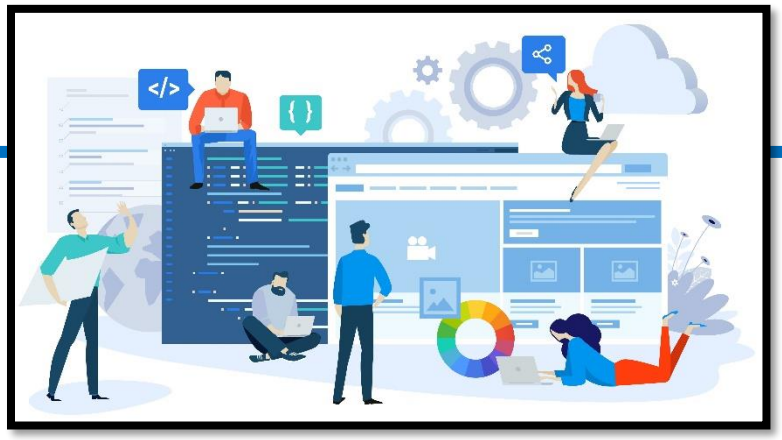
UNIVERSIDAD ISRAEL



TEMA: TAREA SEMANA 6

NOMBRE: VALVERDE PEDRO

CURSO: NAC

FECHA: 22/04/2023

MODALIDAD: VIRTUAL

CARRERA: INGENIERÍA EN INFORMÁTICA

MATERIA: FUNDAMENTOS DE LA ESPECIALIDAD

AÑO LECTIVO

2023 – A

Patrón Estructural

Arquitectura Builder

Declaramos las variables que vamos a utilizar para preparar una pizza con sus repectivos métodos getter and setter

```
public class Pizza {
    private String tipo="";
    private String masa = "";
    private String salsa = "";
    private String relleno = "";
    private String Ingredientes="";

    public String getTipo() {
       return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public String getIngredientes() {
        return Ingredientes;
    }

    public void setIngredientes(String Ingrediente) {
        this.Ingredientes = Ingrediente;
    }

    public void setMasa(String masa) {
        this.masa = masa;
    }

    public void setSalsa(String salsa) {
        this.salsa = salsa;
    }

    public void setRelleno(String relleno) {
        this.relleno = relleno;
    }

    public String getMasa() {
        return masa;
    }

    public String getSalsa() {
        return salsa;
    }

    public String getRelleno() {
        return relleno;
    }
```

Toma un constructor de pizza (PizzaBuilder) y utiliza los métodos proporcionados por El Él constructor para coordinar y completar la construcción de una pizza. Esta separación de responsabilidades permite construir pizzas con diferentes variaciones de ingredientes y características de forma modular y flexible.

```
package backend;

import backend.builder.PizzaBuilder;

/**
 *
 * @author HP PAVILON
 */
public class Cocina {
    private PizzaBuilder builder;

    public void setPizzaBuilder(PizzaBuilder pizzaBuilder) {
        builder = pizzaBuilder;
    }

    public Pizza getPizza() {
        return builder.getPizza();
    }

    public void construirPizza() {
       builder.crearNuevaPizza();
       builder.buildTipo();
       builder.buildMasa();
       builder.buildSalsa();
       builder.buildRelleno();
       builder.buildIngredientes();
    }
}
```

Las subclases de PizzaBuilder implementarán estos métodos abstractos para definir cómo se construyen las diferentes partes de la pizza y así poder crear diferentes tipos de pizzas de forma modular y flexible.

```
public abstract class PizzaBuilder {

    protected Pizza pizza;

    public Pizza getPizza() {
        return pizza;
    }

    public void crearNuevaPizza() {
        pizza = new Pizza();
    }

    public abstract void buildMasa();
    public abstract void buildSalsa();
    public abstract void buildRelleno();
    public abstract void buildIngredientes();
    public abstract void buildTipo();


}
```

Hereda la estructura y los métodos básicos de `PizzaBuilder`, pero proporciona implementaciones específicas para cada parte de la pizza hawaiana, permitiendo así la construcción de este tipo específico de pizza de manera modular y flexible.

```java
public class HawaiPizzaBuilder extends PizzaBuilder {



    @Override
    public void buildMasa() {
        pizza.setMasa("suave");
    }

    @Override
    public void buildSalsa() {
        pizza.setSalsa("dulce");
    }

    @Override
    public void buildRelleno() {
        pizza.setRelleno("chorizo+alcachofas");
    }

    @Override
    public void buildIngredientes() {
        pizza.setIngredientes("Mozarella+salsa+Piña");
    }

    @Override
    public void buildTipo() {
      pizza.setTipo("Hawayana");
    }
```

Hereda la estructura y los métodos básicos de PizzaBuilder, pero proporciona implementaciones específicas para cada parte de la pizza picante, permitiendo así la construcción de este tipo específico de pizza de manera modular y flexible.

```
public class PizzaPicanteBuilder extends PizzaBuilder {


    @Override
    public void buildMasa() {
        pizza.setMasa("cocida");
    }

    @Override
    public void buildSalsa() {
        pizza.setSalsa("picante");
    }

    @Override
    public void buildRelleno() {
        pizza.setRelleno("pimienta+salchichón");
    }

    @Override
    public void buildIngredientes() {
        pizza.setIngredientes("Mozarella+salsa+peperoni");
    }

    @Override
    public void buildTipo() {
        pizza.setTipo("Picante");
    }
}
```

Utilizaremos los constructores para de esta forma poderlo hacer dinámico y que se pueda apreciar de mejor manera usaremos un do while y un switch case para cada caso en este caso usaremos 2 sabores de pizzas

```java
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Cocina cocina = new Cocina();

    // Crear los builders
    PizzaBuilder hawaiPizzaBuilder = new HawaiPizzaBuilder();
    PizzaBuilder pizzaPicanteBuilder = new PizzaPicanteBuilder();

    int opcion;
    do {
        // Menú para seleccionar el tipo de pizza
        System.out.println("Bienvenido a la pizzería!");
        System.out.println("Seleccione el tipo de pizza:");
        System.out.println("1. Pizza Hawaiana");
        System.out.println("2. Pizza Picante");
        System.out.println("0. Salir");
        System.out.println("Seleccione una pizza: ");

        opcion = scanner.nextInt();

        PizzaBuilder pizzaBuilderSeleccionado = null;
        switch (opcion) {
            case 1:
                pizzaBuilderSeleccionado = hawaiPizzaBuilder;
                break;
            case 2:
                pizzaBuilderSeleccionado = pizzaPicanteBuilder;
                break;
            case 0:
                System.out.println("Gracias por visitar nuestra pizzería. ¡Hasta luego!");
                break;
            default:
                System.out.println("Opción no válida, seleccionando pizza hawaiana por defecto.");
                pizzaBuilderSeleccionado = hawaiPizzaBuilder;
                break;
        }

        if (opcion != 0) {
            // Construir la pizza seleccionada
            cocina.setPizzaBuilder(pizzaBuilderSeleccionado);
            cocina.construirPizza();
            Pizza pizza = cocina.getPizza();

            // Mostrar la información de la pizza por consola
            System.out.println("Pizza: "+ pizza.getTipo());
            System.out.println("Masa: " + pizza.getMasa());
            System.out.println("Salsa: " + pizza.getSalsa());
            System.out.println("Ingredientes: " + pizza.getIngredientes());
        }
    } while (opcion != 0);
}
```

Ejecución

En la ejecución obtendremos el siguiente menú con una pizza hawaiana utilizaremos la pizza hawaiana y nos mostrará el sabor, la masa, la salsa, los ingrediente y nos mostrara el menú para la opciones de pizza.

```
run:
Bienvenido a la pizzería!
Seleccione el tipo de pizza:
1. Pizza Hawaiana
2. Pizza Picante
0. Salir
Seleccione una pizza:
3
Opción no válida, seleccionando pizza hawaiana por defecto.
Pizza: Hawayana
Masa: suave
Salsa: dulce
Ingredientes: Mozarella+salsa+Piña
Bienvenido a la pizzería!
Seleccione el tipo de pizza:
1. Pizza Hawaiana
2. Pizza Picante
0. Salir
Seleccione una pizza:
```

En conclusión

Cuando necesite construir objetos intrincados con componentes móviles y desee dotar al

proceso de construcción de una interfaz clara y controlada, puede utilizar el patrón de diseño Builder. Es particularmente útil en escenarios donde la complejidad de la construcción del objeto es grande y se requieren varias variantes de la cosa porque ofrece una solución flexible y modular para la creación de objetos complicados.

Nota: Se adjunta el proyecto por la herramienta git hub https://github.com/Eduardo199614/PatronCreacionalBuilder.git