Taller de Programación Orientada a Objetos

Modelos y Paradigmas de Programación

Escuela de Ingeniería de Sistemas y Computación

Universidad del Valle

Profesor Juan Francisco DIAZ FRIAS*

A continuación Usted encontrará una serie de ejercicios que debe resolver **individualmente**. No se admite ninguna consulta con sus compañeros de otros grupos de trabajo, pero sí con el profesor. Esa es la hipótesis básica sobre la cual se fundamenta esta tarea. Cualquier violación a esta regla será considerada un intento de copia y causará la anulación del taller.

La solución a todos y cada uno de los puntos del taller se debe almacenar en un archivo `.oz` tal como se describe en cada punto. Usted debe entregar como solución al examen, el conjunto de esos archivos empaquetado y comprimido en un solo archivo cuyo nombre debe ser análogo a `taller5JFDiaz.zip` o `taller5JFDiaz.tgz` o `taller5JFDiaz.tar.gz` o `taller5JFDiaz.rar`. Al desempaquetar este archivo, debo encontrar solamente los archivos siguientes: `NuevaHeredarDe.oz` y `Formulas.oz` con el contenido descrito abajo. Adicionalmente debe entregar un archivo `Practica5.pdf`, con el desarrollo del taller que no corresponde a código Oz.

No envíe ningún archivo adicional, así lo haya utilizado para sus pruebas. Su solución debe ser enviada a través del Campus Virtual en el enlace correspondiente al Taller de Programación Relacional según las fechas indicadas allí. El sistema no permitirá que se envíen soluciones después de dicha fecha.

*juanfco.diaz@correounivalle.edu.co

1. *Implementando herencia múltiple en general*

   En la sección 7.6 del libro se presenta la implementación simplificada del sistema de objetos. Allí encuentra la implementación de la función `{HeredarDe C1 C2}` la cual recibe una clase base `C1` y dos superclases `C2` y `C3` y devuelve la clase resultante de combinar `C1`, `C2` y `C3` de acuerdo a las reglas de herencia. En este ejercicio su tarea consiste en generalizar la función `HeredarDe` de manera que funcione para cualquier número de superclases y no solamente dos. Usted debe implementar una función `{NuevaHeredarDe Clase SuperClases}` que recibe una `Clase` base y una lista de `SuperClases` y devuelve la clase correspondiente de combinar `Clase` con las `SuperClases` de acuerdo a las reglas de herencia. Su implementación debe verificar que la herencia sea válida. En caso de no serlo debe señalar un mensaje de error.

   Almacene la función en el archivo **`NuevaHeredarDe.oz`**. Puede comprobar su implementación con los ejemplos que encuentra en los archivos **`PruebaHerenciaExito.oz`** y **`PruebaHerenciaFalla.oz`** que encontrará en el sitio del curso en el campus virtual.

2. *Objetos evaluadores: Polimorfismo*

   El objetivo de este ejercicio es implementar objetos para evaluar fórmulas. Cada subexpresión de una fórmula será representada por un objeto, incluidos los símbolos de variables. Para ello se definen diferentes tipos de expresiones, y cada uno de ellos se implementa con una clase propia. Por ejemplo, se puede representar la fórmula $3x^2 - xy + y^3$ por

```
declare VarX={New Variable inic(0)}
VarY={New Variable inic(0)}
local
  ExprX2={New Potencia inic(VarX 2)}
  Expr3 ={New Constante inic(3)}
  Expr3X2={New Producto inic(Expr3 ExprX2)}
  ExprXY={New Producto inic(VarX VarY)}
  Expr3X2mXY={New Diferencia inic(Expr3X2 ExprXY)}
  ExprY3={New Potencia inic(VarY 3)}
in
  Formula={New Suma inic(Expr3X2mXY ExprY3)}
end
```

   Luego, utilizamos los objetos `VarX`, `VarY` y `Formula` para evaluar la fórmula. Se asocian valores a las variables, y luego se invoca el método `evaluar(X)` del objeto `Formula`. Se debe poder volver a evaluar la fórmula con otros valores, como se muestra en el ejemplo siguiente:

```
{VarX asg(7)}
{VarY asg(23)}
{Browse {Formula evaluar($)}}

{VarX asg(5)}
{VarY asg(8)}
{Browse {Formula evaluar($)}}
```

Implemente las clases `Variable`, `Constante`, `Suma`, `Diferencia`, `Producto`, y `Potencia` para representar fórmulas.

- Todas estas clases deben implementar el método `evaluar(X)`. Este método es entonces polimorfo pues se puede invocar cualquiera que sea la clase del objeto fórmula.
- Las clases tienen también sus métodos propios. Por ejemplo, su método de inicialización, para el cual los argumentos tienen significados diferentes según la clase. Otro ejemplo es el método `asg(X)` de la clase `Variable`, el cual permite asociar un valor a la variable antes de evaluar la fórmula. Este método está ausente de las otras clases.

Almacene la implementación en el archivo `Formulas.oz`.