Universidad Tecnológica del Norte de Guanajuato
Organismo Público Descentralizado del Gobierno del Estado
"Educación y progreso para la vida"

ARQUITECTURA DE SOFTWARE

III. Diseño de Patrones

Actividad 2 – Saber Hacer

Integrantes:

Alvarez Ramírez Francisco – 1220100158

Godínez Morales Martin Gabriel – 1221100313

Alan Manuel Mendoza Arredondo –1221100341

GIDS4071

UTNG Dolores Hidalgo C.I.N GTO 07/12/23

ÍNDICE

Introducción

En el dinámico y cambiante panorama del desarrollo de software, la creación eficiente y flexible de objetos emerge como un pilar fundamental para asegurar la escalabilidad y la adaptabilidad de las aplicaciones. La capacidad de diseñar sistemas que puedan evolucionar y adaptarse a medida que los requisitos cambian es esencial. Entre los desafíos más cruciales que los ingenieros de software enfrentan, se destaca la instancia de objetos, una tarea que se vuelve aún más crítica cuando se trata de informes. La necesidad imperante de una solución que no solo facilite la creación de instancias de objetos informe, sino que permita la incorporación sin inconvenientes de nuevos tipos de informes, sin perturbar el ya existente código, se convierte en un imperativo para el éxito a largo plazo de cualquier aplicación. Es en este contexto que el patrón de diseño Factory Method se presenta como una estrategia clave.

El Factory Method no es simplemente una herramienta más en el repertorio de los desarrolladores; es una elección estratégica. Se selecciona específicamente para abordar el desafío de la creación de instancias de objetos informe. Mientras que las aproximaciones convencionales podrían depender de una única implementación de creación de informes en el componente, el Factory Method ofrece una solución más sofisticada. Este patrón permite encapsular la lógica de creación en una fábrica, proporcionando al sistema la capacidad única de incorporar nuevos tipos de informes en el futuro sin necesidad de alterar el código existente.

La verdadera fortaleza del Factory Method radica en su capacidad para desacoplar la lógica de creación de informes del componente que los utiliza. Este desacoplamiento no es simplemente una separación de responsabilidades; es una completa independencia. La manera en que se crean los informes queda aislada y apartada del código del cliente que los consume. Esta característica es crucial, ya que no solo facilita una fácil sustitución de implementaciones sin afectar el código del cliente, sino que también allana el camino para un mantenimiento más sencillo y evolución del sistema a medida que las necesidades cambian.

Al abordar la creación de instancias de objetos informe, el Factory Method proporciona una estructura que no solo promueve el desacoplamiento y la sustitución sencilla de implementaciones, sino que también abre las puertas para la incorporación sin complicaciones de nuevos tipos de informes en el futuro. La flexibilidad y extensibilidad inherentes a este patrón hacen que sea una elección estratégica para proyectos que aspiran a adaptarse de manera dinámica a cambios en los requisitos y evolucionar con el tiempo.

En este documento, no solo exploraremos la justificación detrás de la elección del Patrón Factory Method para la creación de instancias de objetos informe, sino que también desglosaremos los beneficios asociados a su implementación en detalle. Además, proporcionaremos una descripción exhaustiva de cada elemento involucrado en el proceso, desde la interfaz InformeFactory hasta el uso en el componente Angular, destacando la importancia de la configuración del módulo y la ilustración mediante un diagrama UML. Estos elementos se presentarán de manera integrada, mostrando cómo cada uno contribuye colectivamente a la construcción de sistemas robustos y fácilmente mantenibles, y proporcionando ejemplos concretos para una comprensión más profunda.

Justificación del Patrón Factory Method:

El patrón Factory Method se eligió para abordar la creación de instancias de objetos Informe de manera flexible y extensible. En lugar de tener una única implementación de la creación de informes en el componente, el patrón Factory Method permite encapsular esta lógica en una fábrica, facilitando la incorporación de nuevos tipos de informes en el futuro sin modificar el código existente.

- **Beneficios**:
  - **Desacoplamiento**: La lógica de creación de informes está desacoplada del componente que los utiliza, permitiendo una fácil sustitución de implementaciones sin afectar el código del cliente.
  - **Extensibilidad**: Al introducir nuevas clases de fábrica concretas, se pueden agregar fácilmente nuevos tipos de informes sin afectar el código existente.

Justificación de Cada Elemento:

- **Interfaz InformeFactory:**
  - Define el contrato para las fábricas que crean instancias de Informe.
  - Facilita la incorporación de nuevas fábricas.

```typescript
// informe-factory.interface.ts
import { Informe } from "./informe.interface";

export interface InformeFactory {
    crearInforme(): Informe;
}
```

Ilustración 1 interfaz informeFactory

- **Fábrica Concreta InformePDFFactory:**
  - Implementa InformeFactory y proporciona una implementación concreta de crearInforme, devolviendo una instancia de InformePDF.
  - Permite la creación de informes en formato PDF.

```typescript
// informe-pdf.factory.ts
import { Injectable } from '@angular/core';
import { InformeFactory } from './informe-factory.interface';
import { InformePDF } from './informe-pdf';
import { Informe } from './informe.interface';
import { ApiService } from 'src/app/api.service';

@Injectable({ providedIn: 'root' })
export class InformePDFFactory implements InformeFactory {
  constructor(private apiService: ApiService) {}

  crearInforme(): Informe {
    return new InformePDF(this.apiService);
  }
}
```

Ilustración 2 Fabrica concreta informePDFFactory

- **Producto Abstracto e Implementación Concreta InformePDF:**
  - Informe define el contrato para los diferentes tipos de informes.
  - InformePDF implementa ese contrato y proporciona la lógica para generar informes en formato PDF.

```
export class InformePDF implements Informe {
  constructor(private apiService: ApiService) {}

  async generarInforme(): Promise<void> {
    pdfMake.vfs = pdfFonts.pdfMake.vfs;

    // Obtener datos de la base de datos a través del servicio de API
    const proyectos: Proyecto[] = await this.apiService.loadProyectos().toPromise();

    // Transformar datos en contenido de informe
    const content = [
      { text: 'Informe en formato PDF', style: 'header' },
      { text: 'Datos de Proyectos', style: 'subheader' },
      this.createTable(proyectos),
    ];

    // Definición del documento PDF
    const documentDefinition = {
      content,
      styles: {
        header: {
          fontSize: 18,
          bold: true,
          margin: [0, 0, 0, 10],
        },
        subheader: {
          fontSize: 14,
          bold: true,
          margin: [0, 10, 0, 5],
        },
      },
    };

    // Generar el PDF
    pdfMake.createPdf(documentDefinition).open();
  }

  private createTable(data: Proyecto[]): any {
    // Definir encabezados de la tabla
    const headers = ['ID Proyecto', 'Nombre', 'Estado'];

    // Mapear datos para la tabla
    const body = data.map((proyecto) => [proyecto.idProyecto, proyecto.nombreCorto, proyecto.estadoProyecto]);

    // Agregar encabezados y datos a la tabla
    return {
      table: {
        headerRows: 1,
        widths: ['auto', '*', 'auto'],
        body: [headers, ...body],
      },
    };
  }
}
```

Ilustración 3 Implementación concreta InformePDF

- **Configuración del Módulo (app.module.ts):**
  - Configura la inyección de dependencias para que Angular pueda proporcionar la implementación concreta de InformeFactory.
- **Uso en el Componente Angular (dashboard.component.ts):**
  - Utiliza la interfaz InformeFactory para crear instancias de Informe sin preocuparse por la implementación específica.
  - Facilita la extensibilidad y evita dependencias directas de las clases concretas.

```
generarInforme(): void {
  const informe = this.informeFactory.crearInforme();
  informe.generarInforme();
}
```

Ilustración 4 uso en el componente

Diagrama UML

- **Clase Informe:**
  - La clase Informe es una interfaz o clase abstracta que define el método generarInforme(). Este método representa la operación que será implementada por las clases concretas, pero la interfaz Informe no especifica cómo se implementa.
- **Clase InformePDF**:
  - La clase InformePDF implementa la interfaz Informe, proporcionando una implementación concreta para el método generarInforme(). Aquí es donde se define la lógica específica para generar un informe en formato PDF.
- **Clase InformeFactory:**
  - La clase InformeFactory es una interfaz o clase abstracta que declara el método crearInforme(). Esta interfaz sirve como base para las fábricas concretas que crearán instancias de informes.
- **Clase InformePDFFactory:**
  - La clase InformePDFFactory implementa la interfaz InformeFactory y proporciona una implementación concreta para el método crearInforme(). En este caso, crea instancias de la clase InformePDF.
- **Clase DashboardComponent:**
  - La clase DashboardComponent representa un componente en tu aplicación. Esta clase tiene una dependencia en la interfaz InformeFactory, lo que significa que puede trabajar con cualquier fábrica de informes sin preocuparse por los detalles de implementación.
  - La operación generarInforme() en DashboardComponent utiliza la fábrica (InformeFactory) para crear un informe (Informe). Esto sigue el principio del Factory Method, ya que el componente no sabe qué clase concreta de informe está utilizando, delegando la creación a la fábrica.
  - Esto proporciona flexibilidad, ya que puedes cambiar fácilmente la fábrica de informes sin modificar el código del componente. Si en el futuro decides agregar un nuevo formato de informe, simplemente creas una nueva fábrica y la conectas al DashboardComponent sin afectar su lógica interna.

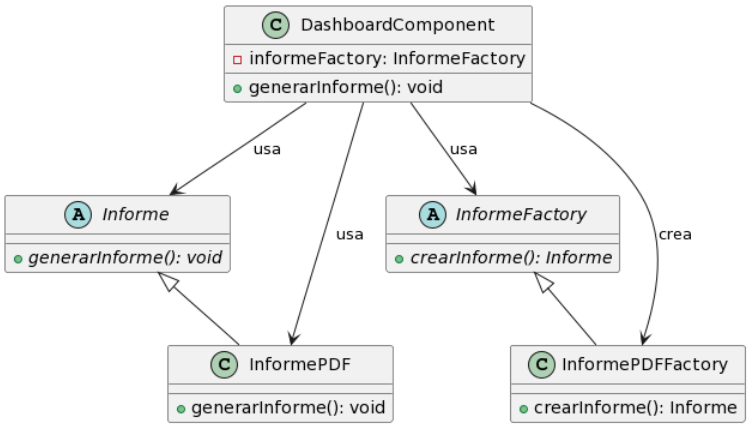

Ilustración 5 diagrama UML del patrón implementado

Conclusión

En el transcurso de este análisis exhaustivo sobre el patrón de diseño Factory Method, hemos explorado en profundidad cómo esta estrategia se ha consolidado como una herramienta esencial en el arsenal de los desarrolladores, proporcionando flexibilidad y extensibilidad inigualables al diseño de sistemas. Más allá de su aplicación específica en la creación de informes, el Factory Method se erige como un paradigma que transforma no solo el código, sino la mentalidad misma de los ingenieros de software.

La introducción de nuevas clases de fábrica concretas no es simplemente una tarea en el diseño; es un acto de previsión que permite que el sistema evolucione de manera orgánica y sin esfuerzo. Este enfoque modular no solo facilita la incorporación de nuevas funcionalidades, sino que también fomenta un entorno donde la innovación y la mejora continua son prácticas arraigadas. En el contexto de la creación de informes, esta modularidad significa que la adición de nuevos tipos de informes se convierte en un proceso sin complicaciones, preservando la integridad del código existente y allanando el camino para la evolución futura.

La creación de informes mediante el Factory Method no es solo un medio para lograr la sustitución de implementaciones sin perturbar el código del cliente; es un principio rector que traduce la filosofía de diseño a una realidad tangible. Los sistemas resultantes son más que simples conjuntos de código; son estructuras adaptables que se moldean según las necesidades cambiantes del negocio. La resistencia al cambio y la facilidad de mantenimiento se convierten en virtudes inherentes, proporcionando un terreno fértil para la evolución constante.

El desacoplamiento, piedra angular del Factory Method, trasciende su función inicial de respuesta a requisitos cambiantes. Se convierte en un catalizador para la reutilización del código, estableciendo una base sólida para el desarrollo futuro. Las implementaciones desacopladas son piezas intercambiables de un rompecabezas, permitiendo que los desarrolladores construyan sobre lo existente de manera eficiente y efectiva.

Al concluir nuestro estudio sobre el Factory Method, no solo lo consideramos una técnica eficaz para resolver desafíos específicos de creación de objetos informe, sino como un faro que guía hacia una filosofía de diseño más amplia. Este patrón encapsula valores fundamentales como la flexibilidad, la adaptabilidad y la simplicidad, sirviendo como un faro para los desarrolladores que buscan construir sistemas robustos y adaptables. Al adoptar este enfoque, los desarrolladores no solo encuentran soluciones efectivas a problemas presentes, sino que también construyen una arquitectura que se prepara para enfrentar los desafíos futuros. En resumen, el Factory Method no es simplemente una técnica; es un elemento crucial en la caja de herramientas de cualquier desarrollador que aspire a trascender las limitaciones actuales y construir software que perdure en entornos dinámicos y cambiantes.

Bibliografía

- Factory Method. (2022, enero 1). Refactoring.Guru. https://refactoring.guru/design-patterns/factory-method
- Eduard Ghergu Software Architect. (2023, marzo 30). Factory Method Design Pattern - definition & examples. Pentalog. https://www.pentalog.com/blog/design-patterns/factory-method-design-pattern/