<!--Estudio Shonos-->

Patrones de diseño {

Por=</"Juan David Ramirez Carmona
Laura Valentia Gomez Alzate
Nicolas Muñoz Viveros"/>

UNIVERSIDAD DEL QUINDÍO

Abstract Factory{

se plantea el problema donde se quieren crear el menú y la comida de dos restaurantes, uno mexicano y uno chino, para esto se debe implementar el patrón abstract factory, para los 2 restaurantes, los cuales serían el restaurante mexicano y restaurante chino.

}

# Que problema soluciona
{

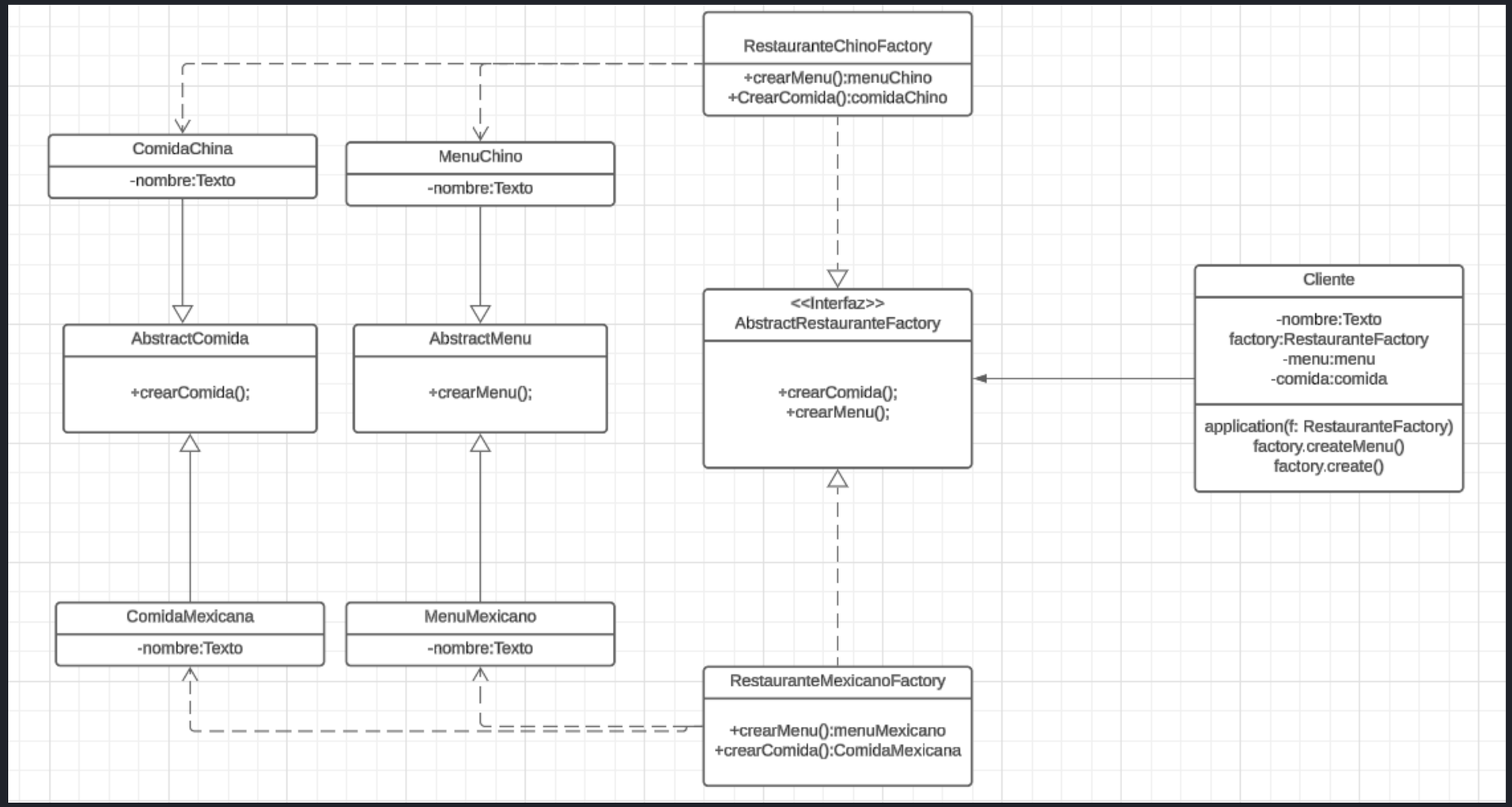
Este proporciona una interfaz para crear familias de objetos sin especificar clases concretas introducción de nuevas familias de objetos sin modificar el código existente

Divide la responsabilidad de creación de objetos entre la fábrica abstracta y sus implementaciones concretas, promoviendo una clara separación de responsabilidades en el diseño.

introduce nuevas familias de objetos sin modificar el código existente haciendo cumplimiento del principio open/close

}

UML {


RestauranteChinoFactory
+crearMenu():menuChino
+CrearComida():comidaChino
ComidaChina
-nombre:Texto
MenuChino
-nombre:Texto
AbstractComida
+crearComida();
AbstractMenu
+crearMenu();
<<Interfaz>>
AbstractRestauranteFactory
+crearComida();
+crearMenu();
Cliente
-nombre:Texto
factory:RestauranteFactory
-menu:menu
-comida:comida
application(f: RestauranteFactory)
factory.createMenu()
factory.create()
ComidaMexicana
-nombre:Texto
MenuMexicano
-nombre:Texto
RestauranteMexicanoFactory
+crearMenu():menuMexicano
+crearComida():ComidaMexicana

}

Interfaz Abstract Restaurant Factory {

```java
package co.edu.uniquindio.poo;

/*El propósito principal de esta interfaz es proporcionar una abstracción para la creación de familias
 de objetos (menús y comidas) que deben ser compatibles entre sí*/

interface RestauranteFactory {
    AbstractMenu crearAbstractMenu();
    AbstractComida crearAbstractComida();
}

```

}

Class Restaurante Chino Factory {

```java
package co.edu.uniquindio.poo;

/*Esta fábrica concreta se utiliza para crear objetos específicos de un restaurante chino,
asegurando la coherencia entre los productos creados*/
public class RestauranteChinoFactory implements RestauranteFactory {
    private MenuChino menu;
    private ComidaChina comida;

    @Override
    public AbstractMenu crearAbstractMenu()  {
    return new MenuChino();
    }

    @Override
    public AbstractComida crearAbstractComida() {
    return new ComidaChina();
    }
}
/*La utilización de la interfaz RestauranteFactory permite que esta clase se integre en un
sistema que utiliza patrones de diseño para la creación de objetos.*/
```

}

Class Restaurante Mexicano Factory {

```java
package co.edu.uniquindio.poo;

/* Esta fabrica es igual a la anterior solo enfocandose en elementos de restaurante mexicano*/
public class RestauranteMexicanoFactory implements RestauranteFactory {
    private ComidaMexicana comida;
    private MenuMexicano menu;

    @Override
    public AbstractComida crearAbstractComida(){
        return new ComidaMexicana();
    }

    @Override
    public AbstractMenu crearAbstractMenu(){
        return new MenuMexicano();
    }
}

```

}

Class Menu Mexicano Factory {

```java
package co.edu.uniquindio.poo;
/**
 * La clase {@code MenuMexicano} implementa la interfaz {@code AbstractMenu},
 * representando un menu especifico de un restaurante mexicano.
 */
public class MenuMexicano implements AbstractMenu {
    private String nombre;

    public MenuMexicano() {
    }
    @Override
    public void leer() {
        System.out.println("leyendo");
    }

        @Override
    public String getNombre() {
        return nombre;
    }
```

}

Class comida Mexicana Factory {

```java
package co.edu.uniquindio.poo;

/**
 * La clase {ComidaMexicana} implementa la interfaz {AbstractComida},
 * representando una comida de un restaurante mexicano.
 */
public class ComidaMexicana implements AbstractComida {

    private String name;

    public ComidaMexicana(){

    }

    @Override
    public void preparar() {
        System.out.println("Preparando comida mexicana");
    }
}

```

}

Clases de objetos concretos {

```java
package co.edu.uniquindio.poo;
/*Los metodos aqui definidos proporcionan una abstraccion para las operaciones comunes que se esperan
de cualquier menu en el sistema. La implementacion exacta
 puede variar segun el tipo especifico de menu*/
interface AbstractMenu {
    void leer();
    String getNombre();
}

```

abstractfactory > src > main > java > co > edu > uniquindio > poo > AbstractComida.java > AbstractComida

```java
package co.edu.uniquindio.poo;
 /**
     * Realiza las acciones necesarias para preparar la comida.
     * La implementacion concreta de este metodo puede variar segun la naturaleza de la comida.
     */

public interface AbstractComida {
    void preparar();
}
```

}

Clase Cliente {

abstractfactory > src > main > java > co > edu > uniquindio > poo > Cliente.java > Cliente

```java
/**
 * La clase { Cliente} representa un cliente en un sistema de restaurantes que utiliza
 * el patron Abstract Factory para crear y mostrar informacion sobre comidas y menus.
 */
public class Cliente {
    private RestauranteFactory factory;
    private AbstractComida comida;
    private AbstractMenu menu;


public Cliente(RestauranteFactory factory) {
        this.factory = factory;
    }


public RestauranteFactory getFactory() {
    return factory;
}


public AbstractComida getComida() {
    return comida;
}


public AbstractMenu getMenu() {
    return menu;
}

public void createUI(){
    this.comida = factory.crearAbstractComida();
    /**
     * Muestra informacion sobre la comida y el menu.
     * Se asume que el metodo {@code createUI()} ha sido llamado previamente.
     */
public void paint() {
        comida.preparar();
        System.out.println("Nombre del menú: " + menu.getNombre());
        System.out.println("Tipo de comida: " + comida.getClass().getSimpleName
        menu.leer();
}

}
```

}

Clase App{

```java
public class App {

    /**
 * La clase { App} contiene el metodo principal ({ main}) para probar el funcionamiento
 * del patron Abstract Factory en un entorno de restaurantes chinos y mexicanos.
 */
    Run | Debug
    public static void main(String[] args) {

        RestauranteFactory restauranteChinoFactory = new RestauranteChinoFactory();

        Cliente clienteChino = new Cliente(restauranteChinoFactory);

        clienteChino.createUI();

        clienteChino.paint();


        RestauranteFactory restauranteMexicanoFactory = new RestauranteMexicanoFactory();

        Cliente clienteMexicano = new Cliente(restauranteMexicanoFactory);

        clienteMexicano.createUI();

        clienteMexicano.paint();

    }
```

}

Prototype {

que problema resuelve

este patron soluciona el problema de clonacion de objetos teniendo un objeto de referencia, que en algunos casos requiere mucho tiempo y recursos.

que hace el patron

se conecta la interface con la clase cloneable para indicar que esta bien clonarla

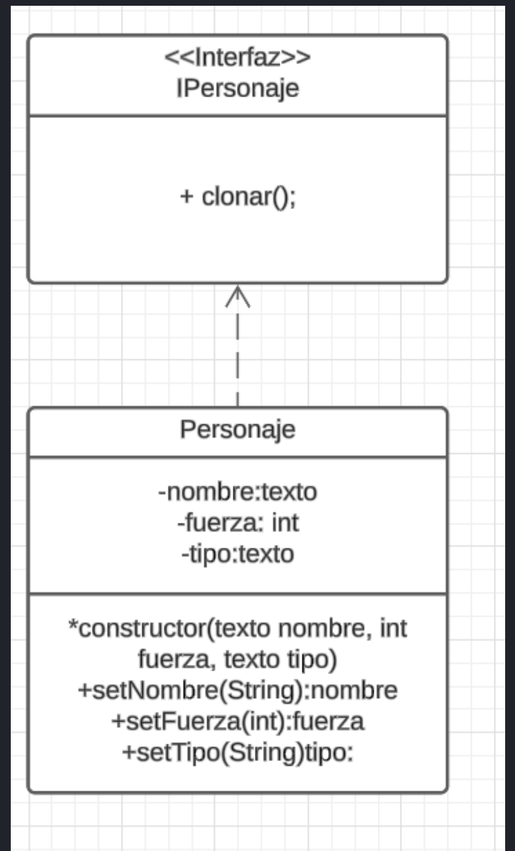
este patron proporciona un prototipo para no tener que crearlo desde cero, y lo hace desde el propio objeto par tener acceso a atributos privados sin problema

}

proposito {

en este caso se tiene la clase personaje, y la interfaz IPersonaje, con el proposito de clonar a el personaje

uml{




```java
public class Personaje implements Ipersonaje{
    private String nombre;
    private int fuerza;
    private String tipo;
// Constructor para inicializar los campos del personaje
    public Personaje(String nombre, int fuerza, String tipo){
        assert nombre != null;
        this.nombre = nombre;
        this.fuerza = fuerza;
        this.tipo = tipo;
    }
     // Métodos para establecer los valores de los campos
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public void setFuerza(int fuerza) {
        this.fuerza = fuerza;
    }

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

    @Override
    public Ipersonaje clonar() {
        Personaje personaje = null;
        // Utiliza el método clone() para obtener una copia
        try{
            personaje = (Personaje) clone();//mapeo se busca obtener un clon de esa insta
        }catch(CloneNotSupportedException e){
            e.printStackTrace();
        }
        return personaje;
    }
// Método toString para representar el objeto como una cadena de texto
    @Override
    public String toString(){
        return "Personaje "+ " [ nombre: "+nombre+" fuerza: "+fuerza+ " tipo: "+tipo+"]";
    }
}
```

}

```java
//interfaz que implementa cloneable
public interface Ipersonaje extends Cloneable {
    //metodo clonar para crera una copia de personaje
    Ipersonaje clonar();

}
```

```java
// Clase principal que prueba la funcionalidad de clonación del personaje
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Se crea un objeto de tipo Personaje
        Personaje personaje = new Personaje(nombre:"Juan", fuerza:200, tipo:"paladin");

        // Se clona el personaje
        Personaje personajeClon = (Personaje) personaje.clonar();

        // Se verifica si el clon fue creado correctamente
        if(personajeClon != null){
            // Se imprime el personaje clonado
            System.out.println("Personaje clonado: " + personajeClon);
        } else {
            // En caso de que el clon no se haya creado correctamente
            // (esto no debería ocurrir en este ejemplo)
            System.out.println(x:"Error al clonar el personaje.");
        }
    }
}
```

link repositorio

https://github.com/JuamchoOrion/Abstract-Factory-Prototype

}

<!--Estudio Shonos-->

Gracias {

<Link GitHub: >

}