TÉCNICO PROCESAMIENTO DE PRUEBAS DE SOFTWARE

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

Servicio Nacional de Aprendizaje SENA

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

EJEMPLO DE REFACTORIZACIÓN

Este es un ejemplo muy básico de un programa para calcular el valor total de una compra en un negocio de computación. A partir de que el programa sabe que artículos se compraron para ser facturados, se calcula importe total de la factura. Existen tres tipos de artículos: los que son tipificados como software, los artículos de hardware y otros, que no entran en ninguna de las dos categorías anteriores. Además de calcular el costo de los artículos teniendo en cuenta si existen impuestos que regulen la operación, en este caso el IVA, se calculan una serie de bonificaciones según el tipo de articulo comprado. El diagrama de clases (figura 1):

A continuación, se lista el código fuente de cada clase en C#

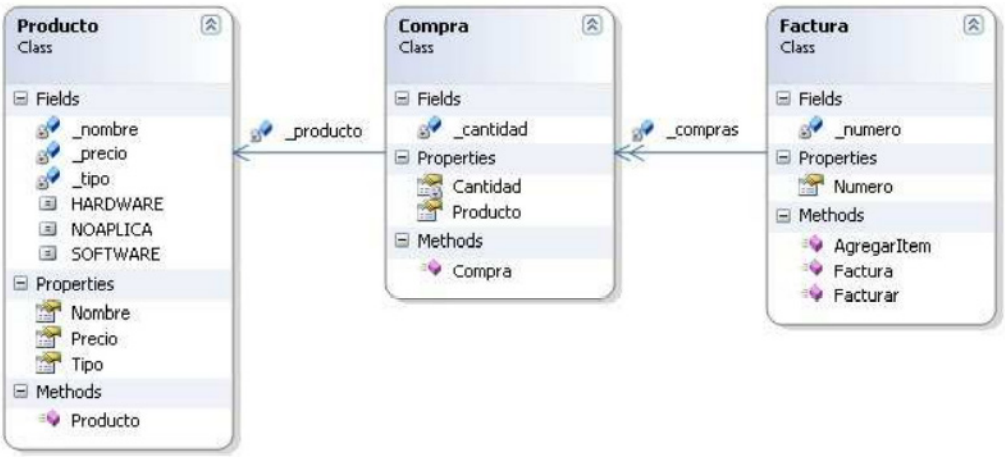

Figura 1: Diagrama de clases

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

La clase producto

```
class Producto {
        public const int SOFTWARE=1;
        public const int HARDWARE=2;
        public const int NOAPLICA=0;

        private string _nombre;
        private float _precio;
        private int _tipo;


        public Producto( string nombre, float precio, int tipo ) {
            _nombre = nombre;
            _precio = precio;
            _tipo = tipo;
        }

        public string Nombre
        {
            get { return _nombre;}
            set { _nombre = value; }
        }
        public float Precio {
            get { return _precio; }
            set { _precio = value; }
        }

        public int Tipo {
            get { return _tipo; }

        set { _tipo = value; }
    }
```

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

La clase compra

```
class Compra {
        private Producto _producto;
        private int _cantidad;

        public Compra( Producto producto, int cantidad) {
            _producto = producto;
            _cantidad = cantidad;
        }

        public Producto Producto {
            get { return _producto; }
            set { _producto = value; }
        }

        private int Cantidad {
            get { return _cantidad; }
            set { _cantidad = value; }
        }

}
```

Clase Factura

Representa el importe a pagar por todas las compras realizadas: esta clase es la que se encarga de determinar el valor a pagar de todas las compras realizadas. Esta clase posee un método llamado facturar() que es el encargado de calcular el valor total y estado de cuenta de la factura, aplicar el impuesto correspondiente y además calcular puntos por las compras realizadas. Ver diagrama de secuencia figura 2.

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

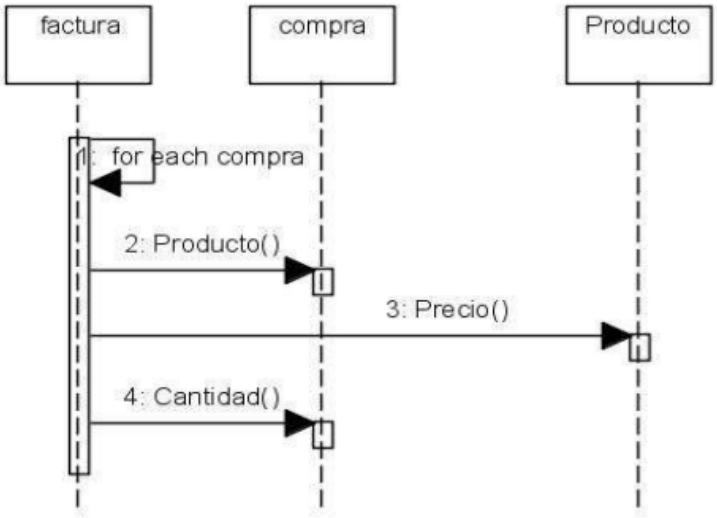

Figura 2: Diagrama de secuencia de método facturar()

```
class Factura {

        private List<Compra> _compras;
        private string _numero;

        public Factura(string numero) {
            _numero = numero;
            _compras = new List<Compra>();
        }

        public void AgregarItem(Compra item) {
            _compras.Add(item);
        }

        public String Numero {
            get { return _numero; }
            set { _numero = value; }
        }

        public string Facturar() {
            double totalFactura;
            IEnumerator it = _compras.GetEnumerator();
            int puntaje = 0;
            int ticket = "";


            totalFactura = 0;
            while (it.MoveNext) {
                double valorCompra = 0;
                Compra compra = it.Current;
                //Calcular el valor de una compra de la factura
                switch (compra.Producto.Tipo){
                    case Producto.HARDWARE:
                        valorCompra = compra.Producto.Precio * 10.5;
                        break;
                    case Producto.SOFTWARE:
                        valorCompra = compra.Producto.Precio * 21;
                        break;
                    case Producto.NOAPLICA:
                        valorCompra = compra.Producto.Precio;
                        break;

                   }
                   // calcular bonificacion
                   puntaje++;
                   if (compra.Producto.Tipo() == Producto.HARDWARE && compra.Cantidad >1 ) puntaje++;
                   // Calcular el valor Total de la factura
                   totalFactura += valorCompra;
                   // imprimir
                   ticket= "\t" + compra.Producto.Nombre + "\t" + compra.Cantidad +"\t" +valorCompra + "\n";
                }


                // Totalizar
                ticket = "El total de su compra es :" + totalFactura.ToString() + "\n";
                ticket = "Su puntaje acumunlado en esta compra es: " + puntaje + "\n";
                }
        }
```

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

El código fuente que se encuentra en el método facturar, sin dudas tiene algunos problemas, si quisiera ser visto desde la programación orientada a objetos. La escasa posibilidad de reutilización, por ejemplo, habría que pensar qué sucedería si las reglas de cálculo de los impuestos como el IVA variaran; se debería reescribir completamente el método. Otro cambio al que podría estar sujeta la aplicación podría ser la variación en la clasificación de los tipos de productos.

Antes de producir alguna transformación al código fuente de esta aplicación, hay que tener en cuenta que la refactorización, por definición, no debe introducir ningún cambio en el comportamiento externo de la aplicación. Por ende, hay que estar seguro de que esto no suceda, la forma de asegurarnos esto, es precisamente haciendo un muy buen conjunto de test de unidad para esta porción de código fuente.

```
public string Facturar() {
    double totalFactura;
    IEnumerator it = _compras.GetEnumerator();
    int puntaje = 0;
    int ticket = "";


    totalFactura = 0;
    while (it.MoveNext) {
        double valorCompra = 0;
        Compra compra = it.Current;
        //Calcular el valor de una compra de la factura
        switch (compra.Producto.Tipo){
            case Producto.HARDWARE:
            valorCompra = compra.Producto.Precio * 10.5;
            break;
         case Producto.SOFTWARE:
            valorCompra = compra.Producto.Precio * 21;
            break;
         case Producto.NOAPLICA:
            valorCompra = compra.Producto.Precio;
            break;
    }
    // calcular bonifica
    puntaje++;
    if (compra.Producto.Tipo() == Producto.HARDWARE && compra.Cantidad >1 ) puntaje++;
    // Calcular el valor Total de la factura
    totalFactura += valorCompra;
    // imprimir
    ticket= "\t" + compra.Producto.Nombre + "\t" + compra.Cantidad +"\t" +valorCompra + "\n";
 }
// Totalizar
ticket = "El total de su compra es :" + totalFactura.ToString() + "\n";
ticket = "Su puntaje acumulado en esta compra es: " + puntaje + "\n";
}
```

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

Este nuevo método podría llamarse CalcularCompra(compra), quedando el nuevo fragmento de código así:

```
public string Facturar() {
            double totalFactura;
            IEnumerator it = _compras.GetEnumerator();
            int puntaje = 0;
            int ticket = "";


            totalFactura = 0;
            while (it.MoveNext) {
                double valorCompra = 0;
                Compra compra = it.Current;
                //Calcular el valor de una compra de la factura
                valorCompra =CalcularCompra(compra);
                // calcular bonifica
                puntaje++;
                if (compra.Producto.Tipo() == Producto.HARDWARE && compra.Cantidad >1 ) puntaje++;
                // Calcular el valor Total de la factura
                totalFactura += valorCompra;
                // imprimir
                ticket= "\t" + compra.Producto.Nombre + "\t" + compra.Cantidad +"\t" +valorCompra + "\n";
             }
            // Totalizar
            ticket = "El total de su compra es :" + totalFactura.ToString() + "\n";
            ticket = "Su puntaje acumunlado en esta compra es: " + puntaje + "\n";
            }

        private double CalcularCompra(Compra Item) {
            double valor = 0;
            switch (item.Producto.Tipo) {
                case Producto.HARDWARE:
                    valor = item.Producto.Precio * 10.5;
                    break;
                case Producto.SOFTWARE:
                    valor = item.Producto.Precio * 21;
                    break;
                case Producto.NOAPLICA:


                    valor = item.Producto.Precio;
                    break;
            }
            return valor;
    }
```

Luego de producido el cambio, el siguiente paso es correr los tests para el nuevo fragmento de código y verificar que todos los tests resulten favorables. Una vez hecho esto se puede observar cómo al reducir en tamaño el método, un cambio casi cosmético, se ha favorecido el código fuente en varios factores:

- **Comprensibilidad:** el código fuente ha quedado más fácil de comprender.
- **Complejidad:** el código fuente ha quedado menos complejo. Se acota la responsabilidad del método facturar.
- **Legibilidad:** el código fuente es ahora legible.

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

- **Mantenibilidad:** el código fuente es más fácil de mantener. Supongamos un cambio en la forma de cálculo de la facturación, ésta ha quedado aislada del resto de la funcionalidad de la clase en el método CalcularCompra().

Si bien este cambio ha aportado mejoras desde el punto de vista de la estructura interna del código fuente, y además se ha validado que el comportamiento externo del mismo no ha cambiado (testing) es innegable que se podría mejorar aún más esta estructura interna.

Si leemos más detenidamente el código resultante veremos que en CalcularCompra() no se está utilizando ningún dato que pertenece a la factura, por ende, se puede llegar a la conclusión que el método CalcularCompra() está en el objeto equivocado, pues debería pertenecer a la clase Compra. He aquí la posibilidad de aplicar otra refactorización propuesta también en llamada" Mover Método" que consiste en mover un método de una clase a otra. En este caso se debería mover CalcularCompra() de la clase Factura a la clase Compra.

```
class Compra {

    private Producto _producto;
    private int _cantidad;

    public Compra (Producto producto, int cantidad)
        _producto = producto;
        _cantidad = cantidad;
    }



    private double CalcularCompra() {
        double valor = 0;
        switch (_producto.Tipo) {
            case Producto.HARDWARE:
                valor = _producto.Precio * 10.5;
                break;
            case Producto.SOFTWARE:
                valor = _productoPrecio * 21;
                break;
            case Producto.NOAPLICA:
                valor = _productoPrecio;
                break;
        }
        return valor;
    }
}
```

ACTIVIDAD REFACTORIZANDO CÓDIGO FUENTE

Tras haber aplicado este cambio, que desde el punto de la programación orientada a objetos es más lógico que el método CalcularCompra pertenezca a la clase Compra que a la clase Factura. Otra vez se han aplicado cambios internos que no modifican el comportamiento externo, pero aportan a la estructura del código fuente mejoras sustanciales.