Patrones de Diseño: Factory Method

PAMN - Programación de Aplicaciónes Moviles Nativas

Chamil José Cruz Razeq

9 de diciembre de 2023

1. Introducción

Todas las tareas propuestas se encuentran disponibles en el siguiente repositorio de GitHub.

Para esta resolución se ha diseñado una estructura de clases para una aplicación de notas en Kotlin utilizando el patrón Factory Method [3].

2. Diagrama de Clases

Para el diseño del programa se ha utilizado como referencia el diagrama [Figura 1].

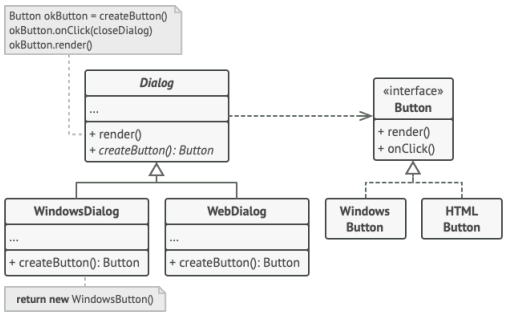

Figura 1: Diagrama de clases de aplicación de ejemplo de www.refactoring.com [3]

Se ha replicado la estructura, siguiendo las funcionalidades propuestas en el enunciado, dando como resultado al diagrama [Figura 2]

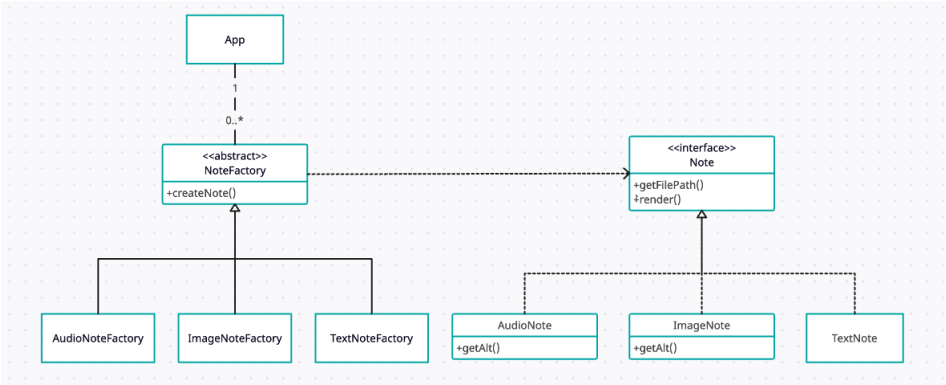

Figura 2: Diagrama de clases de aplicación de notas

3. Código de Prueba

Se ha elaborado un pequeño programa de prueba siguiendo la estructura planteada en elaborado diagrama de clases. Para ello se ha utilizado el lenguaje de programación "Kotlinz para ejecutarlo el laboratorio de pruebas online ofrecido por "Google".

Comenzamos con la interfaz "Note"que servirá como abstracción de todos los tipos de notas de nuestro programa:

```
36  interface Note {
37      /** getFilePath
38       *
39       *  Retrieves the path of the file related to the current object.
40       *
41       *  @return String with the path of the file.
42       */
43      fun getFilePath(): String?;
44
45      /** render
46       *
47       *  Renders a visually dintiguishable note.
48       */
49      fun render();
50  }
```

Figura 3: Interfaz de nota

Seguidamente se implementó utilizando la interfaz anterior los tres tipos de notas, desarrollando asimísmo sus peculiaridades, tales como "getAlt"mas afín a elementos audiovisuales:

```
37  class ImageNote : Note {
38      // [!] To be implemented
39      override fun getFilePath() : String? = "\\image.jpg";
40
41      // [!] To be implemented
42      override fun render() = println("I am an image note.");
43
44      // [!] To be implemented
45      fun getAlt(): String = "Description of the image.";
46  }
```

Figura 4: Implementación de nota de imagen

```
36  class AudioNote : Note {
37      // [!] To be implemented
38      override fun getFilePath() : String? = "\\audio.mp3";
39
40      // [!] To be implemented
41      override fun render() = println("I am an audio note.");
42
43      // [!] To be implemented
44      fun getAlt(): String = "Transcription of the audio.";
45  }
```

Figura 5: Implementación de nota de audio

```
38  class TextNote : Note {
39      // [!] To be implemented
40      override fun getFilePath() : String? = "\\text.txt";
41
42      // [!] To be implemented
43      override fun render() = println("I am a text note.");
44  }
```

Figura 6: Implementación de nota de texto

Por otro lado se creó la clase abstracta de "producción", que servirá como abstracción de las clases productoras respectivas a cada clase creada anteriormente:

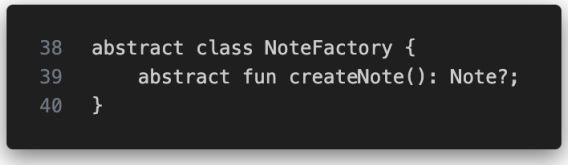

Figura 7: Clase abstracta de productor de notas

Como era de esperar, se desarrolló una clase productora por cada clase "Nota", cumpliendose así finalmente el patrón "Factory Method":

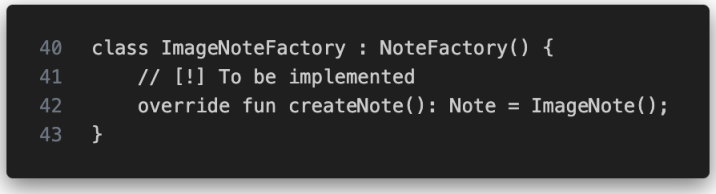

Figura 8: Clase productora de notas de imagen

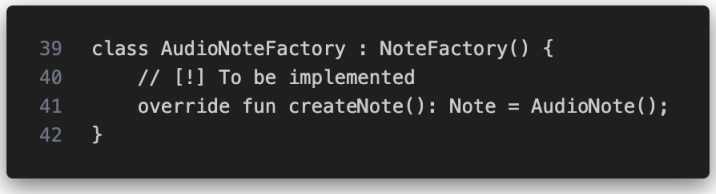

Figura 9: Clase productora de notas de audio

```
39  class TextNoteFactory : NoteFactory() {
40      // [!] To be implemented
41      override fun createNote(): Note = TextNote();
42  }
```

Figura 10: Clase productora de notas de texto

4. Reflexión

Este patrón resulta útil ante la incertidumbre de la escalabilidad del código.

Por un lado, al abstraer las características comunes de las "Notas"podemos garantizar la facilidad de incrementar el número de tipo de notas, además de promover el polimorfismo.

Cabe destacar que, abstrayendo la producción conseguimos el mismo efecto anteriormente mencionado, sumando ciertas ventajas en cuanto al control y la escalabilidad de la producción.

En resumidas cuentas, esta estructura facilita el despliegue de nuevas características con la sencilla intervención de crear una nueva clase productora y tipo, siguiendo respectivamente lo estipulado por la clase abstracta y interfáz.

Referencias

[1] PAMN Lab3 Arquitectura

[2] Rooms - https://medium.com/dvt-engineering/android-room-versus-sqlite-which-is-best-32ff651bc361

[3] Factory Method - https://refactoring.guru/es/design-patterns/factory-method