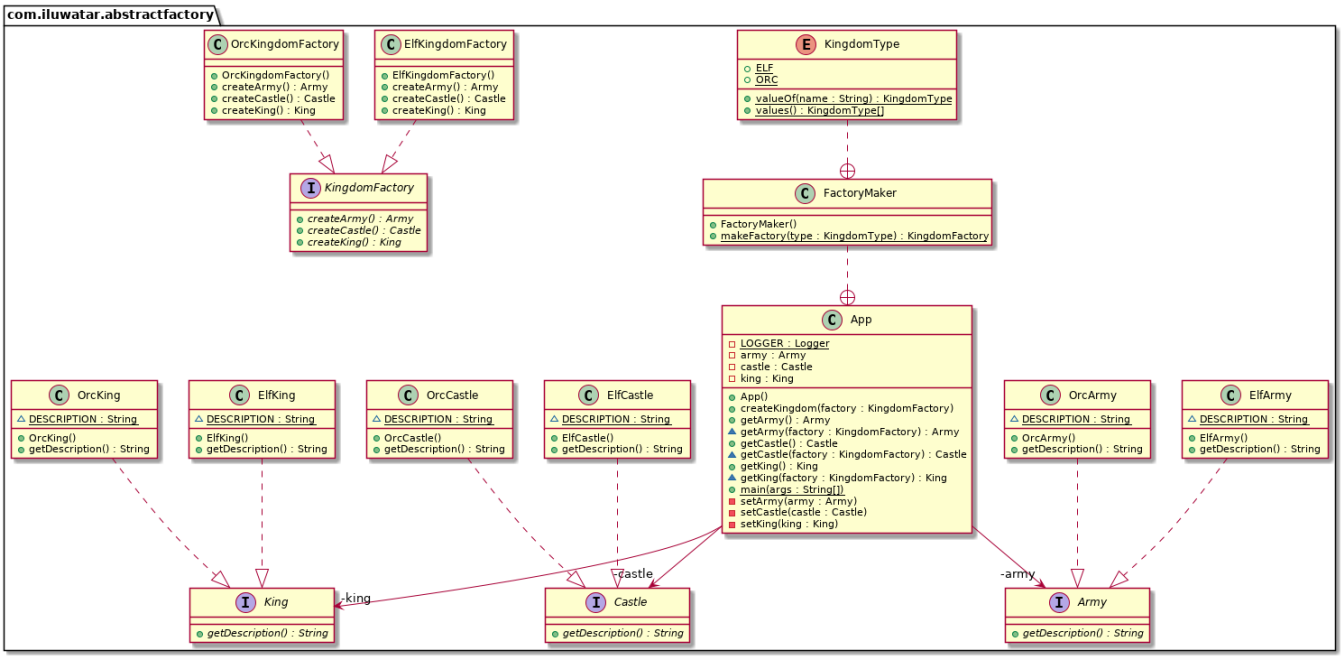
Alejandro Pardo Sánchez - 202223709

Patrón de diseño Abstract Factory

Este es un patrón de tipo creacional que permite generar objetos que son similares en estructura y pertenecen a la misma familia, pero son de distintos “padres” por lo que este patrón ayuda a que un objeto funcione con varias familias de objetos relacionados sin necesidad de especificar sus clases concretas.

Para este patrón encontré un programa que utiliza este patrón para conocer los objetos puntuales de reinos imaginarios, es decir, en el programa existen predeterminadamente 2 reinos; “ElfKingdom” y “OrcKingdom”, estos reinos contienen cada uno su castillo, su rey y su ejército. El patrón de Abstrac Factory es utilizado cuando otros reinos quieren ser agregados, para ello el usuario deberá usar el “Factory Maker” para crear un fabrica (reino) nueva que producirá objetos abstractos y concretos a ese reino, es decir que creará los objetos castillo, rey y ejercito para ese reino en particular.

El diagrama de clases fue realizado por los creadores del programa



En este diagrama de clases se pueden apreciar todas las conexiones que existen en el programa, para empezar se puede apreciar que “King”, “Castle” y “Army” son interfaces que heredan sus comportamientos a sus clases hijas que en este caso son las partes fundamentales del reino (rey, castillo y ejercito), con esto se logra apreciar que pertenecen a la misma familia sin ser de la misma

clase concreta, por lo que es posible tratar a los reyes, castillos y ejércitos de maneras distintas pero con las mismas características. La clase “KingdomType” (“Kingdom” en el programa) es aquella que puede crear una nueva fabrica para un nuevo reino, simplemente se agrega un nuevo reino como opción y se crea el rey, castillo y ejercito de un nuevo reino compartiendo las mismas características de los otros reinos ya creados anteriormente.

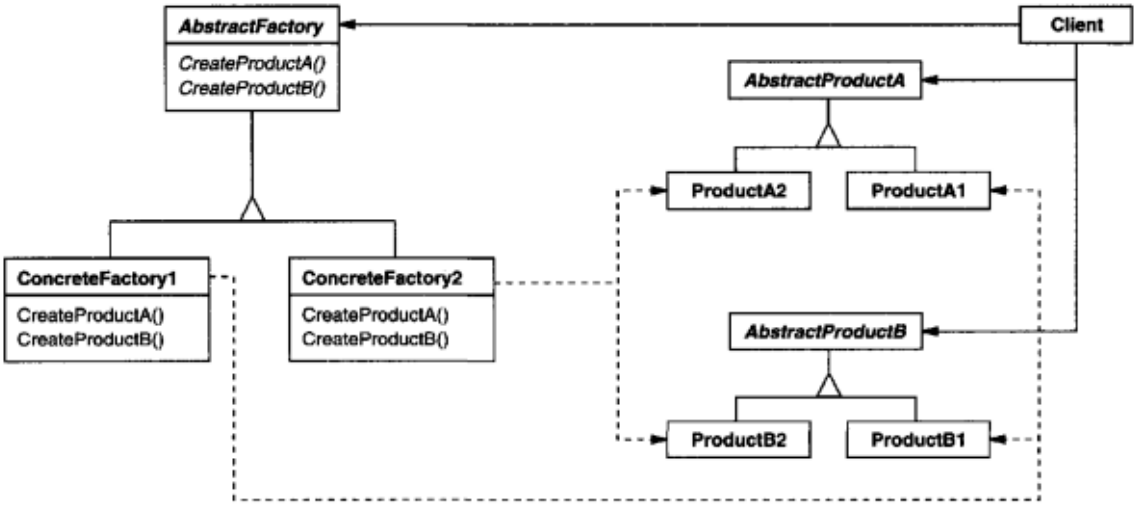
En el libro “Design Patterns Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides” se especifican 5 participantes cruciales dentro de este patrón: “AbstractFactory”, “ConcreteFactory”, “AbstractProduct”, “ConcreteProduct” y “Client”.

Continuando con las definiciones del libro para cada uno de los participantes del patrón es posible inferir que el participante “AbstractFactory” en el programa es “KingdomType” (“Kingdom” en el programa), pues aquí es donde se declaran las operaciones que crean los objetos de manera abstracta para cada reino. El participante “ConcreteFactory” es la clase “FactoryMaker”, esta clase hace alusión a las fabricas concretas que crean concretamente los objetos rey, castillo y ejercito para un reino, en el programa existen 2 fabricas ya realizadas, estas son “OrcKingdomFactory” y “ElfKingdomFactory” las cuales heredan a su vez los comportamientos de la interfaz “KingdomFactory”. El participante “AbstractProduct” son todas las interfaces que heredan comportamientos, en este ejemplo las interfaces “King”, “Castle” y “Army” son los productos abstractos que son las partes fundamentales de cada uno de los reinos. El participante “ConcreteProduct” ya son las clases concretas para cada reino, en este caso todas las clases que implementan “King”, “Castle” y “Army” son los productos o partes concretas de un reino. Por último, el participante “Client” quien es el que usa las interfaces declaradas por los participantes “AbstractFactory” y “AbtractProduct”, en este caso la clase “App” es aquella que implementa estas interfaces para conocer a los reinos e interactuar con ellos.

El libro contiene una explicación sobre las colaboraciones que puede que permita entender de mejor manera el patrón:

- Normally a single instance of a ConcreteFactory class is created at run-time. This concrete factory creates product objects having a particular implementation. Tocreate different product objects, clients should use a different concrete factory.
- AbstractFactory defers creation of product objects to its ConcreteFactory subclass. (Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, pp 89, 2009)

El libro también propone una estructura general que deben seguir todos los proyectos que implementen este tipo de patrón de diseño



(Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, pp 88, 2009)

En este diagrama de clases que propone el libro se puede apreciar como se conectan las clases e interfaces, donde es posible darse cuenta de la relación que tiene este diagrama con el diagrama del proyecto de los reinos, donde las fabricas concretas son las fabricas de cada reino, los productos abstractos son el rey, castillo y ejercito y los productos concretos son estes productos abstractos, pero para cada reino. Con esto, queda claro el uso de este patrón en este programa.

Es bastante útil utilizar este patrón para este problema del proyecto, pues los reinos tienen todas las mismas partes por lo que cada uno tiene rey, castillo y ejercito cada una de estas partes pertenece a una familia que comparte características, por lo tanto, ayuda a que un objeto funcione con varias familias de objetos relacionados sin necesidad de especificar sus clases concretas.

Por último, el link del repositorio Git del proyecto de los reinos es:
https://github.com/iluwatar/java-design-patterns/tree/master/abstract-factory