Material descargable

egg

Patrón Experto

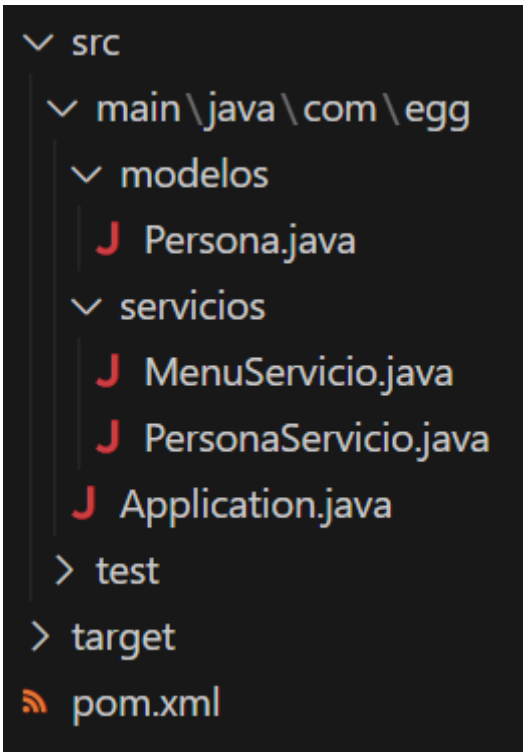
Patrón Experto (Expert)

El **Patrón Experto** es uno de los principios de diseño definidos en **GRASP** (*General Responsibility Assignment Software Patterns*), los cuales guían la asignación de responsabilidades en el diseño orientado a objetos.

Este patrón establece que **una responsabilidad debe asignarse a la clase que posee la mayor cantidad de información necesaria para cumplirla**, reduciendo la dependencia entre clases y mejorando la cohesión del sistema.

Ejemplo práctico

Supongamos que tenemos una clase `Persona` con algunos atributos y queremos imprimir sus propiedades. Una implementación incorrecta sería incluir la impresión dentro de la propia clase:



```java
public class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public void imprimirPropiedades() {
         System.out.println("Nombre: " + nombre + ", Edad: " +
edad);
    }
}
```

⚠️ **Problema**: Aquí, la clase `Persona` está asumiendo una responsabilidad que no le corresponde: **manejo de la presentación**. Esto reduce su cohesión, ya que mezcla datos con lógica de impresión.

✔️ **Solución con Patrón Experto**:
La impresión debe delegarse a otra clase que tenga la responsabilidad de manejar la presentación de los datos. En su lugar, `Persona` solo debe proporcionar los datos:

```java
public class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public String obtenerPropiedades() {
        return "Nombre: " + nombre + ", Edad: " + edad;
    }
}
```

Ahora, la impresión de los datos se delega a una clase de servicio:

```
public class PersonaServicio {
    public void imprimirPersona(Persona persona) {
        System.out.println(persona.obtenerPropiedades());
    }
}
```

Esta separación sigue el **Patrón Experto**, ya que la clase `Persona` mantiene solo los datos y `PersonaServicio` se encarga de la lógica relacionada con la presentación.

Organización del código con clases de servicio

En aplicaciones más grandes, es común organizar la lógica de negocio en **clases de servicio** que agrupan operaciones relacionadas con los objetos.

Por ejemplo, en una aplicación con múltiples personas, podríamos tener un servicio para gestionar la creación y almacenamiento de personas:

```
import java.util.ArrayList;
import java.util.List;

public class PersonaServicio {
    private List<Persona> personas = new ArrayList<>();

    public void agregarPersona(Persona persona) {
        personas.add(persona);
    }

    public void imprimirPersonas() {
        for (Persona persona : personas) {
            System.out.println(persona.obtenerPropiedades());
        }
    }
}
```

Aplicación del Patrón Experto en Menús

Otra buena práctica es delegar la lógica de interacción con el usuario a una clase independiente. En lugar de que `PersonaServicio` maneje la entrada del usuario, podemos utilizar una clase `MenuServicio`:

```
import java.util.Scanner;

public class MenuServicio {
    private final PersonaServicio personaServicio;
    private final Scanner scanner;

    public MenuServicio(Scanner scanner) {
        this.scanner = scanner;
        this.personaServicio = new PersonaServicio();
    }

    public void generarMenu() {
        System.out.println("Menú:");
        System.out.println("1 - Crear persona");
        System.out.println("2 - Mostrar personas");
        System.out.println("Selecciona una opción:");
        int opcion = scanner.nextInt();
        scanner.nextLine(); // Consumir el salto de línea

        switch (opcion) {
            case 1 -> crearPersona();
            case 2 -> personaServicio.imprimirPersonas();
            default -> System.out.println("Opción no válida.");
        }
    }

    private void crearPersona() {
        System.out.println("Ingrese un nombre:");
        String nombre = scanner.nextLine();
        System.out.println("Ingrese una edad:");
        int edad = scanner.nextInt();
        scanner.nextLine(); // Consumir el salto de línea

        Persona persona = new Persona(nombre, edad);
        personaServicio.agregarPersona(persona);
    }
}
```

Finalmente, la **clase principal** (`Application`) solo debe encargarse de iniciar el programa:

```java
import java.util.Scanner;

public class Application {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        MenuServicio menuServicio = new MenuServicio(scanner);
        menuServicio.generarMenu();
        scanner.close();
    }
}
```

✔ Aplicar el **Patrón Experto** mejora la organización y mantenimiento del código.
✔ La **clase que tiene la información relevante debe asumir la responsabilidad**, evitando que otras clases dependan innecesariamente de ella.
✔ Separar responsabilidades en **clases de servicio** ayuda a mantener un código más limpio, modular y reutilizable.

Al seguir este enfoque, logras un diseño más robusto y flexible, facilitando futuras modificaciones sin afectar otras partes del sistema. 🚀