Taller 5
Mariana Ortega 202211233

El proyecto en cuestión implementa el patrón de diseño Abstract Factory para la creación de un reino (Kingdom) compuesto por componentes como Castle, King y Army. La finalidad principal es proporcionar una estructura flexible que permita la creación de familias de objetos relacionados sin especificar sus clases concretas. La organización del código sigue el principio de separación de responsabilidades, utilizando la interfaz KingdomFactory como la Abstract Factory principal, con implementaciones específicas como ElfKingdomFactory y OrcKingdomFactory. Estas implementaciones están destinadas a crear objetos pertenecientes a familias específicas, como Elfo u Orco.

El código está estructurado en paquetes que contienen clases específicas para cada componente del reino (Castle, King, Army), y el paquete com.iluwatar.abstractfactory alberga las interfaces base y la clase principal Kingdom. Los retos de diseño incluyen la necesidad de mantener flexibilidad y extensibilidad para la adición de nuevos tipos de reinos, garantizar la coherencia de objetos pertenecientes a una misma familia, y mantener una estructura de código clara y fácilmente mantenible con la introducción de nuevas funcionalidades.

El patrón Abstract Factory según la página Web Refactoring Guru, se utiliza cuando se desea que el "código deba funcionar con varias familias de productos relacionados, pero no desees que dependa de las clases concretas de esos productos, ya que puede ser que no los conozcas de antemano o sencillamente quieras permitir una futura extensibilidad" (Refactoring Guru, sf). Lo anterior quiere en pocas palabras que evitamos un acoplamiento tan fuerte, esto nos ayuda a que en un futuro se puede ampliar incluso las familias de objetos. No debemos preocuparnos por la conexión entre el código "base" y las implementaciones más específicas. En este caso lo veremos con Elf y Orc.

El patrón Abstract Factory se aplica en el proyecto para encapsular la creación de objetos de reino (Elf o Orc). El método createKingdom en la clase App toma un parámetro KingdomType y utiliza la Abstract Factory correspondiente para crear un conjunto coherente de objetos de reino (Castle, King, Army). Esto permite que la aplicación cree y utilice diferentes tipos de reinos sin acoplar el código directamente a las implementaciones concretas.

Entre las ventajas de utilizar el patrón se encuentran la flexibilidad para agregar nuevos tipos de reinos sin modificar el código existente, la garantía de coherencia entre objetos creados y el desacoplamiento del código cliente de las clases concretas. Sin embargo, se debe tener en cuenta la introducción de complejidad adicional y un posible aumento en el número de clases e interfaces. Alternativamente, se podrían haber abordado los desafíos mediante un constructor directo, configuración externa o incluso el patrón Builder, aunque este último podría haber introducido complejidad innecesaria para este caso específico. Por otro lado, otra manera de resolverlo, tal como plantea la página web anteriormente mencionada, Refactoring Guru, se puede utilizar adicionar a la lista el patrón Bridge, ya que "este emparejamiento resulta útil cuando algunas abstracciones definidas por Bridge sólo pueden funcionar con implementaciones específicas

Abstract Factory. (s/f). Refactoring.guru. de https://refactoring.guru/es/design-patterns/abstract-factory

Link: https://github.com/iluwatar/java-design-patterns/tree/master/abstract-factory/src/main/java/com/iluwatar/abstractfactory