Universidad Tecnológica del Perú

Docente:

Dilian Anabel HURTADO PONCE

Tarea 11: Patrones estructurales: Patrón proxy, Patrón bridge, Patrón decorator y Patrón composite

.

Semana: 11

GRUPO 5

Integrantes:

Roberto Agustín Mejía Collazos

Miguel Ángel Velásquez Ysuiza

Manuel Ángel Pecho Santos

Daniel Wilfredo Sotomayor Beteta

31/10/24

Codigo GitHub

Bienvenido | Client.java 1 | MenuComponent.java 1 | MenuComposit

menu > Client.java > ...

```java
package composite.examples.menu;
public class Client {
    private MenuComponent allMenus;

    public Client(MenuComponent todosLosMenus) {
        this.allMenus = todosLosMenus;
    }

    public void printMenu() {
        allMenus.print();
    }
}
```

Bienvenido | Client.java 1 | MenuComponent.java 1 | MenuComposite.java 1 | MenuIte

menu > MenuComponent.java > ...

```java
package composite.examples.menu;

public abstract class MenuComponent {

    protected String name;
    protected static StringBuffer identado = new StringBuffer();

    public void add(MenuComponent component) {
        throw new UnsupportedOperationException();
    }

    public void remove(MenuComponent component) {
        throw new UnsupportedOperationException();
    }

    public String getName() {
        throw new UnsupportedOperationException();
    }

    public double getPrice() {
        throw new UnsupportedOperationException();
    }

    public boolean isVegetarian() {
        throw new UnsupportedOperationException();
    }

    public void print() {
        throw new UnsupportedOperationException();
    }
}
```

Bienvenido | Client.java 1 | MenuComponent.java 1 | MenuComposite.java 1 | MenuItem.java 1 | Me

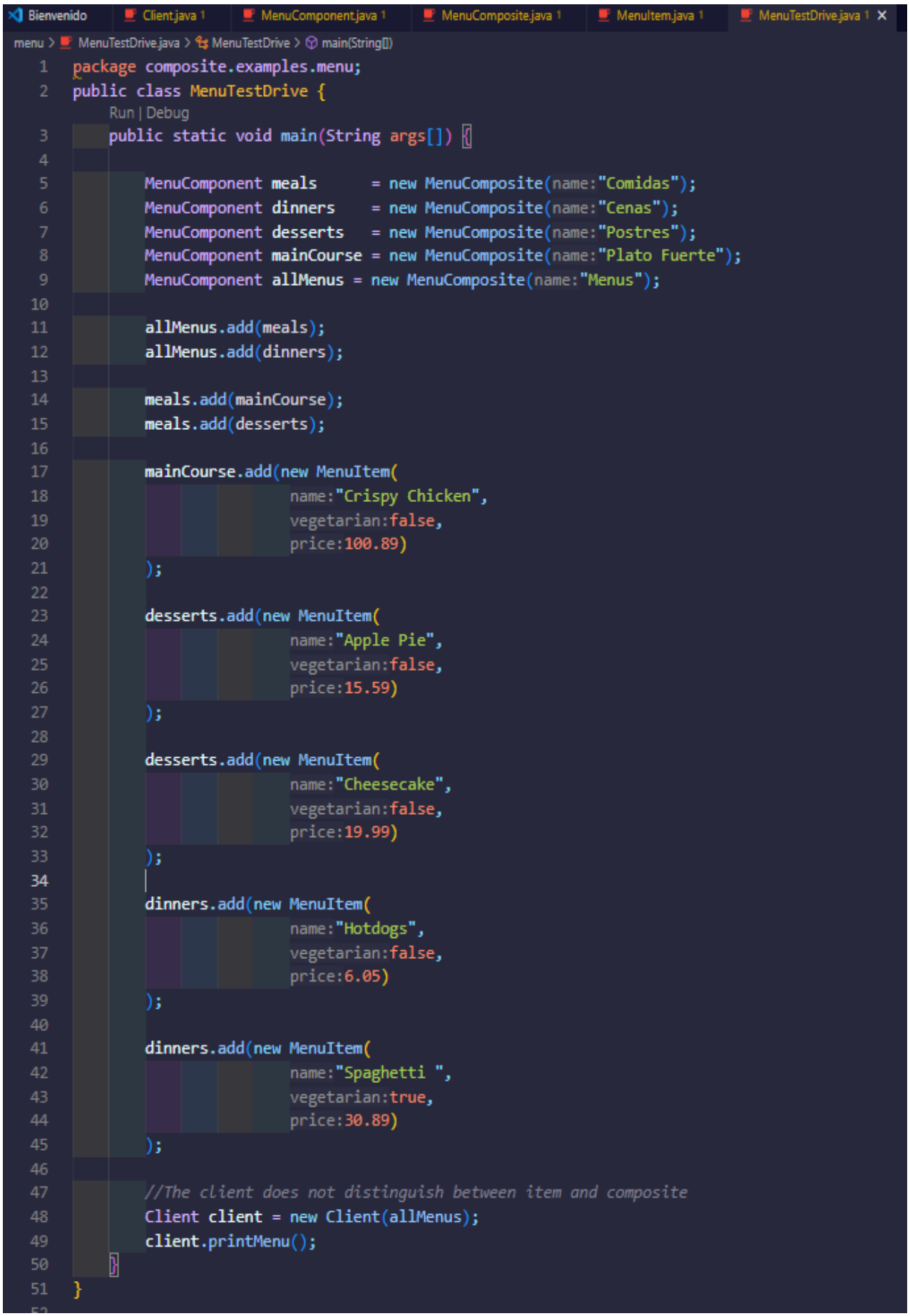
menu > MenuComposite.java > ...

```java
package composite.examples.menu;

import java.util.ArrayList;

public class MenuComposite extends MenuComponent {

    private ArrayList<MenuComponent> menuComponents;

    public MenuComposite(String name) {
        this.name = name;
        menuComponents = new ArrayList<>();
    }

    @Override
    public void add(MenuComponent component) {
        this.menuComponents.add(component);
    }

    @Override
    public void remove(MenuComponent component) {
        this.menuComponents.remove(component);
    }

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public void print() {

        System.out.print(identado.toString() + "* " + getName() + "\n");
        //System.out.println(identado.toString() + "--------------------");
        identado.append(str:"     ");
        for(MenuComponent menuComponent : menuComponents){
            menuComponent.print();
        }
        identado.setLength(identado.length() - 5);
    }
}
```

Bienvenido | Client.java 1 | MenuComponent.java 1 | MenuComposite.java 1 | MenuItem.java 1

menu > MenuItem.java > MenuItem

```java
package composite.examples.menu;

/**
 *
 * @author luisburgos
 */
public class MenuItem extends MenuComponent {

    private boolean vegetarian;
    private double price;

    public MenuItem(String name, boolean vegetarian, double price) {
        this.name = name;
        this.vegetarian = vegetarian;
        this.price = price;
    }

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public double getPrice() {
        return this.price;
    }

    @Override
    public boolean isVegetarian() {
        return this.vegetarian;
    }

    @Override
    public void print() {
        System.out.print(identado.toString() + "# " +  getName());
        if (isVegetarian()) {
            System.out.print(s:"(v)");
        }
        System.out.println("," + getPrice());
    }

}

```


Bienvenido
Client.java 1
MenuComponent.java 1
MenuComposite.java 1
MenuItem.java 1
MenuTestDrive.java 1
menu > MenuTestDrive.java > MenuTestDrive > main(String[])
package composite.examples.menu;
public class MenuTestDrive {
Run | Debug
public static void main(String args[]) {
MenuComponent meals = new MenuComposite(name:"Comidas");
MenuComponent dinners = new MenuComposite(name:"Cenas");
MenuComponent desserts = new MenuComposite(name:"Postres");
MenuComponent mainCourse = new MenuComposite(name:"Plato Fuerte");
MenuComponent allMenus = new MenuComposite(name:"Menus");
allMenus.add(meals);
allMenus.add(dinners);
meals.add(mainCourse);
meals.add(desserts);
mainCourse.add(new MenuItem(
name:"Crispy Chicken",
vegetarian:false,
price:100.89)
);
desserts.add(new MenuItem(
name:"Apple Pie",
vegetarian:false,
price:15.59)
);
desserts.add(new MenuItem(
name:"Cheesecake",
vegetarian:false,
price:19.99)
);
dinners.add(new MenuItem(
name:"Hotdogs",
vegetarian:false,
price:6.05)
);
dinners.add(new MenuItem(
name:"Spaghetti ",
vegetarian:true,
price:30.89)
);
//The client does not distinguish between item and composite
Client client = new Client(allMenus);
client.printMenu();
}
}

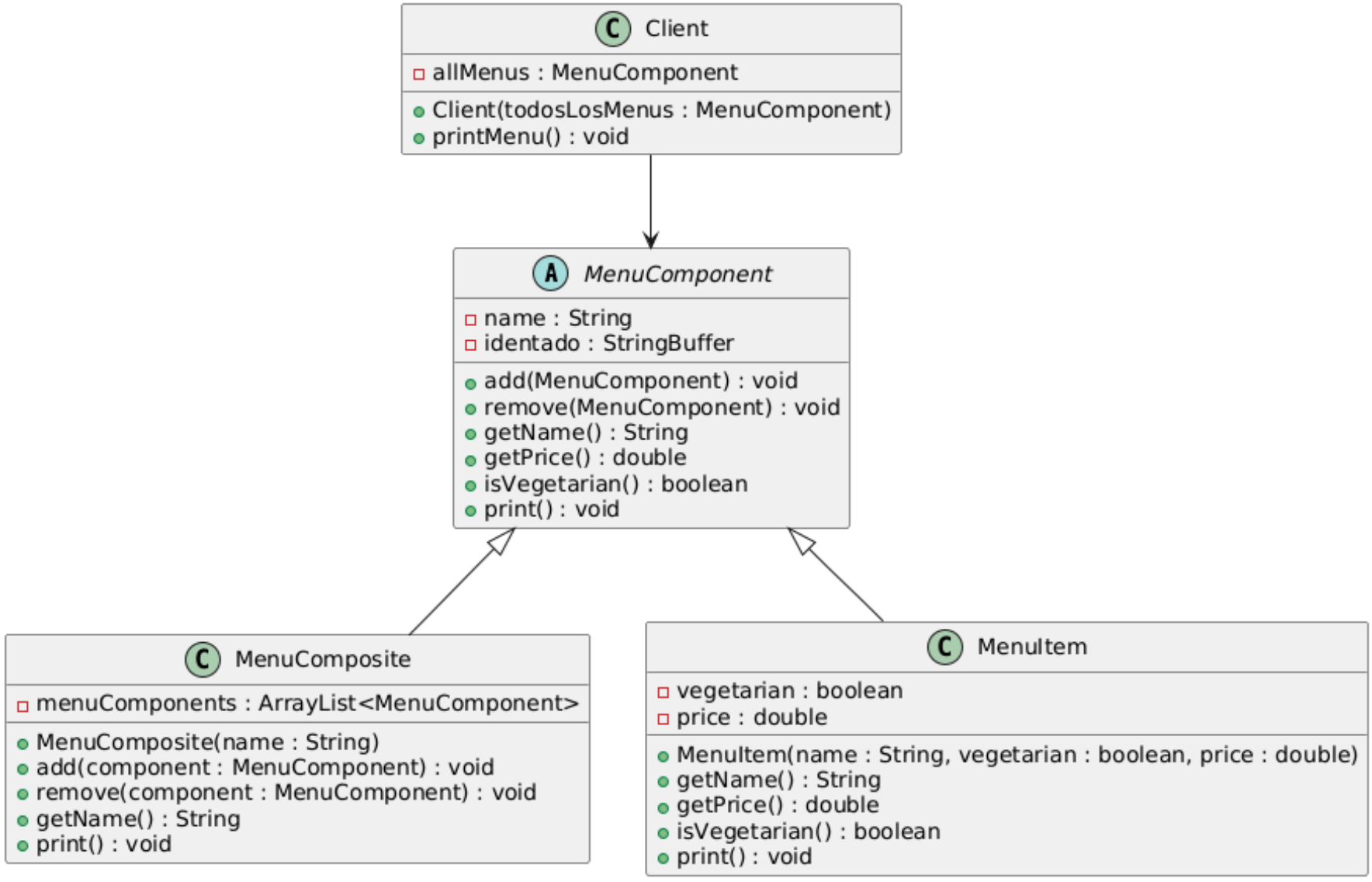
Análisis del Código y Uso del Patrón Composite

1. **Clases Clave**:
    - **`MenuComponent`**: Es la clase base abstracta, que define la estructura común para los elementos y las composiciones de menús, incluyendo métodos como `add()`, `remove()`, y `print()`, los cuales son sobrescritos en subclases según sea necesario. Métodos como `getPrice()` y `isVegetarian()` lanzan una excepción en esta clase base porque pueden no aplicarse a todos los componentes (e.g., un menú de varios elementos no tiene un precio único).
    - **`MenuComposite`**: Esta clase representa una composición de `MenuComponent` (es decir, un grupo de componentes). Implementa los métodos `add()`, `remove()`, y `print()` para gestionar y mostrar todos los componentes de forma jerárquica.
    - **`MenuItem`**: Representa un elemento del menú que tiene atributos específicos como `price` y `vegetarian`. Implementa los métodos `getPrice()`, `isVegetarian()` y `print()` para mostrar sus propios detalles.
2. **Cliente (`Client`)**: El cliente trabaja con la jerarquía de `MenuComponent`, llamando al método `print()` en `allMenus`. Dado que `allMenus` puede contener tanto elementos individuales (`MenuItem`) como compuestos (`MenuComposite`), no necesita diferenciar entre ellos; simplemente llama a `print()` y el sistema determina la lógica.

Diferencia con un Enfoque Tradicional de POO

En un enfoque tradicional de POO, cada tipo de objeto (e.g., un menú o un elemento de menú) podría tener métodos únicos, y el cliente tendría que conocer la clase exacta de cada objeto para manipularlo adecuadamente. Sin el patrón Composite, el cliente necesitaría realizar verificaciones condicionales para determinar si está trabajando con un menú o un elemento antes de llamarlos, lo que resultaría en un código más complejo y difícil de mantener. El uso del patrón Composite permite evitar este problema al ofrecer una interfaz uniforme.

DIAGRAMA UML



Explicación del Diagrama UML

`MenuComponent`: Clase abstracta que define la interfaz común para todos los componentes. Las clases `MenuComposite` y `MenuItem` heredan de esta clase.

`MenuComposite`: Esta clase es una colección de `MenuComponent` y, por lo tanto, puede agregar o eliminar componentes.

`MenuItem`: Representa elementos individuales en el menú, como platos específicos, con atributos adicionales (`price` y `vegetarian`).

`Client`: Contiene una referencia al componente raíz `MenuComponent` y llama a `printMenu()` para mostrar el menú.