**Información general del proyecto: para qué sirve, cuál es la estructura general del diseño, qué grandes retos de diseño enfrenta (i.e. ¿qué es lo difícil?). Deben incluir la URL para consultar el proyecto.**

El proyecto es un juego construido en Java, en el cual un elemento, que es el conejo de pascua, hace uso de la bolsa mágica de huevos para crear 3 tipos de huevos al azar. Cada que creaba un huevo, lo decora. Finalmente, se encuentra 3 observadores que revisan para que cuando el numero de huevos llegue a 5 en la canasta, avisar al usuario del juego.

La estructura general de diseño se puede decir que consiste en 2 componentes principales, el control sobre la creación y modificación de los huevos de pascua, y la información acerca de los huevos existentes, y su numero en la canasta. En este caso, el primer componente es el controlador del juego de estereotipo controller, ya que controla los huevos creados por el conejo y su decoración. Por lo tanto, el segundo componente es el encargado de la canasta, por lo que se puede llamar “Canasta” y se encarga de los observadores de la canasta, a su vez de llevar el control del numero de huevos asociado al elemento canasta (InfromationHolder). En el siguiente diagrama se puede apreciar un sencillo modelo del diseño explicado.

Imagen que contiene Diagrama

Descripción generada automáticamente

El mayor reto de diseño está en la granulación del mismo, ya que a pesar de ser un proyecto bastante sencillo, sus componentes y responsabilidades se encuentran altamente divididas y granuladas por lo que a pesar del alto acoplamiento, el diseño se puede complicar y puede haber colaboraciones innecesarias entre elementos y componentes.

**• Información y estructura del fragmento del proyecto donde aparece el patrón. No se limite únicamente a los elementos que hacen parte del patrón: para que tenga sentido su uso, probablemente va a tener que incluir elementos cercanos que sirvan para contextualizarlo.**

Loe elementos necesarios para entenderlo dentro del proyecto son la clase “EasterEggFactory” y las clases EasterEgg, QuailEgg, DinosaurEgg y ChickenEgg. En este caso, estas clases están estructuradas de la siguiente manera: EasterEgg es la clase padre de las otras tres clases de huevos, mientras que EasterEggFactory instancia los tres diferentes tipos de huevos existentes.

Diagrama

Descripción generada automáticamente

**• Información general sobre el patrón: qué patrón es y para qué se usa usualmente.**

El patrón para analizar es el patrón de creación “Factory Method”, el cual para solucionar el problema de crear objetos relacionados entre sí, hace que existan subclases que implementen la lógica concerniente a cada objeto. En la actualidad es usado por ejemplo para la creación de diferentes usuarios dentro de una aplicación, ya que permite diferenciar la lógica entre, por ejemplo, un administrador a un cliente, a pesar de que compartan las bases de un usuario.

**• Información del patrón aplicado al proyecto: explicar cómo se está utilizando el patrón dentro del proyecto.**

En este caso, el patrón se está usando mediante el hecho de que existe una “base” que define las bases de los huevos, en este caso “EasterEgg”. Sin embargo, la lógica como tal está en las subclases, en la clase padre solo se encuentra el método abstracto. Esto permite que en el momento en que la “Factory”, que en este caso es “EasterEggFactory” cree alguno de los objetos similares, estas sobrescriban el método de la clase padre y de esta manera creen objetos similares, pero con su propia lógica distinta.

**• ¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?**

La ventaja de haber utilizado este patrón radica en la capacidad de diferenciar la lógica para diferentes tipos de objetos, que comparten similares características. De esta manera, por ejemplo, al querer añadir un nuevo tipo de huevo, no hace falta modificar las clases, solo añadir otra clase que herede la clase padre, cumpliendo así los principios de la POO.

**• ¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?**

Las desventajas pueden radican en el hecho de que puede que se requiera que algunos objetos tengan diferentes características hasta el punto en el que difieran tanto, que pierda el sentido tener una clase padre ya que no comparten nada.

**• ¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?**

Puede que con el patrón AbstractFactory se pudiera hacer hecho distinto, ya que se mantendría todo dentro de una misma jerarquía al tener la creación de objetos y sus características en la misma jerarquía de herencia, lo cual podría haber sido útil.