MEMORIA AEC2 PATRONES DE DISEÑO

Recordemos que mi primera AEC, se basaba en crear una APP (ElementosGUI) para construir este tipo de elementos gráficos, siendo una expansión de unos de los ejemplos de la AA1 sobre patrones creacionales.

1.Distribución del proyecto en paquetes.

Primeramente, como hay bastantes clases que implementar, lo que hago es dividir el proyecto en cinco paquetes:

- **InterfacesGenerales:** contendrá tres interfaces:
    1. **CreadorElemGUI** que contendrá tres métodos: addElemGUI(), borrarElemGUI() y mostrarElemGUI() que serán implementados por las clases abstractas de objetos GUI (CreadorVentanas, CreadorCajasTexto y CreadorBotones) representando de esta forma el patrón **Abstract Factory.**
    2. **Agregado**: con un único método crearIterador() sin implementar y que será parte del patrón **Iterator**.
    3. **Iterador**: con cuatro métodos sin implementar: primero(), siguiente(), haTerminado() y elementoActual(). Representando también parte del patrón **Iterator**.

- **ClasesBotones**, que contendrá las siguientes clases (8):

    1. Interfaz **Boton**: que contendrá un único método mostrarBoton() sin implementar
    2. Clases **BotonCuadrado** y **BotonRedondo** que implementarán el

método de la interfaz concreta Boton, cada uno a su conveniencia.

3. Clase **CreadorBotones** la cual contendrá una colección de botones, un método que recorre los botones almacenados (getCantidadBotones) donde se puede ver el patrón **Composite**, varios métodos para borrar, añadir o mostrar botones y el método fábrica crearBoton() para representar el patrón **Factory Method**.
4. Clases **CreadorBotonCuadrado** y **CreadorBotonRedondo** las cuales extienden de la clase CreadorBotones e implementarán el método crearBoton() cada uno a su conveniencia.
5. Clase **AgregadoBotones** que representa la clase agregador concreto. Esta clase construirá un objeto iterador implementando el método de la interfaz Agregado que luego será cogido por la siguiente clase para recorrer las colecciones creadas de los elementos GUI construídos. Representará parte del patrón **Iterator.**
6. Clase **IteradorBotones** que contendrá varios métodos para recorrer una colección de botones. Representará parte del patrón **Iterator**.

- **ClasesVentanas** y **ClasesCajasTexto**: que contendrán las mismas clases (8 cada una) que ClasesBotones con la salvedad que cada uno creará sus objetos de forma diferente con los métodos de mostrarX() donde cada elemento será mostrado de forma diferente ya que son distintos.
- **Cliente**: con la clase Main para crear nuestros objetos, sus colecciones e iteraciones.

2.Diferencias con respecto al diseño inicial.

- En la interfaz CreadorElemGUI contendrá solo tres métodos addElemGUI, borrarElemGUI y mostrarElemGUI, mientras que crearElemGUI se quita, y se pone en cada clase abstracta (CreadorVentanas, CreadorBotones, CreadosCajasTexto) este método abstracto de forma personalizada (crearVentana(), crearBoton() y crearCajaTexto()) ya que como se devuelve en cada una un tipo de elemento distinto no se puede poner en la inferfaz, de ahí este cambio.
- También se añade a cada clase abstracta el método estático getCantidadVentanas(), getCantidadBotones() y getCantidadCajasTexto() los cuales contarán los objetos que hemos creado de cada tipo de elemento GUI para llevar la cuenta de estos. Ya que como no se pueden crear objetos de clases abstractas se crean estos métodos estáticos para que luego puedan ser probados en la clase Main y de esta forma probar el patrón **Composite**.
- Se crean tres clases de iteradores concretos, una para cada elemento GUI, en el anterior diseño solo había una clase de iterador concreto.

3.Reflexión personal.

Creo que escogiendo **Abstract Factory** y **Factory Method** se le da a la aplicación, si bien es cierto que se generan demasiadas clases, extensibilidad y bajo acoplamiento. Extensibilidad para poder introducir más elementos GUI si se desea (solo habría que seguir las interfaces). Bajo acoplamiento ya que si deseamos quitar un elemento GUI ya creado de la aplicación solo habría que borrar las clases de la misma familia de ese elemento sin tener que quitar ni modificar nada más del resto.

En cuanto al patrón **Composite** me ha parecido fácil ensamblar con los demás, ya que solo he tenido que introducir una colección para cada tipo de elemento y un método que la recorra para que cuente la cantidad de cada elemento GUI creado.

Luego, el patrón **Iterator** es el que más complejo me ha parecido de ensamblar con el resto, de hecho, no he podido probarlo en la clase Main porque me daba error, no sé si porque he creado más el diseño del patrón u otra cosa. Este patrón me ha generado dudas de como ensamblarlo con el resto. En un principio pensé que con un agregado concreto me bastaría, pero vi que había que crear un objeto iterador para cada tipo de elemento GUI. Hubo un momento que pensé en meter el método de creación de iteradores y los métodos de iteraciones concretos en las clases Creadoras, pero también me daba error a la hora de probarlo.

En términos generales, me ha parecido complicado hacer un diseño donde haya que unir varios patrones de distintos tipos, aunque se aprende bastante. No es lo mismo hacer ejemplos de un solo patrón como en las actividades de aprendizaje, que hacerlo en este tipo de prácticas donde debe verse cada patrón ensamblado con los demás.