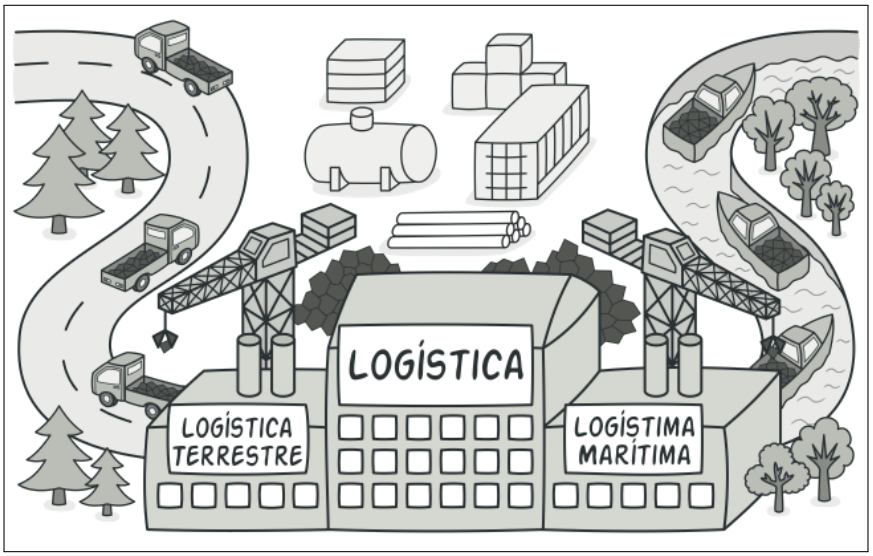
ULPGC · UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Práctica 8

Patrones de Diseño



Programación de Aplicaciones Móviles Nativas
05 de Noviembre de 2023

Autores:

Ana del Carmen Santana Ojeda (ana.santana152@alu.ulpgc.es)

Alejandro David Arzola Saavedra (alejandro.arzola101@alu.ulpgc.es)

Índice

1. Introducción

Los patrones de diseño sirven como herramientas esencial para enfrentar desafíos recurrentes y **alcanzar soluciones robustas**. Este principio no es una excepción en el mundo de las aplicaciones móviles, donde la **modularidad**, la **mantenibilidad** y la **escalabilidad** son imperativos para el éxito del desarrollo.

Reflexionaremos sobre cómo este diseño puede evolucionar en respuesta a la adición de **nuevos tipos de notas**, explorando las áreas de la aplicación que podrían necesitar ajustes y las nuevas clases que podrían surgir.

2. Desarrollo

2.1. Diagrama de clases

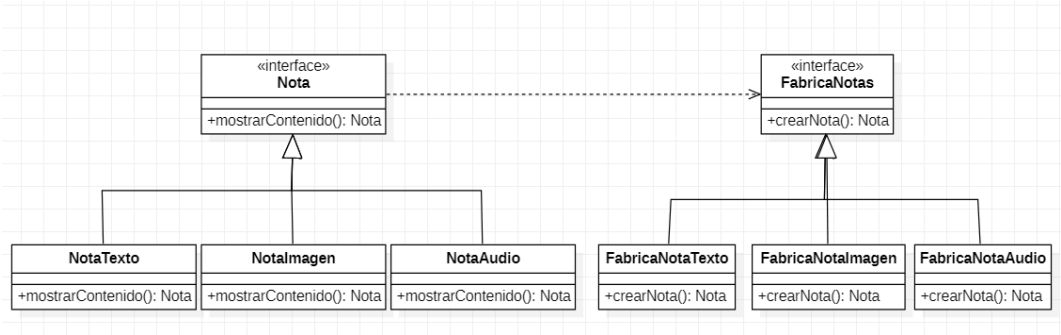


Figura 1: Diagrama de Clase

En el siguiente apartado, mostraremos el diagrama de clases de notas. En el diagrama, podemos observar dos interfaces, '**Nota**' y '**Fabrica Notas**', con una dependencia hacia '**Nota**'. '**Nota**' tiene tres clases que **heredan** de ella: '**Nota Texto**', '**Nota Imagen**' y '**Nota Audio**', las cuales deben implementar la función de mostrar contenido. De manera similar, '**Fabrica Notas**' tiene tres clases que heredan de ella: '**Fabrica Nota Texto**', '**Fabrica Nota Imagen**' y '**Fabrica Nota Audio**', las cuales deben implementar la **creación de notas**.

2.2. Código y descripción de las clases

```
1  package com.example.factory.pattern
2
3  // Interfaz para la fábrica de notas
4  interface FabricaNotas {
5      fun crearNota(): Nota
```

Figura 2: Interfaz Fabrica de Notas

Interfaz FabricaNotas

Esta interfaz define una **fábrica para crear instancias de la interfaz Nota**. Proporciona un método **crearNota()** que devuelve un **objeto Nota**. Este método es implementado de manera diferente en cada fábrica concreta, permitiendo la **creación de diferentes tipos de notas**.

```kotlin
package com.example.factory.pattern

// Implementación concreta de la fábrica para notas de texto
class FabricaNotaTexto(private val contenido: String) : FabricaNotas {
    override fun crearNota(): Nota {
        return NotaTexto(contenido)
    }
}

// Implementación concreta de la fábrica para notas de imagen
class FabricaNotaImagen(private val rutaImagen: String) : FabricaNotas {
    override fun crearNota(): Nota {
        return NotaImagen(rutaImagen)
    }
}

// Implementación concreta de la fábrica para notas de audio
class FabricaNotaAudio(private val rutaAudio: String) : FabricaNotas {
    override fun crearNota(): Nota {
        return NotaAudio(rutaAudio)
    }
}
```

Figura 3: Implementacion de la fabrica de Notas

Clase FabricaNotaTexto, FabricaNotaImagen y FabricaNotaAudio

Estas clases son una **implementación concreta de la interfaz FabricaNotas**. Crea instancias de la clase **NotaTexto, NotaImagen y NotaAudio**. Cada una proporciona una implementación específica del **método crearNota()**. Esto abstrae la lógica de creación de objetos, permitiendo que **la lógica del cliente dependa de la interfaz** FabricaNotas **en lugar de las implementaciones concretas**.

```kotlin
package com.example.factoryPattern

interface Nota {
    fun mostrarContenido()
}
```

Figura 4: Interfaz Nota

Interfaz Nota

Esta interfaz declara una función **mostrarContenido()**, que es implementada por las diversas clases de **Nota**. Se define la **interfaz de los productos que serán creados por**

las fábricas. En este caso, **la interfaz Nota tiene el método mostrarContenido()**.

```kotlin
package com.example.factorypattern

class NotaTexto(contenido: String?): Nota {

    override fun mostrarContenido() = println("Nota de texto: $contenido")
}

package com.example.factorypattern

class NotaImagen(rutaImagen: String?): Nota {

    override fun mostrarContenido() = println("Nota de imagen: $rutaImagen")
}

package com.example.factorypattern

class NotaAudio(rutaAudio: String?): Nota {

    override fun mostrarContenido() = println("Nota de audio: $rutaAudio")
}
```

Figura 5: Implementacion de las Notas

Implementacion NotaImagen, NotaAudio y NotaTexto

Una clase que **implementa la interfaz Nota**. Representa una nota de texto, audio e imagen.**Implementan la interfaz Nota y proporcionan la implementación específica para cada tipo de nota**.

```kotlin
fun main() {
    // Crear fábricas de notas
    val fabricaTexto: FabricaNotas = FabricaNotaTexto("Hola mundo!")
    val fabricaImagen: FabricaNotas = FabricaNotaImagen("ruta/directorio/imagen_ejemplo.jpg")
    val fabricaAudio: FabricaNotas = FabricaNotaAudio("ruta/directorio/audio_ejemplo.mp3")

    // Crear notas utilizando las fábricas

    val notaImagen: Nota = fabricaImagen.crearNota()
    val notaAudio: Nota = fabricaAudio.crearNota()

    // Mostrar el contenido de las notas
    notaTexto.mostrarContenido()
    notaImagen.mostrarContenido()
    notaAudio.mostrarContenido()
}
```

Figura 6: Codigo Main

Funcion Main

En la función main, se puede ver cómo se utilizan las fábricas para crear diferentes tipos de notas **sin preocuparse por la lógica específica de cada tipo**. Esto **simplifica el código del cliente**.

3. Reflexion sobre cambios futuros

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

Si necesitas agregar un nuevo tipo de nota, simplemente **creas una nueva clase que implementa la interfaz Nota y su respectiva fábrica**.

3.2. ¿Qué partes de tu aplicación tendrías que modificar?

No necesitas modificar el código existente, lo que facilita la extensión del sistema.

3.3. ¿Qué nuevas clases tendrías que añadir?

Habria que añadir 2 clases, una que forma parte de los tipos de notas, y la fabrica de la nota.

Por ejemplo **si quiseramos añadir una nota de video** habria que añadir:

```
1  package com.example.factory.pattern
2
3  class FabricaNotaVideo(private val rutaVideo: String) : FabricaNotas {
4      override fun crearNota(): Nota {
5          return NotaVideo(rutaVideo)
6      }
7  }
```

Figura 7: Implementacion de la fabrica notas

```
1  package com.example.factorypattern
2
3  class NotaVideo(private val rutaVideo: String?) : Nota {
4      override fun mostrarContenido() = println("Nota de video: $rutaVideo")
5  }
```

Figura 8: Implementacion de las Notas

```
1  fun main() {
2      // Crear fábrica y nota de video
3      val fabricaVideo: FabricaNotas = FabricaNotaVideo("ruta/video.mp4")
4      val notaVideo: Nota = fabricaVideo.crearNota()
5
6      // Mostrar el contenido de la nota de video
7      notaVideo.mostrarContenido()
8  }
```

Figura 9: Main del video

4. Conlusion

En esta práctica, aplicamos patrones de diseño, como la **Fábrica Abstracta y el Patrón de Método Factory**, para abordar la **creación de diversos tipos de notas en una aplicación móvil**. Este enfoque modular y escalable **permite la adición de nuevas funcionalidades sin alterar el código existente**, promoviendo la **mantenibilidad y la flexibilidad del sistema**.