Patrones de Diseño: Factory Method

Programación de Aplicaciones Móviles Nativas

Autor: Fernando Sanfiel Reyes

Fecha: 23/10/2023

Índice

Introducción 3
¿Qué pasa si en un futuro se quisiera añadir un nuevo tipo de notas? 6
¿Qué partes de tu aplicación tendrías que modificar? 6
¿Qué nuevas clases tendrías que añadir? 6
Repositorio de GitHub 7
Bibliografía 7

Introducción

Los patrones de diseño son esenciales en el desarrollo de software, Estos patrones ofrecen soluciones probadas a problemas comunes, permitiendo crear aplicaciones más modulares, mantenibles y escalables. Se nos pide diseñar una estructura de clases para una aplicación de notas en *Kotlin* utilizando el patrón *Factory Method*. Las notas pueden ser de tipo texto, imagen y audio.

Diagrama de clases

Como vemos en la Ilustración 1 tenemos el diagrama de clases donde tenemos la interfaz *Note*, con los atributos y métodos generales a las notas. También la clase *NoteFactory* que nos permite crear y obtener instancias de todos los tipos de notas posibles. Y los tipos diferentes de nota implementan la interfaz Nota y heredan de la clase *NoteFactory*.

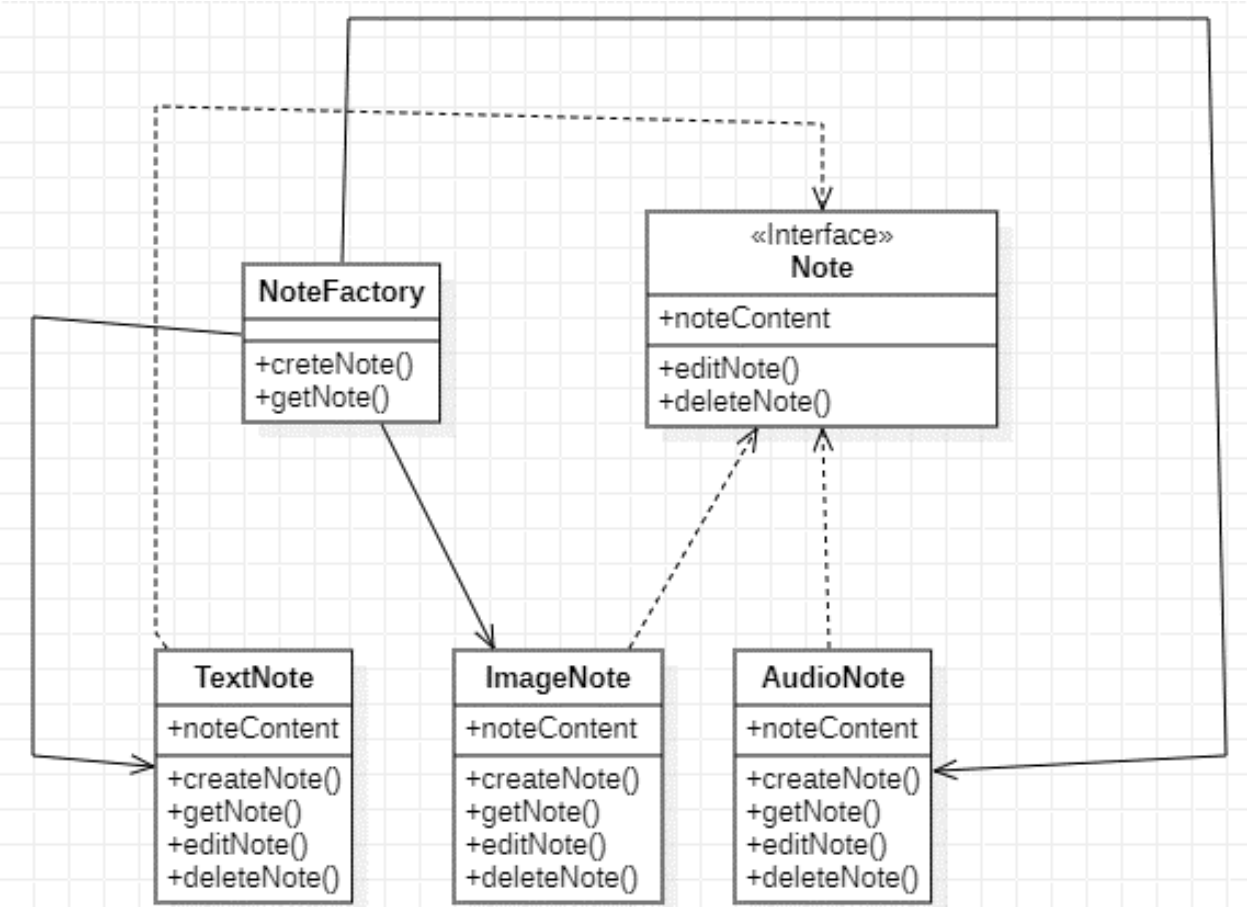

Ilustración 1: Diagrama de clases

Código de las clases

En las siguientes imágenes podemos ver como podrían ser el código genérico de la clase *NoteFactory*(Ilustración 2), la de la interfaz *Note*(Ilustración 3) y también se ve el código de la clase *TextNote*(Ilustración 4) como ejemplo de una de las clases específicas, ya que la de *ImageNote* y *AudioNote* serían bastante similares.

```
class NoteFactory {
    fun createNote(): Note? =
        when {
            isTextNote(noteId: String) -> return new TextNote()
            isImageNote(noteId: String) -> return new ImageNote()
            isAudioNote(noteId: String) -> return new AudioNote()
            else -> throw IllegalStateException
        }

    fun getNote(noteId: String): Note? =
        when {
            isTextNote(noteId: String) -> TextNote(noteId)
            isImageNote(noteId: String) -> ImageNote(noteId)
            isAudioNote(noteId: String) -> AudioNote(noteId)
            else -> throw IllegalStateException
        }
    private fun isTextNote(noteId: String) = true
    private fun isImageNote(noteId: String) = true
    private fun isAudioNote(noteId: String) = true
}
```

Ilustración 2: Código de la clase NoteFactory

```
interface Note {
    val id: Int
    val noteContent: String
    fun editNote(noteId: String): Boolean
    fun deleteNote(noteId: String): Boolean
}
```

Ilustración 3: Código de la interfaz Note

```
class TextNote(id: String): Note {
    fun createNote(): TextNote = return new TextNote()

    fun getNote(noteId: String): Note? {
        if (this.id == noteId) {
            return this
        } else {
            return null
        }
    }

    fun editNote(noteId: String): Boolean {
        if (this.id == noteId) {
            //Se edita la nota
            return true
        } else {
            return false
        }
    }

    fun deleteNote(noteId: String): Boolean {
        if (this.id == noteId) {
            //Se elimina la nota
            return true
        } else {
            return false
        }
    }

    override private fun isTextNote(noteId: String) = true
}
```

Ilustración 4: Código de la clase TextNote

Breve descripción de cada clase

La clase *NoteFactory* nos permite crear y obtener instancias de cada tipo de nota, pero a su vez nos permite tratar los diferentes tipos de *Nota* de forma genérica. La interfaz Note nos permite asegurarnos que los atributos y métodos que son comunes a todos los tipos de notas estén implementados de forma "forzada". En cada una de las clases especificas tenemos la implementación de la clase padre *NoteFactory* y además la implementación de la Interfaz *Note* generalizando así el comportamiento de estas.

Reflexión de futuros cambios

¿Qué pasa si en un futuro se quisiera añadir un nuevo tipo de notas?

Sería bastante sencillo ya que sería tan simple como crear la clase para el nuevo tipo de nota y que esta implemente tanto la interfaz *Note* como la clase *NoteFactory*.

¿Qué partes de tu aplicación tendrías que modificar?

En la aplicación principal solo habría que añadir las partes de código necesarias para el tratamiento especifico de la nueva clase de notas y los demás no es necesario especificarlo.

¿Qué nuevas clases tendrías que añadir?

Tan solo sería necesario añadir la clase correspondiente al nuevo tipo de nota que se quiere implementar.

Repositorio de GitHub

https://github.com/reyesanfer/Patrones_Diseno-Factory_Method/tree/main

Bibliografía

Method, F. (23 de 10 de 2023). *refactoring guru*. Obtenido de refactoring guru: https://refactoring.guru/es/design-patterns/factory-method