación mediante *tags*.

5.4 [5%] Desarrolle un diagrama de clases detallado que ilustre, de acuerdo al lenguaje de implementación las clases que componen la aplicación, así como las relaciones entre ellas. Si aplicación se comunica mediante *pipelines*, ilustre ello en el diagrama de clase empleando la metáfora del emparedado.