Pauta Lectura 2 CC51H: Programación Orientada a Objetos

Profesora: Nancy Hitschfeld Kahler.
Auxiliar: Alcides Quispe Sanca.
Ayudante: Diego Díaz Espinoza

July 20, 2010

1. Pregunta uno

   Un contrato es un acuerdo entre dos o más clases o métodos. Ellos definen qué se espera como entrada y como salida en un método o qué se espera como estado inicial y final de una clase. Se dividen en tres tipos: precondiciones, postcondiciones e invariantes. Precondiciones son las condiciones previas a la ejecución de un método o instanciación de clase. Postcondiciones son las condiciones posteriores a la ejecución del método o estado final de la instanciación de una clase. Invariante son las condiciones transversales durante la ejecución de un método o durante el periodo de vida de una clase. Sirven para construir software (diseño de software) de una forma más controlada, confiable y de manera modular. Además se adaptan perfectamente para hace debugging en la etapa de testing. Se centran en describir las condiciones (factores externos) que permiten un buen funcionamiento de las clases y/o métodos.

2. Pregunta dos.

   Cuando se usa herencia (polimorfismo, enlace dinámico, redeclaración) sólo se permite que los contratos de la subclase (subcontratos) sean mejores (en el sentido de compatibles) con las clases padres. Esto pues debe considerarse que las subclases pueden ser usadas en el contexto de la clase padre. De este modo, sólo se permite que en cuanto a las

pre-condiciones, éstas sean más débiles que las del padre; mientras que las post-condiciones deben ser más fuertes que las del padre. Esto está relacionado con las leyes de la varianza y contra-varianza, pues cabe recordar que la pre-condiciones están asociadas a los estados anteriores de la clase, mientras que las post-condiciones están asociadas a los estados posteriores de la clase.

En el caso de los invariantes, deben ser aplicables a la clase y la sub-clase. De este modo los invariantes del padre se deben heredar a los hijos.

3. Pregunta tres.

   Una falla ocurre en la ejecución de una rutina, específicamente al momento de efectuar los contratos. Si alguno de los contratos no se cumple, se está en presencia de una falla. Pueden ser debido a un malfuncionamiento de hardware, errores en la implementación o otros eventos externos inesperados. Se incluyen de 3 formas:

   (a) Dejar los objetos en un estado consistente y efectura un nuevo intento.

   (b) Dejar los objetos en un estado consistente, evitar el contrato y reportar una falla al sistema.

   (c) Falsa-alarma;chequear si se trata de una falsa alarma.

4. Pregunta cuatro.

```
abstract class Figure{
  public:
    public Figure();
    abstract Figure clone()
    abstract Polygon convertToPolygon()
    abstract boolean equals(Figure);
    abstract double area();
}

class Triangle extends Figure
{
```

```
     //INVARIANTE
    assert (this.area()!=0.0)&&(check_all_points!=null)

     Point[] points;
     public Triangle( Point p1, Point p2, Point p3)
     {
  //PRECONDICION
  assert (p1!=null)&&(p2!=null)&&(p3!=null);

  //POSTCONDICION
  assert (this.area() != 0.0)&&(check_all_points(this.points));
     }

     Figure clone()
     {

//PRECONDICION
  //INVARIANTE
//POSTCONDICION
  //INVARIANTE PARA LA NUEVA FIGURA
     }

     boolean equals(Figure f)
     {
  //PRECONDICION
  assert (f!=null);



     }

     Polygon convertToPolygon();
     double area();

}
class Rectangle extends Figure
{
     //INVARIANTE
```

```
      assert (this.area()!=0.0)&&(center!=null)

      Point center;
      double height, width;
      public Rectangle( Point _center, double _ height, double _width)
      {
  //PRECONDICION
  assert (_center!=null)$$(_height>0)&&(_widht>0);


  //OSTCONDICION
  assert (this.area() != 0.0)&&(_center!=null)$$(_height>0)&&(_widht>0);
      }

      Figure* clone();
      boolean equals();
      Polygon convertToPolygon();
      double area();
}
class Polygon extends Figure{
    Point[] points;
    int numero_puntos;
    public Polygon( Point[] _points, int n_points)
     {
  //PRECONDICION
  assert (n!=null)&&(n>0)&&(points!=null);

  //POSTCONDICION
  assert (this.area() != 0.0)&&(points!=null);


     }
    Figure* clone();
    boolean equals(Polygon p)
     {
  //PRECONDICION
  assert (P!=null)&&(es_poligono(P));
```

```
  //POSTCONDICIONES
  assert (this.area>0)&&(es_poligono(P));

     }

    Polygon convertToPolygon();
    double area();

    check_polygon(Polygon p1);
    //CHEQUEA QUE EL POLIGONO SEA VALIDO

    check_all_points(Points[] points, int length)
    {
for(int i=0; i<length; i++)
  assert (points[i]!=null)

    }

}
```