Ejercicio 8

Martín Eduardo Barriga Vargas

UML

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional, Rational Unified Process o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que programación estructurada es una forma de programar como lo es la orientación a objetos, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML solo para lenguajes orientados a objetos.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

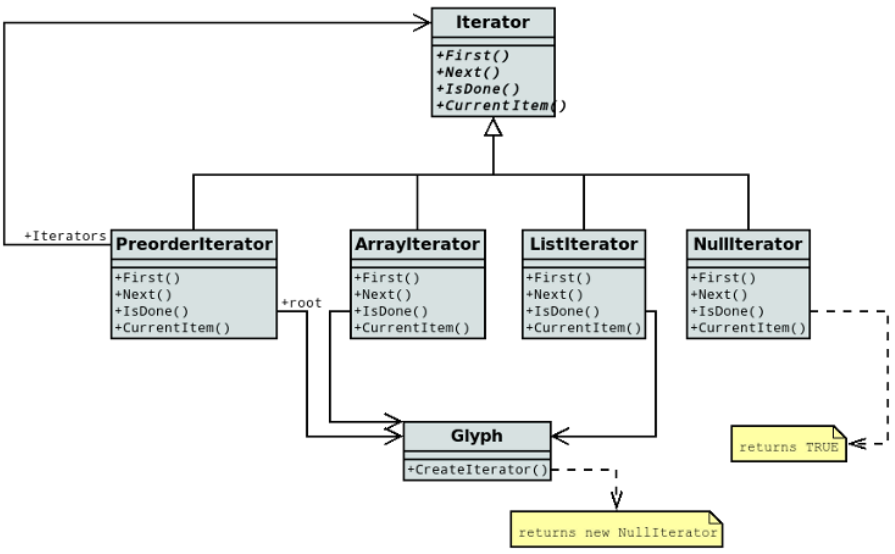


Diagrama de clases

En ingeniería de software, un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.
UML proporciona mecanismos para representar los miembros de la clase, como atributos y métodos, así como información adicional sobre ellos.

Para especificar la visibilidad de un miembro de la clase (es decir, cualquier atributo o método), se coloca uno de los siguientes signos delante de ese miembro:

+	Público
-	Privado
#	Protegido
/	Derivado (se puede combinar con otro)
~	Paquete

UML especifica dos tipos de ámbitos para los miembros: *instancias* y *clasificadores* y estos últimos se representan con nombres subrayados.

- Los miembros clasificadores se denotan comúnmente como “estáticos” en muchos lenguajes de programación. Su ámbito es la propia clase.
    - Los valores de los atributos son los mismos en todas las instancias
    - La invocación de métodos no afecta al estado de las instancias
- Los miembros instancias tienen como ámbito una instancia específica.
    - Los valores de los atributos pueden variar entre instancias
    - La invocación de métodos puede afectar al estado de las instancias(es decir, cambiar el valor de sus atributos)

Para indicar que un miembro posee un ámbito de clasificador, hay que subrayar su nombre. De lo contrario, se asume por defecto que tendrá ámbito de instancia.

Relaciones

Asociación:

Una asociación representa a una familia de enlaces. Una asociación binaria (entre dos clases) normalmente se representa con una línea continua. Una misma asociación puede relacionar cualquier número de clases. Una asociación que relacione tres clases se llama asociación ternaria.
A una asociación se le puede asignar un nombre, y en sus extremos se puede hacer indicaciones, como el rol que desempeña la asociación, los nombres de las clases relacionadas, su multiplicidad, su visibilidad, y otras propiedades.
Hay cuatro tipos diferentes de asociación: bidireccional, unidireccional, agregación (en la que se incluye la composición) y reflexiva. Las asociaciones unidireccional y bidireccional son las más comunes.
Por ejemplo, una clase vuelo se asocia con una clase avión de forma bidireccional. La asociación representa la relación estática que comparten los objetos de ambas clases.

Agregación:

La agregación es una variante de la relación de asociación “tiene un”: la agregación es más específica que la asociación. Se trata de una asociación que representa una relación de tipo parte-todo o parte-de.

Como se puede ver en la imagen del ejemplo (en inglés), un Profesor 'tiene una' clase a la que enseña.

Al ser un tipo de asociación, una agregación puede tener un nombre y las mismas indicaciones en los extremos de la línea. Sin embargo, una agregación no puede incluir más de dos clases; debe ser una asociación binaria.

Una agregación se puede dar cuando una clase es una colección o un contenedor de otras clases, pero a su vez, el tiempo de vida de las clases contenidas no tienen una dependencia fuerte del tiempo de vida de la clase contenedora (de el todo). Es decir, el contenido de la clase contenedora no se destruye automáticamente cuando desaparece dicha clase.

En UML, se representa gráficamente con un rombo hueco junto a la clase contenedora con una línea que lo conecta a la clase contenida. Todo este conjunto es, semánticamente, un objeto extendido que es tratado como una única unidad en muchas operaciones, aunque físicamente está hecho de varios objetos más pequeños.

Composición:

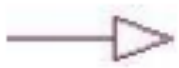
Composición es una forma fuerte de composición donde la vida de la clase contenida debe coincidir con la vida de la clase contenedor. Los componentes constituyen una parte del objeto compuesto. De esta forma, los componentes no pueden ser compartidos por varios objetos compuestos. La supresión del objeto compuesto conlleva la supresión de los componentes.
El símbolo de composición es un diamante de color negro colocado en el extremo en el que está la clase que representa el “todo” (Compuesto).

Dependencia:

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). Se denota por una flecha punteada. El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación grafica que instancia una ventana (la creación del Objeto Ventana está condicionado a la instanciación proveniente desde el objeto Aplicación).

Herencia:

Indica que una clase (clase derivada) hereda los métodos y atributos especificados por una clase (clase base), por lo cual una clase derivada además de tener sus propios métodos y atributos, podrá acceder a las características y atributos visibles de su clase base (public y protected).



Clases de los ejercicios en C hechos en clase

Ejercicio 1. Memoria USB

```
public class USB{
    String nombre;
    float capacidad;
    String tipoArchivo;
    float capacidadEnUso;

    public void inicializar(String nombreInicial, float capacidadInicial, String tipoArchivoInicial){
        nombre = nombreInicial;
        capacidad = capacidadInicial;
        tipoArchivo = tipoArchivoInicial;
        capacidadEnUso = 0.0;
    }

    public void cambiarNombre(String nuevoNombre) {
        nombre = nuevoNombre;
    }

    public void formatear(String nuevoNombre, String nuevoTipoArchivo){
        nombre = nuevoNombre;
        tipoArchivo = nuevoTipoArchivo;
        capacidadEnUso = 0.0;
    }
}
```

Ejercicio 2.1 Fracciones

```java
public class Fecha{
    int dia;
    int mes;
    int anio;

    public int obtenerNumDiasmMes(){
        if( (mes == 4) || (mes == 6) || (mes == 9) || (mes==11)) return 30;
        else if(mes == 2) return 28;
        return 31;
    }

    public int serFinDeMes(){
        if(dia == obtenerNumDiasmMes()) return 1;
        return 0;
    }

    public void obtenerDiaSiguiente(){
        if(serFinDeMes() == 1){
            if(mes == 12){
                mes = 1;
                anio++;
            }
            else {
                mes++;
            }
            dia =1;
        }
        else {
            dia++;
        }
    }
}
```

Ejercicio 2.2 Fecha

```java
import javax.swing.JOptionPane;
public class Fraccion{
    int numerador;
    int denominador;
    int i;

    public void sumar(Fraccion fraccion1 Fraccion fraccion2){
        denominador = fraccion1.denominador * fraccion2.denominador;
        numerador = fraccion1.numerador * fraccion2.denominador + fraccion1.denominador * fraccion2.numerador;
    }

    public void simplificar(){
        i = 2;
        do{
            if((denominador % i == 0) && (numerador % i == 0)){
                denominador /= i;
                numerador /= i;
                i = 2;
            }
            else {
                if( (i > denominador) || (i > numerador)) break;
                i++;
            }
        }while(1);
    }

    public void imprimir(){
        JOptionPane.showMessageDialog(null, numerador + "/" + denominador, "Resultado", JOptionPane.INFORMATION_MESSAGE);
    }
}
```