Patrones de Diseño

Percy Taquila Carazas, Katerin Merino Quispe, Abraham Lipa Calabilla

October 10, 2020

Abstract

Design patterns provide a coded mechanism for describing problems and their solution in a way that allows the software engineering community to design knowledge for reuse. A pattern describes a problem, indicates the context and allows the user to understand the environment in which the problem occurs, and lists a system of forces that indicate how the problem can be interpreted in context, and the way in which the problem is applied. solution. The Abstract Factory pattern is usually implemented with manufacturing methods that are also generally called from within the Template Method.

Resumen

Los patrones de diseño dan un mecanismo codificado para describir problemas y su solución en forma tal que permiten que la comunidad de ingeniería de software diseñe el conocimiento para que sea reutilizado. Un patrón describe un problema, indica el contexto y permite que el usuario entienda el ambiente en el que sucede el problema, y enlista un sistema de fuerzas que indican cómo puede interpretarse el problema en su contexto, y el modo en el que se aplica la solución. El patron Abstract Factory suele implementarse con metodos de fabricacion que tambien generalmente son llamados desde el interior de Template Method.

I. Introduccion

El aprendizaje es esencial para la mayoría de las arquitecturas de redes neuronales, por lo que la elección de un algoritmo de aprendizaje es un punto central en el desarrollo de una red, este implica que una unidad de procesamiento es capaz de cambiar su comportamiento entrada/salida como resultado de los cambios en el medio.

El camino hacia la construccion de maquinas inteligentes comienza en la Segunda Guerra Mundial con el diseÑo de computadoras analogicas ideadas para controlar cañones antiaereos o para la navegacion. Algunos investigadores observaron que existıan semejanzas entre el funcionamiento de estos dispositivos de control y los sistemas reguladores de los seres vivos. De este modo, combinando los avances de la electronica de la posguerra y los conocimientos sobre los sistemas nerviosos de los seres vivos, inicio el reto de construir maquinas capaces de responder y aprender como los animales.

II. Desarrollo

i. Patrones de diseño estructurales

Consisten en configurar la estructura de nuestra aplicación para cumplir con los principios SOLID, así como mejorar la usabilidad y la mantenibilidad del código. Podemos aplicar los muy conocidos métodos de:

- Herencia
- Composición
- Agregación

También debemos tener en cuenta que hay tres formas de definir estructuras de datos:

- Estáticamente
- Generado por código
- Dinámicamente

i.1 Adaptadores

Tal como el nombre lo dice, mediante este patrón de diseño, podemos adaptar la funcionalidad de una clase a través de una interfaz.

Si tenemos una clase Punto y una clase Linea:

```
public class Punto
{
    public int X, Y;
}
public class Linea
{
    public Punto Inicio, Fin;
}
```

Pensamos en una Figura como una colección de Líneas:

```
public abstract class Figura :
    Collection<Linea> {}
```

Y creamos una clase Rectangulo a partir de la clase Figura (mediante herencia)

```
public class Rectangulo : Figura
{
    public Rectangulo(int x, int y
        , int ancho, int alto)
    {
        // Codigo que agrega
            lineas a la coleccion
    }
}
```

Nos toparemos con un problema. En la interfaz que tenemos para pintar en pantalla tenemos un método que solo funciona con Puntos.

```
public static void PintarPunto(
    Punto p)
{
    // Codigo para pintar en las
        coordenadas X, Y de p
}
```

Para solucionar eso, tenemos los Adaptadores. Necesitamos que el adaptador "convierta" una Línea en una colección de Puntos para que puedan ser pintados con la interfaz.

```
public class AdaptadorLineaAPunto
    : Collection<Punto>
{
    public AdaptadorLineaAPunto(
        Linea linea) {
        // Codigo para agregar los
            puntos en el
            recorrido de las
            Lineas
    }
}
```

Finalmente, implementamos la solución de la siguiente forma.

```
private static void PintarPuntos()
{
    foreach (var linea in
        rectangulo)
    {
        var adaptador = new
            AdaptadorLineaAPunto(
            linea);
        adaptador.ForEach(
            PintarPunto);
    }
}
```

Tomamos cada Linea en el Rectangulo, almacenamos en una variable una instancia del adaptador y pintamos cada Punto en el adaptador.

III. Conclusiones

Los patrones de diseño dan un

IV. Recomendaciones

- Cuando se conoce el efecto colateral que conlleva el patrón de diseño y es viable la aparición de este efecto.
- Suministrar alternativas de diseño para poder tener un software flexible y reutilizable.

Referencias

[1] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John(1995).Design Patterns: Elements of Reusable Object- Oriented Software. Reading,Massachusetts: Addison Wesley Longman, Inc.

[2] Nesteruk, D. (2019). Design Patterns in .NET: Reusable Approaches in C and F for Object-Oriented Software Design (1st ed.). Apress.