Daza Corredor Alejandro Paolo

Patrones Creacionales

Builder

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [Christopher Alexander].

1. Introducción

El Patrón de diseño Builder o Constructor es usado para permitir la creación de una variedad de objetos complejos desde un objeto fuente(Producto), el objeto fuente se compone de una variedad de partes que contribuyen individualmente a la creación de cada objeto complejo a través de un conjunto de llamadas a interfaces comunes de la clase Abstract Builder.

2. Builder

Nombre del patrón:
Builder

Clasificación del patrón:
Creacional.

Intención:
Su intención es de abstraer el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto, de tal forma que el mismo proceso de construcción pueda crear representaciones diferentes.

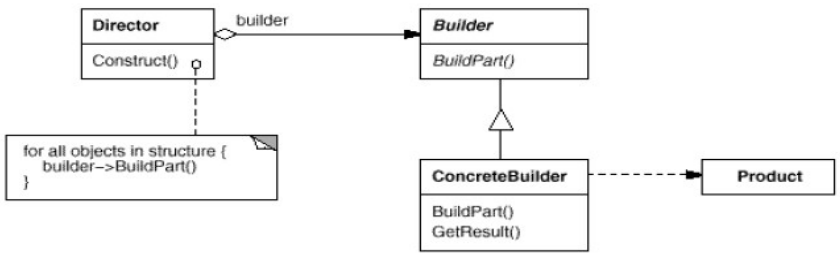
También conocido como:
Builder.

Motivación:
Un lector de RTF (Rich Text Format) debe tener la capacidad de convertir RTF a muchos formatos, el problema esta en que esto debe respetar el principio de abierto-cerrado, es decir debe ser fácil adicionar una nueva conversión sin modificar el lector. Cada conversor de documentos es llamado constructor y al lector de documentos se le llama director.

Aplicabilidad:
El patrón Builder debe ser usado cuando:

- Se requiera un algoritmo para que la creación de objetos complejos sea independiente de las partes que construyen el objeto y de como este es armado.
- El proceso de construcción debe permitir diferentes representaciones del objeto que se construye.

Estructura:



Participantes:

- **Builder**: especifica una interfaz abstracta para crear un objeto complejo Producto.
- **ConcreteBuilder**: construye y ensambla las partes del Producto implementando la interfaz de Constructor, define y mantiene un rastreo de la representación que crea, provee una interfaz para retornar el producto.
- **Director**: construye un objeto usando la interfaz de Constructor.
- **Producto**: representa el objeto complejo en construcción, incluye clases que definen las partes que lo componen, incluye interfaces para armar las partes en un resultado final.

Colaboraciones:

- El cliente crea el objeto Director y lo configura con el objeto Constructor deseado.
- El Director notifica al Constructor cuando quiere que una parte del producto sea construida.
- El Constructor maneja las peticiones del Director y adiciona partes al producto.
- El Cliente recupera el Producto del Constructor.

Consecuencias:

- Desacopla al director del constructor, permitiendo nuevas representaciones de objetos cambiando el constructor
- Aisla a los clientes de la estructura interna del producto
- Permite controlar la construcción paso a paso de un objeto complejo

Implementación:

- La implementación por defecto en Constructores no debe hacer nada ya que esta esta implementada en los concretos.
- Si se necesita acceder a partes de producto ya construido el constructor devolverá los nodos hijos al director y este se les pasará de nuevo al constructor más la nueva información para construir el nodo padre.

Código de ejemplo:

```
class Director{
   private Constructor constructor;
   public Director (Constructor constructor){
          this.constructor = constructor;
   }
   public void construir(String datos){
          String aux1,aux2,dato;
          StringTokenizer st = new StringTokenizer(datos);
          while (st.hasMoreTokens()) {
               aux1 = st.nextToken();
               if (aux1.equals("<CLIENTE>")){
                   dato = new String();
                   while (!((aux2 = st.nextToken()).equals("</CLIENTE>")))
                   dato = dato + " " + aux2;
                   constructor.construirCliente(dato);
               }
               if (aux1.equals("<PRODUCTO>")){
                   dato = new String();
                   while (!((aux2 = st.nextToken()).equals("</PRODUCTO>")))
                        dato = dato + " " + aux2;
                   constructor.construirProducto(dato);
               }
          }
   }
}



public class EjemploBuilder{
   public static void main(String ar[]){
          String datos = "<CLIENTE> Jose Juncal </CLIENTE>
                              <PRODUCTO> Miel </PRODUCTO>
                              <PRODUCTO> Tomates </PRODUCTO>";
          ConstructorConcreto constructorconcreto = new ConstructorConcreto();
          Director director = new Director(constructorconcreto);
          director.construir(datos);
          Cliente c = constructorconcreto.getCliente();
          System.out.println(c);
   }
}
```

```
class Constructor{
    public void construirCliente(String cliente){}
    public void construirProducto(String producto){}
}


class ConstructorConcreto extends Constructor{
    private Cliente cliente;
    public void construirCliente(String datoscliente){
          cliente = new Cliente(datoscliente);
    }
    public void construirProducto(String datosproducto){
          cliente.addProducto(datosproducto);
    }
    public Cliente getCliente(){
          return cliente;
    }
}
```

Usos conocidos:

- Usado en aplicaciones de conversores de documentos RTF.
- El servicio Configurator Framework en ambientes adaptativos de comunicaciones usa el patrón para construir componentes de servicios de red que son ligados a un servidor en ejecución.

Patrones relacionados:

- Abstract Factory y Composite.

Referencias bibliográficas

[1] Design Patterns. Elements of Reusable Object-Oriented Software - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Addison Wesley (GoF- Gang of Four)

[2] Desing Patterns Java. James W Cooper. ebook – Addison Wesley.

[3] Patrones de Diseño, Diseño de Software Orientado a Objetos - Joaquin Garcia. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

Daza Corredor, Alejandro Paolo
Ingeniero de Sistemas, egresado de la Universidad Distrital Francisco José de Caldas, Docente TCO del proyecto curricular de Ingeniería de Sistemas en el área de programación y estudiante de la especialización de Ingeniería de Software de la Universidad Distrital Francisco José de Caldas.

e-mail: apdaza@gmail.com