UNIVERSIDAD DEL QUINDÍO ®
Res.MEN 014915 - 02 AGO 2022
RENOVACIÓN ACREDITACIÓN

Template Method

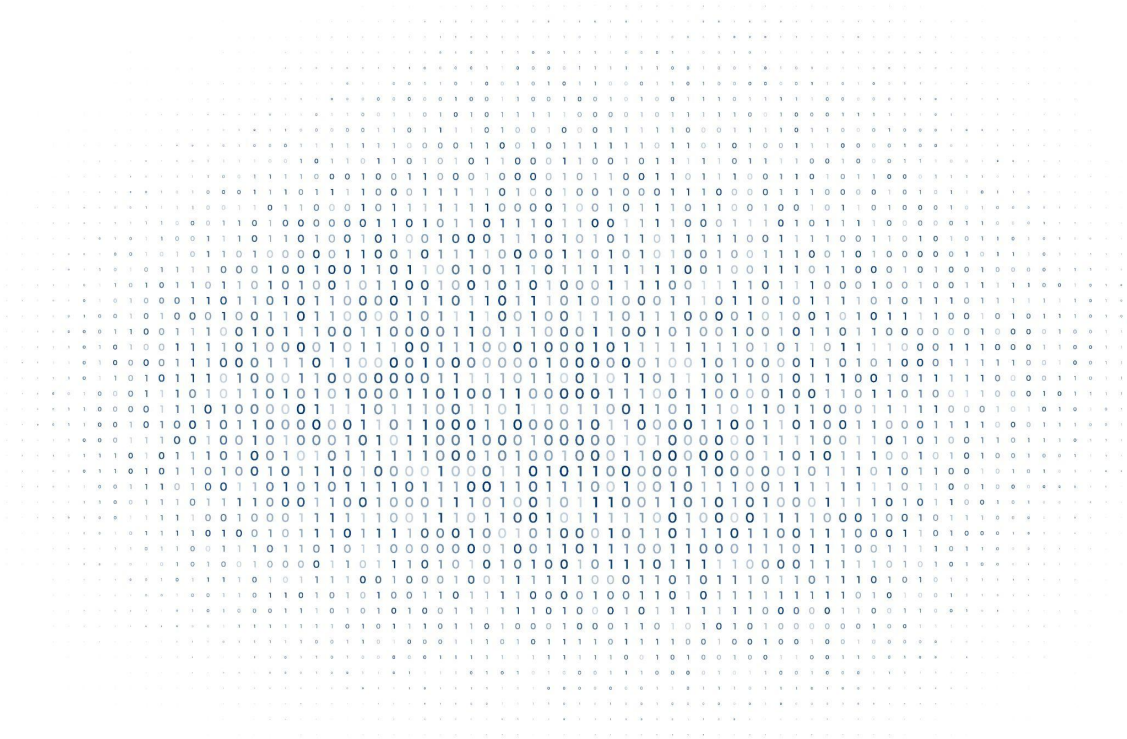
Juan Camilo López Fuentes
Juan Pablo López Gómez
Daniel Josué Narvaéz Hincapié
Willinton Vergara Cataño

UNIQUINDÍO
en conexión territorial
www.uniquindio.edu.co

¿Qué es el Template Method?

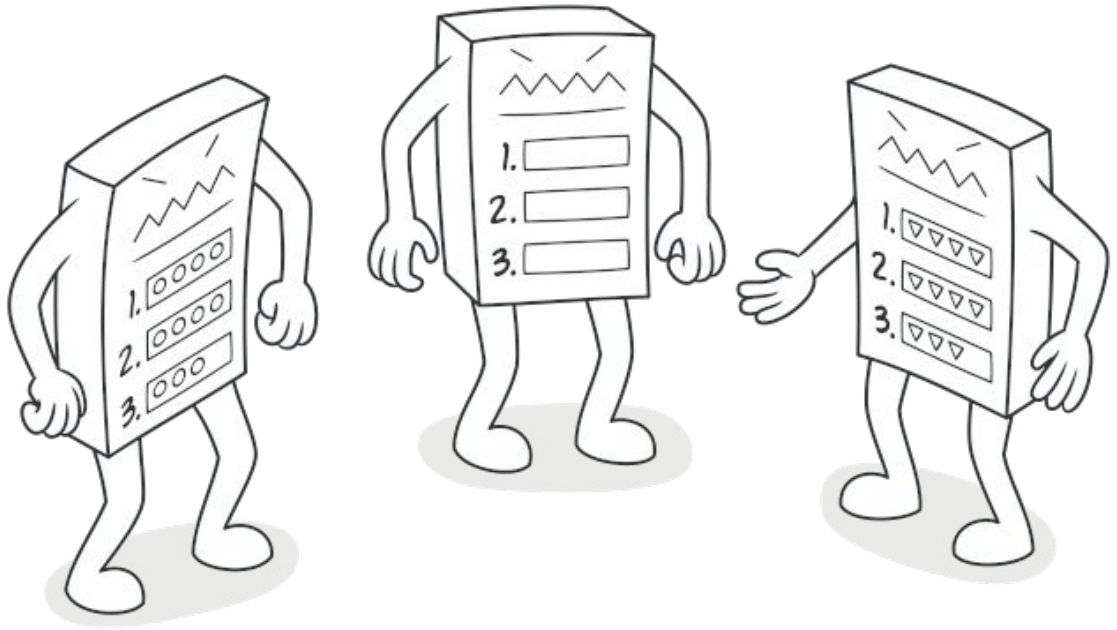
Es un patrón de diseño comportamental.

Define un algoritmo base para un conjunto de clases, es decir, una plantilla; esto, permitiendo que las subclases puedan cambiar partes del algoritmo sin modificar el original, por ejemplo, el tema de estas diapositivas.



¿Cuál es el propósito del Template Method?

En pocas palabras el propósito del método plantilla es proporcionar una forma ordenada y flexible de hacer las cosas que se repiten de manera similar permitiendo pequeños cambios.



Por ejemplo:

Es como tener un esquema o una plantilla para realizar una tarea, pero permitiendo ajustar algunos detalles según lo necesite. Esto ayuda a evitar errores, ahorra tiempo y hace que sea más fácil hacer cambios en el futuro.

¿En dónde lo puedo utilizar?

Procesamiento de datos

1. ETL: Establece cómo extraer, transformar y cargar datos; las subclases detallan cada paso según la fuente y el destino.
2. Procesamiento en tiempo real: Recibe, procesa y almacena datos en tiempo real, con subclases que manejan la lógica específica.
3. Limpieza y validación de datos: Carga, limpia y valida datos, con subclases que implementan reglas particulares.

¿En dónde lo puedo utilizar?

Interfaces de usuario

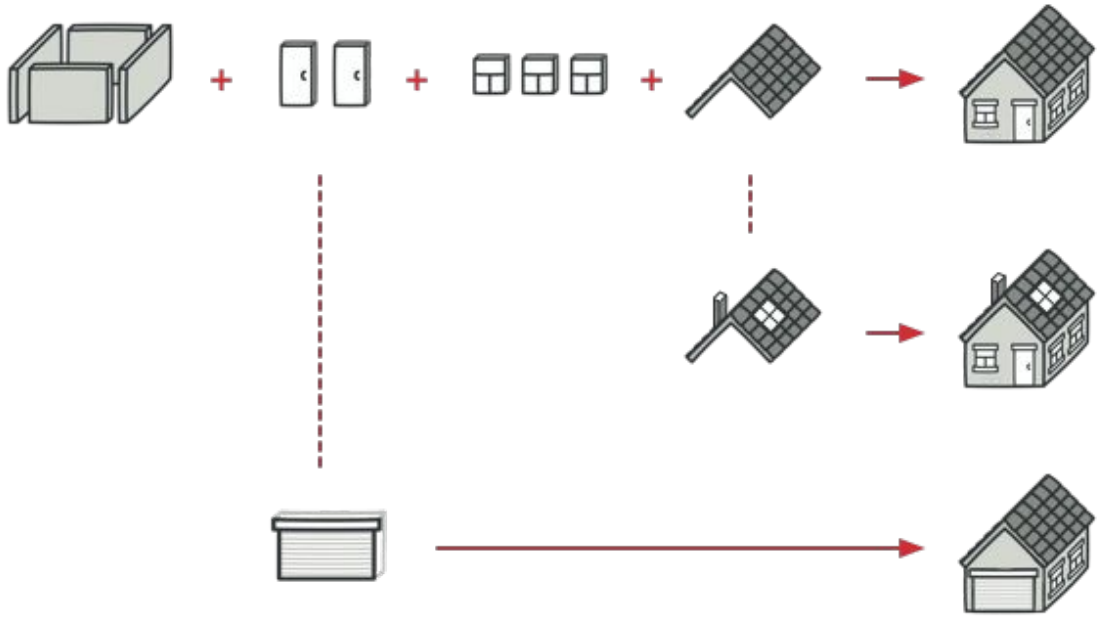
1. Componentes Reutilizables: Definir una estructura común con variaciones en partes específicas, como ventanas modales con un encabezado y pie de página comunes y contenido variable.

2. Flujos de Trabajo: Gestionar flujos de trabajo complejos con pasos fijos pero detalles variables, como asistentes de configuración con etapas de inicio, configuración y finalización personalizables.

3. Renderizado de Páginas: Generar páginas web con una estructura estándar (encabezado y pie de página) y contenido dinámico específico, asegurando consistencia y reutilización del código.

Aplicación

Su aplicación es muy sencilla, ya que define el esqueleto de algún algoritmo en una superclase permitiendo que sus subclases sobreescriban algunos pasos sin cambiar su estructura.



En otras palabras, define la estructura base para usarla en un objeto o contexto similar.

Ventajas

1. Reutilización de Código: Permite definir el esqueleto de un algoritmo en una clase base y reutilizar este código común en múltiples subclases, evitando la duplicación.

2. Mantenimiento Simplificado: Facilita el mantenimiento centralizando los cambios en un único lugar. Cualquier modificación en el flujo del algoritmo se realiza en la clase base, reduciendo el riesgo de errores e inconsistencias.

3. Extensibilidad: Las subclases pueden extender la funcionalidad del algoritmo añadiendo o modificando pasos específicos sin alterar la estructura general definida en la clase base.

4. Encapsulamiento del Código Común: Encapsula el código común a múltiples implementaciones en la clase base, mejorando la coherencia del código.

5. Fácil de Implementar y Usar: Su estructura basada en herencia y en la definición de métodos abstractos o concretos lo hace accesible y fácil de aplicar incluso para desarrolladores con menos experiencia en patrones de diseño.

Desventajas

1. Complejidad Inicial: Requiere una mayor planificación y un diseño más complejo, lo que puede ser desalentador para desarrolladores junior.

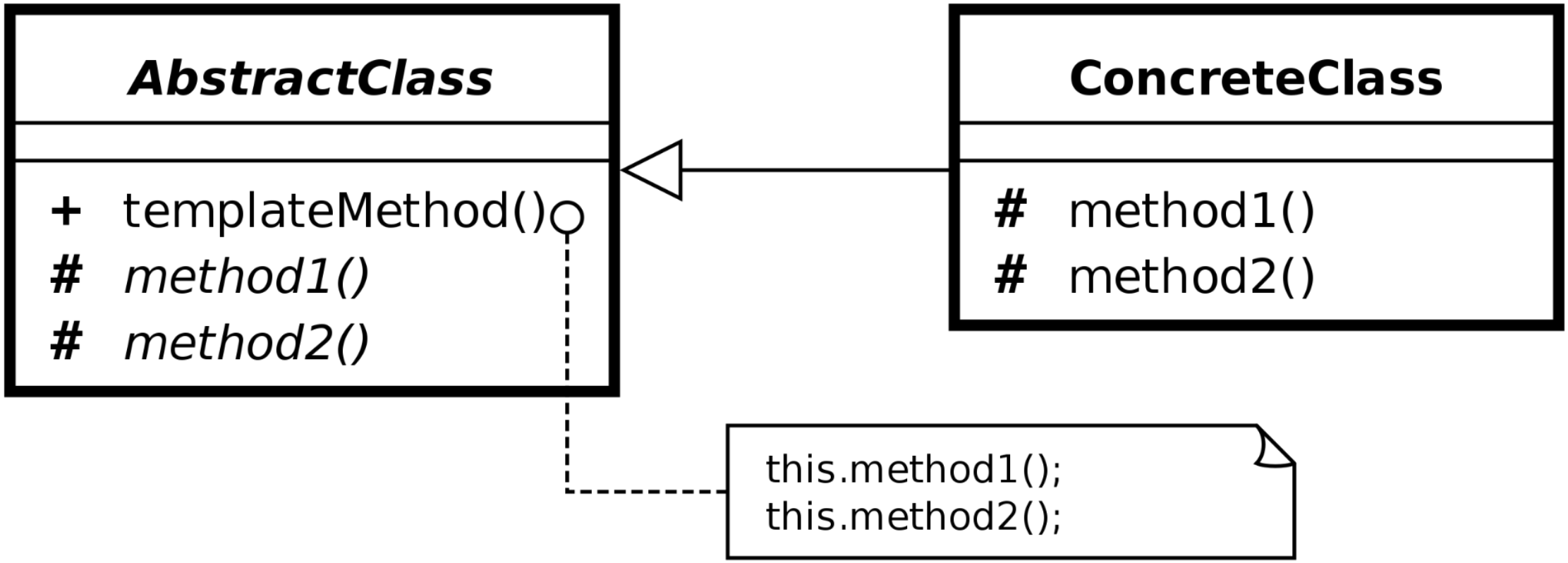
2. Fuerte Acoplamiento: Las subclases están fuertemente acopladas a la clase base, lo que puede dificultar la evolución del código ya que cualquier cambio en la clase base afecta a todas las subclases.

3. Falta de flexibilidad: La capacidad de personalización está limitada, ya que las subclases solo pueden modificar ciertos pasos del algoritmo sin alterar su estructura general.

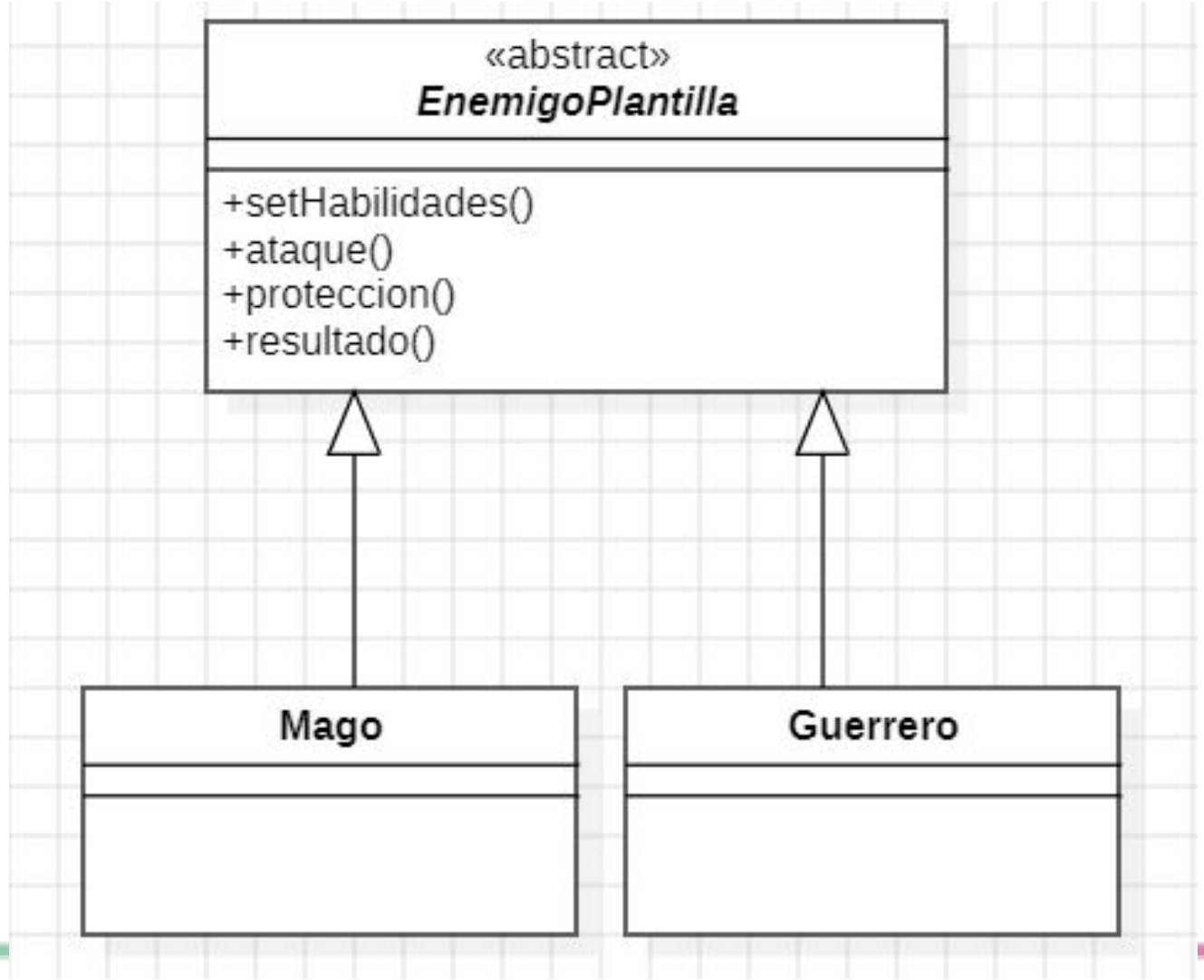
4. Sobrecarga de Herencia: Depende del uso extensivo de herencia, lo que puede llevar a jerarquías de clases profundas y complejas, dificultando la comprensión y el mantenimiento del diseño.

5. Rigidez en la Evolución del Código: Cambios fundamentales en el algoritmo pueden requerir una reestructuración extensa tanto en la clase base como en las subclases, lo que puede ser laborioso y propenso a errores.

Diagrama UML genérico



Ejemplo de programa

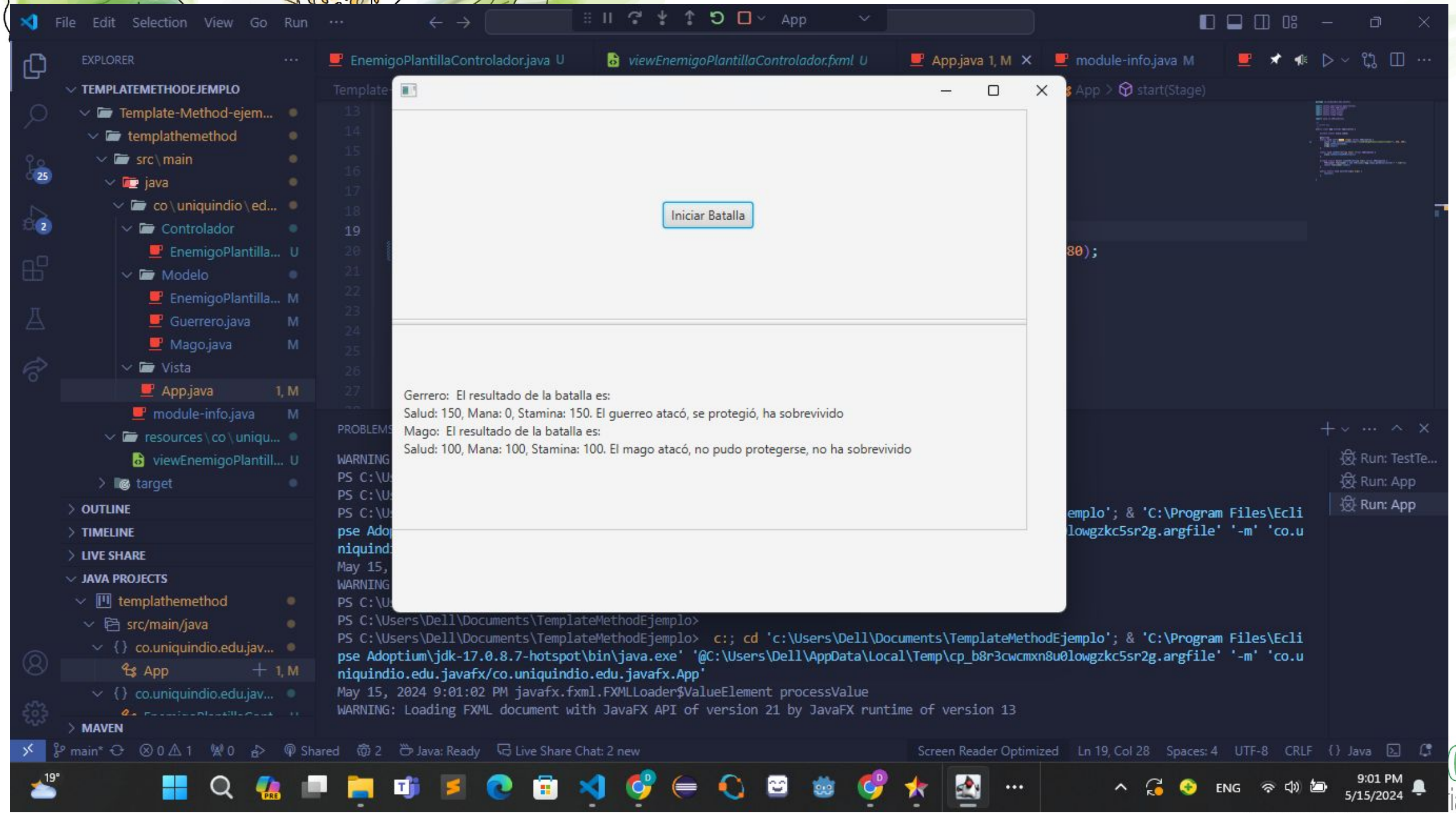


```java
package co.uniquindio.edu.javafx.Modelo;

public abstract class EnemigoPlantilla {

    public abstract String setHabilidades();
    public abstract String ataque();
    public abstract String proteccion();
    public abstract String resultado();

    public final String template() {
        String mensaje ="El resultado de la batalla es:\n" + setHabilidades() + ataque() + proteccion() + resultado();
        return mensaje;
    }
```

```java
package co.uniquindio.edu.javafx.Modelo;

public class Guerrero extends EnemigoPlantilla {

    @Override
    public String setHabilidades() {
        String mensajeHabilidades = "Salud: 150, Mana: 0, Stamina: 150";
        return mensajeHabilidades;
    }

    @Override
    public String ataque() {
        String mensajeAtaque = ". El guerreo atacó";
        return mensajeAtaque;
    }

    @Override
    public String proteccion() {
        String mensajeProteccion =", se protegió";
        return mensajeProteccion;
    }

    @Override
    public String resultado() {
        String mensajeResultado = ", ha sobrevivido";
        return mensajeResultado;
    }
}
```

```java
package co.uniquindio.edu.javafx.Modelo;

public class Mago extends EnemigoPlantilla{

    @Override
    public String setHabilidades() {
        String mensajeHabilidades = "Salud: 100, Mana: 100, Stamina: 100";
        return mensajeHabilidades;
    }

    @Override
    public String ataque() {
        String mensajeAtaque = ". El mago atacó";
        return mensajeAtaque;
    }

    @Override
    public String proteccion() {
        String mensajeProteccion =", no pudo protegerse";
        return mensajeProteccion;

    }

    @Override
    public String resultado() {
        String mensajeResultado = ", no ha sobrevivido";
        return mensajeResultado;
    }
}
```

```java
import co.uniquindio.edu.javafx.Modelo.EnemigoPlantilla;
import co.uniquindio.edu.javafx.Modelo.Guerrero;
import co.uniquindio.edu.javafx.Modelo.Mago;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

public class EnemigoPlantillaControlador {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private Button btnIniciar;

    @FXML
    private Label txtResultado;

    @FXML
    void iniciarBatalla(ActionEvent event) {
        EnemigoPlantilla guerrero = new Guerrero();
        EnemigoPlantilla mago = new Mago();
        //guerrero.template();
        //mago.template();
        txtResultado.setText("Gerrero:   " + guerrero.template() +"\n" + "Mago:   " + mago.template());
    }
```


Iniciar Batalla
Gerrero: El resultado de la batalla es:
Salud: 150, Mana: 0, Stamina: 150. El guerreo atacó, se protegió, ha sobrevivido
Mago: El resultado de la batalla es:
Salud: 100, Mana: 100, Stamina: 100. El mago atacó, no pudo protegerse, no ha sobrevivido

UNIVERSIDAD DEL QUINDÍO ®

ALTA CALIDAD
Res.MEN 014915 - 02 AGO 2022

CERTIFICADO POR SU COMPROMISO RESPONSABILIDAD SOCIAL FENALCO SOLIDARIO ®

Referencia:
Template Method. (s.f.). Refactoring and Design Patterns.
https://refactoring.guru/es/design-patterns/template-method

Link del repositorio:
https://github.com/DanielJNarvaezH/Template-Method-ejemplo