ULPGC

Escuela de Ingeniería Informática

Patrones de Diseño: Factory Method

40983 – Programación de Aplicaciones Móviles Nativas

Pablo Santana Susilla

Pr. Laboratorio 01.01

Profesor adjunto: Antonio Iván Hernández Fragiel

Curso: 2023/202

Introducción

Los patrones de diseño son esenciales en el desarrollo de software, y en el ámbito de las aplicaciones móviles no es la excepción. Estos patrones ofrecen soluciones probadas a problemas comunes, permitiendo crear aplicaciones más modulares, mantenibles y escalables. Al entender y aplicar estos patrones, los desarrolladores pueden mejorar la calidad y eficiencia de sus aplicaciones móviles.

1. **Diagrama de clases con la estructura del patrón de diseño aplicada a la aplicación de notas.**

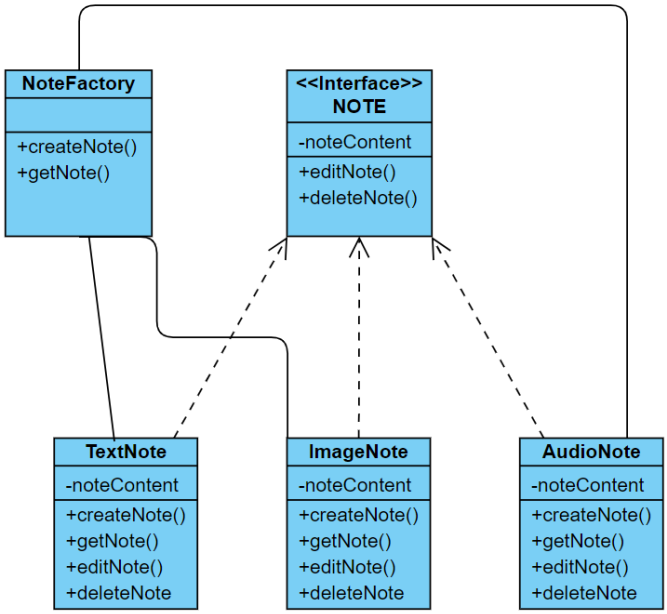

Figura 1. Diagrama de clases

2. **Código de las clases sin entrar en detalles de implementación, en la presentación de clases puedes seguir el nivel de detalle del ejemplo del validador de documentos**

```
// Interfaz para las notas
interface Note {
    fun createNote()
    fun getNote()
    fun editNote()
    fun deleteNote()
}

// Clase para notas de texto
class TextNote : Note {
    override fun createNote() {
        // Implementación
    }

    override fun getNote() {
        // Implementación
    }

    override fun editNote() {
        // Implementación
    }

    override fun deleteNote() {
        // Implementación
    }
}
```

```kotlin
// Clase para notas de imagen
class ImageNote : Note {
    override fun createNote() {
        // Implementación
    }

    override fun getNote() {
        // Implementación
    }

    override fun editNote() {
        // Implementación
    }

    override fun deleteNote() {
        // Implementación
    }
}

// Clase para notas de audio
class AudioNote : Note {
    override fun createNote() {
        // Implementación
    }

    override fun getNote() {
        // Implementación
    }

    override fun editNote() {
        // Implementación
    }

    override fun deleteNote() {
        // Implementación
    }
}

```

```kotlin
interface NoteFactory {
    fun createNote(): Note
}

// Clases para crear instancias
class TextNoteFactory : NoteFactory {
    override fun createNote(): Note {
        return TextNote()
    }
}

class ImageNoteFactory : NoteFactory {
    override fun createNote(): Note {
        return ImageNote()
    }
}

class AudioNoteFactory : NoteFactory {
    override fun createNote(): Note {
        return AudioNote()
    }
}


```

Figura 2. Código de las distintas clases

3. **Breve Descripción de Cada Clase:**
   - ***Note:*** Es una interfaz que define las operaciones básicas que una nota puede realizar (crear, obtener, editar y eliminar).
   - ***TextNote, ImageNote, AudioNote*****:** Son clases concretas que implementan la interfaz Note y proporcionan implementaciones específicas para cada tipo de nota.
   - ***NoteFactory:*** Es una interfaz que define el método para crear una instancia de Note.
   - ***TextNoteFactory, ImageNoteFactory, AudioNoteFactory:*** Son clases concretas que implementan la interfaz ***NoteFactory*** y crean instancias de las notas correspondientes.

4. **Reflexiona sobre cambios futuros**

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

Si en un futuro se quisiera añadir un nuevo tipo de nota, como, por ejemplo, una nota de vídeo sería bastante sencillo de implementar gracias al método de diseño elegido. Simplemente se debería de crear una nueva clase concreta para la nota de vídeo.

- ¿Qué partes de tu aplicación tendrías que modificar?

Para llevar a cabo estas modificaciones, se debería de hacer lo siguiente:

- Modificar la Clase Factory (NoteFactory): Se añadiría un nuevo método estático en NoteFactory para crear instancias de la nueva clase de nota de video.
- Modificar el Diagrama de Clases: Se añadiría una nueva clase para representar la nota de video en el diagrama de clases.

- ¿Qué nuevas clases tendrías que añadir?

Se debería de añadir una nueva clase para el nuevo tipo de nota y un nuevo método en la clase '*NoteFactory'* para crear instancias de la nueva clase creada.

Conclusión

El empleo del patrón *Factory Method* hace que la incorporación de un nuevo tipo de nota sea altamente modular y no demande modificaciones significativas en el código preexistente. Esto simplifica la capacidad de expandir y mantener el sistema a medida que se introducen nuevas funcionalidades.

Enlace a Github

https://github.com/pablosanttanaa/PAMN.git