PATRONES DE DISEÑO

ANDRES DAVID ORJUELA GRACIA

(20152020923)

DANIEL SIERRA

(20161020105)

MODELOS DE PROGRAMACION 1

INGNIERIA DE SISTEMAS

FACULTAD DE INGENIERIA



Bogotá, Colombia

23 de noviembre del 2017

TABLA DE CONTENIDO

[1. INTRODUCCION 3](#_Toc499177990)

[2. OBJETIVOS 4](#_Toc499177991)

[2.1 Objetivo general 4](#_Toc499177992)

[2.2 Objetivos específicos 4](#_Toc499177993)

[3. ESCENARIO 5](#_Toc499177994)

[4. PATRONES CREACIONALES 6](#_Toc499177995)

[4.1 Abstract Factory 6](#_Toc499177996)

[4.2 Metodo FÁBRICA 8](#_Toc499177997)

[4.3 builder 10](#_Toc499177998)

[4.4 SINGLETON 12](#_Toc499177999)

[4.5 PROPOTYPE 14](#_Toc499178000)

[5. PATRONES DE COMPORTAMIENTO 15](#_Toc499178001)

[5.1 ADAPTER 15](#_Toc499178002)

[5.2 BRIDGE 17](#_Toc499178003)

[5.3 COMPOSITE 18](#_Toc499178004)

[5.4 DECORADOR 18](#_Toc499178005)

[5.5 FACADE 18](#_Toc499178006)

[5.6 PESO LIGERO 18](#_Toc499178007)

[5.7 PROXY 18](#_Toc499178008)

[6. PATRONES DE COMPORTAMIENTO 18](#_Toc499178009)

[6.1 PLANTILLA 18](#_Toc499178010)

[6.2 ESTRATEGIA 18](#_Toc499178011)

[6.3 CADENA DE RESPONSABILIDAD 18](#_Toc499178012)

[6.4 COMANDO 18](#_Toc499178013)

[6.5 INTERPRETER 18](#_Toc499178014)

[6.6 ITERATOR 18](#_Toc499178015)

[6.7 MEDIATOR 18](#_Toc499178016)

[6.8 MEMENTO 18](#_Toc499178017)

[6.9 OBSERVER 18](#_Toc499178018)

[6.10 STATE 18](#_Toc499178019)

[6.11 VISITOr 18](#_Toc499178020)

[7. CONCLUSIONES 18](#_Toc499178021)

[8. REFERENCIAS BIBLIOGRÁFICAS 19](#_Toc499178022)

# INTRODUCCION

Los patrones de diseño aportan una posible solución a un problema concreto, usualmente relacionado con la estructura básica de una aplicación. Teniendo en cuenta las características de la aplicación a desarrollar y el tipo de problemas detectados durante el diseño de dicha aplicación, es posible determinar qué patrón de diseño será el más apropiado para resolver dichos problemas; estos patrones están reunidos en tres grandes ramas: Creacionales (como su nombre lo dice se encargarán de la creación), estructurales (solucionan problemas de composición o agregación de clases y objetos) y finalmente los patrones de comportamiento (Relacionados con la asignación de responsabilidades entre clases y la colaboración entre objetos.)

# OBJETIVOS

## Objetivo general

Analizar el comportamiento e implementación de los patrones de diseño vistos durante la catedra.

## Objetivos específicos

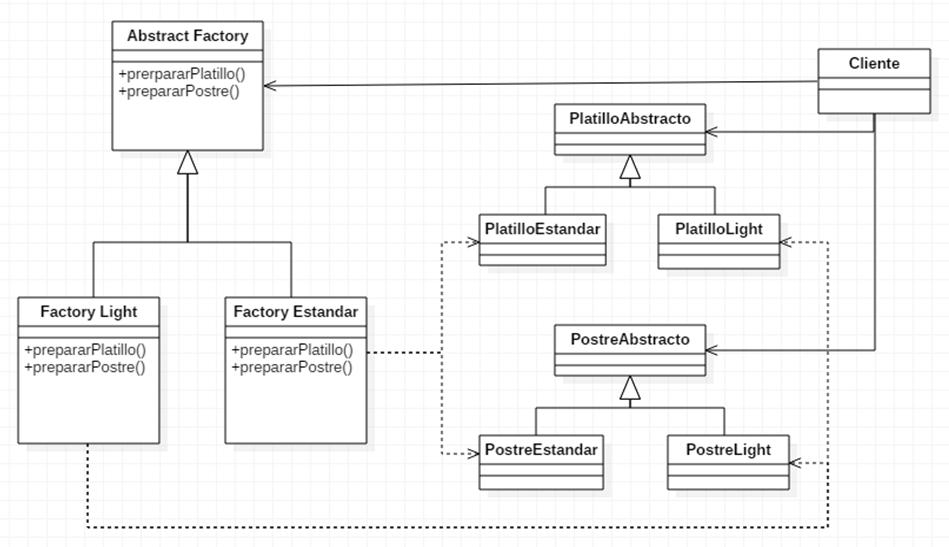
* Orientar la implementación de todos los patrones al caso de un restaurante para observar la aplicación estos patrones en un caso real.
* Identificar los beneficios y perjuicios de implementar estos patrones en casos similares.

# ESCENARIO

Nuestro contexto principal para implementar los patrones de diseño, es un restaurante, en el buscaremos aplicar los distintos tipos de patrones con el fin de resolver una situación específica que se acople a las condiciones de solución de cada patrón verificando de este modo cómo se comporta cada patrón aplicado a una situación de la vida real.

# PATRONES CREACIONALES

## Abstract Factory



*public class Cliente {*

*public static void main(String[] args) {*

FabricaAbstracta fabrica;

boolean cerrar = false;

Vector<PlatilloAbstracto> platillosListos = new Vector<PlatilloAbstracto>();

Vector<PostreAbstracto> postresListos = new Vector<PostreAbstracto>();

while (cerrar == false) {

System.out.println("-------------------------");

System.out.println("BIENVENIDO AL RESTAURANTE");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

String orden = JOptionPane.showInputDialog(null, "Orden: (1)Platillo (2)Postre: ");

String nombre = JOptionPane.showInputDialog(null, "Nombre Del Plato: ");

String tipo = JOptionPane.showInputDialog(null, "Tipo De Preparacion: (1)Normal (2)Light :");

int cantidad = Integer.parseInt(JOptionPane.showInputDialog(null, "Cantidad: # "));

switch (tipo) {

case "1":

fabrica = new FabricaEstandar();

if ("1".equals(orden)) {

while (cantidad > 0) {

platillosListos.addElement(fabrica.crearPlatillo(nombre));

System.out.println("Platillo Listo: " + nombre);

cantidad--;

}

}

if ("2".equals(orden)) {

while (cantidad > 0) {

postresListos.addElement(fabrica.crearPostre(nombre));

System.out.println("Postre Listo: " + nombre);

cantidad--;

}

}

break;

case "2":

fabrica = new FabricaLight();

if ("1".equals(orden)) {

while (cantidad > 0) {

platillosListos.addElement(fabrica.crearPlatillo(nombre));

System.out.println("Platillo Listo: " + nombre+"-Light");

cantidad--;

}

}

if ("2".equals(orden)) {

while (cantidad > 0) {

postresListos.addElement(fabrica.crearPostre(nombre));

System.out.println("Postre Listo: " + nombre+"-Light");

cantidad--;

}

}

break;

default:

System.out.println("Opcion NO Valida...");

break;

}

String exit = JOptionPane.showInputDialog("Nueva Orden: (1)SI (2)NO");

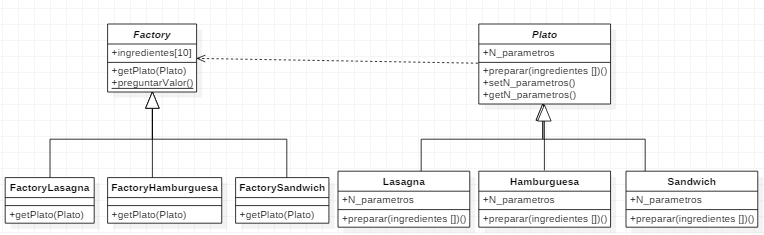
cerrar = !"1".equals(exit);

}

}

}

## Metodo FÁBRICA



public class ClienteRestaurante {

public static void main(String[] args) {

Factory Fac = new Factory() {};

int tipo = preguntarProducto();

plato F = null;

switch(tipo) {

case 1:

Fac = new FactoryLasagna();

F= new Lasagna();

Escribir("Lasagna creado exitosamente!");

break;

case 2:

Fac = new FactoryHamburguesa();

F = new Hamburguesa();

Escribir("Hamburguesa creado exitosamente!");

break;

case 3:

Fac = new FactorySandwich();

F= new Sandwich();

Escribir("Sandwich creado exitosamente!");

break;

case 4:

Escribir("Cerrando programa.");

System.exit(0);

break;

default:

Escribir("Se ha ingresado una opción inválida.");

Escribir("ERROR!");

}

Escribir(Fac.getPlato(F));

}

public static int preguntarProducto() {

System.out.print(

"MENÚ DE OPCIONES\n"

+ "---- -- -------->\n"

+ "- Digite un numero ->\n"

+ "1. Solicitar crear una lasagna.\n"

+ "2. Solicitar crear una hamburguesa.\n"

+ "3. Solicitar crear una sandwich.\n"

+ "4. Cerrar programa.\n"

+ "Seleccione opción... "

);

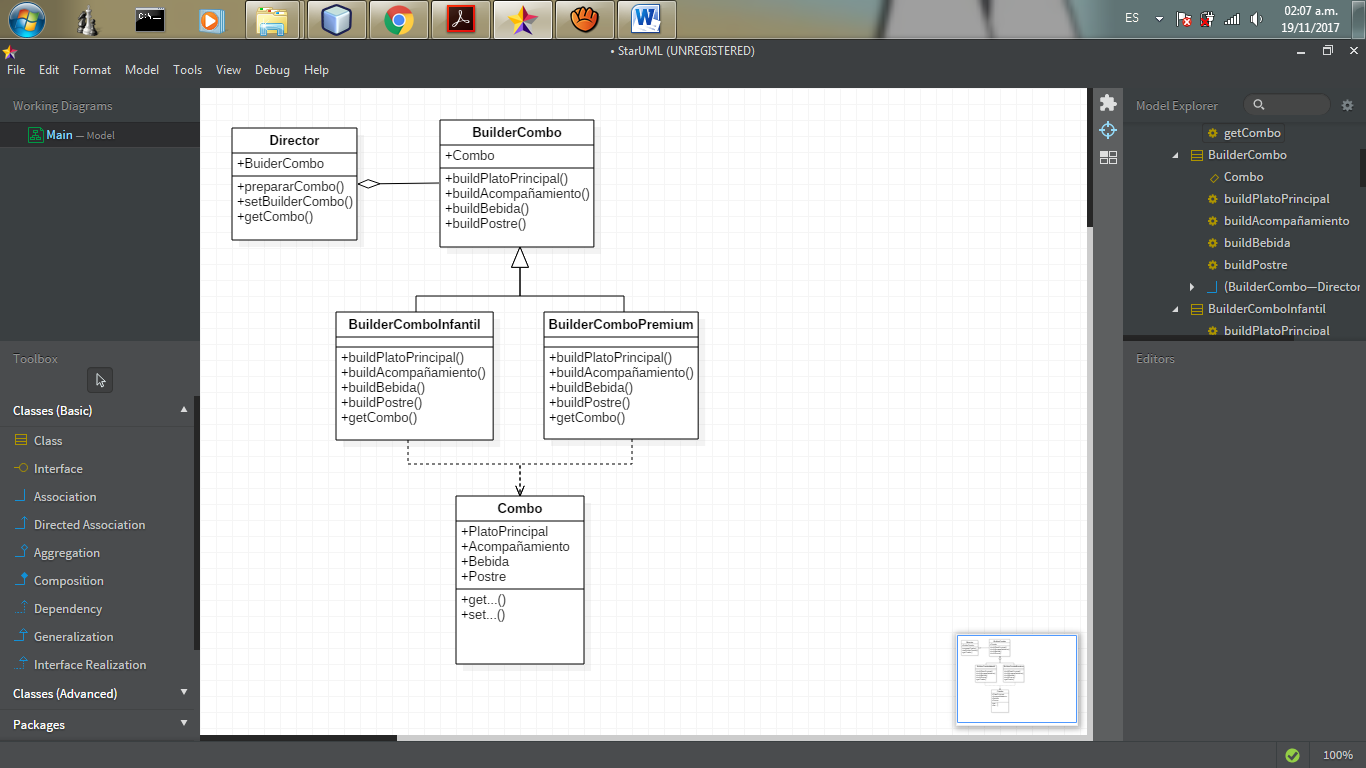
String s = leer();

return Integer.parseInt( s );

}

}

## builder



public class Cliente {

public static void main(String[] args) {

boolean cerrar = false;

String tipoCombo;

DirectorCocina director = new DirectorCocina();

Combo combo;

while (cerrar == false) {

System.out.println("-------------------------");

System.out.println("BIENVENIDO AL RESTAURANTE");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

tipoCombo = JOptionPane.showInputDialog(null, "Tipo De Combo: (1)Premium (2)Infantil: ");

switch (tipoCombo) {

case "1":

director.setBuilderCombo(new BuilderComboPremium());

director.prepararCombo();

combo = director.getCombo();

System.out.println("PREPARADO COMBO PREMIUM");

System.out.println("PLATO PRINCIPAL: " + (combo.getPlatoPrincipal().getNombre())

+ "|| Tamaño: " + combo.getPlatoPrincipal().getTamaño());

System.out.println("ACOMPAÑAMIENTO: " + (combo.getAcompañamiento().getNombre())

+ "|| Tamaño: " + combo.getAcompañamiento().getTamaño());

System.out.println("BEBIDA: " + (combo.getBebida().getNombre())

+ "|| Tamaño: " + combo.getBebida().getTamaño());

System.out.println("POSTRE: " + (combo.getPostre().getNombre())

+ "|| Tamaño: " + combo.getPostre().getTamaño());

break;

case "2":

director.setBuilderCombo(new BuilderComboInfantil());

director.prepararCombo();

combo = director.getCombo();

System.out.println("PREPARADO COMBO INFANTIL");

System.out.println("PLATO PRINCIPAL: " + (combo.getPlatoPrincipal().getNombre())

+ "|| Tamaño: " + combo.getPlatoPrincipal().getTamaño());

System.out.println("ACOMPAÑAMIENTO: " + (combo.getAcompañamiento().getNombre())

+ "|| Tamaño: " + combo.getAcompañamiento().getTamaño());

System.out.println("BEBIDA: " + (combo.getBebida().getNombre())

+ "|| Tamaño: " + combo.getBebida().getTamaño());

System.out.println("POSTRE: " + (combo.getPostre().getNombre())

+ "|| Tamaño: " + combo.getPostre().getTamaño());

break;

default:

System.out.println("Opcion NO Valida...");

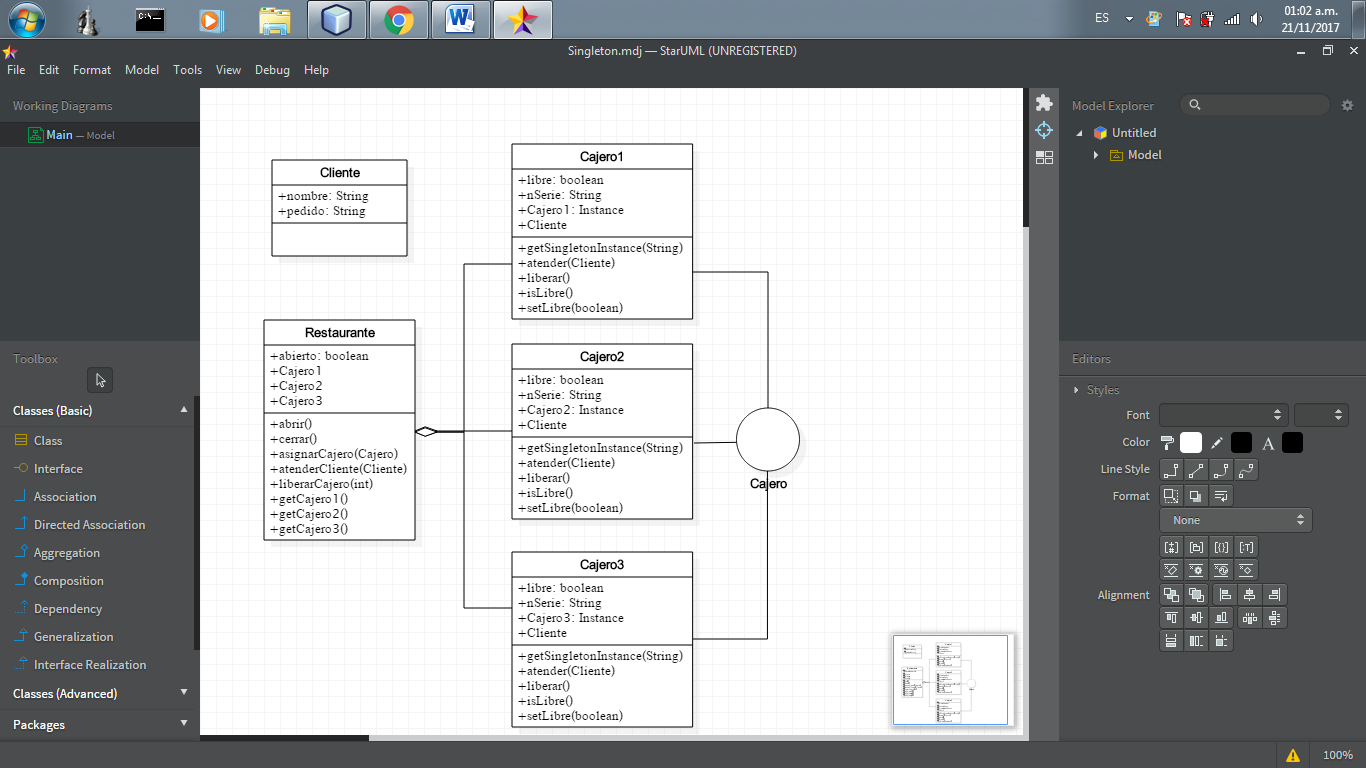
break;

}

String exit = JOptionPane.showInputDialog("Nueva Orden: (1)SI (2)NO");

cerrar = !"1".equals(exit); } } }

## SINGLETON



public class SingletonCajeros {

public static void main(String[] args) {

Restaurante restaurantePrueba = new Restaurante();

restaurantePrueba.abrir();

Cliente cliente;

String opcion;

int num;

while (restaurantePrueba.abierto == true) {

System.out.println("-------------------------");

System.out.println("BIENVENIDO AL RESTAURANTE");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("...CAJEROS DISPONIBLES...");

System.out.println("......................");

System.out.println("Cajero1: " + restaurantePrueba.getCajero1().isLibre());

System.out.println("Cajero2: " + restaurantePrueba.getCajero2().isLibre());

System.out.println("Cajero3: " + restaurantePrueba.getCajero3().isLibre());

System.out.println("......................");

opcion = JOptionPane.showInputDialog(null, "Digite Una Opcion:\n(1)NuevoCliente \n(2)LiberarCajero \n(3)Salir");

try {

switch (opcion) {

case "1":

String nombre = JOptionPane.showInputDialog(null, "Nombre Cliente: ");

cliente = new Cliente(nombre);

String orden = JOptionPane.showInputDialog(null, "Orden: ");

cliente.setPedido(orden);

System.out.println("Atendiendo...Cajero#"+restaurantePrueba.atenderCliente(cliente));

System.out.println("Cliente: "+nombre+"");

System.out.println("Orden: "+orden+"");

break;

case "2":

try {

num = Integer.parseInt(JOptionPane.showInputDialog(null, "Liberar Cajero #: "));

restaurantePrueba.liberarCajero(num);

System.out.println("Cajero #" + num + "Liberado...");

} catch (NumberFormatException ex) {

System.out.println("No Es Un Valor Valido");

}

break;

case "3":

restaurantePrueba.cerrar();

break;

default:

System.out.println("Opcion NO VALIDA");

break;

}

} catch (NumberFormatException ex) {

System.out.println("No Es Un Valor Valido");

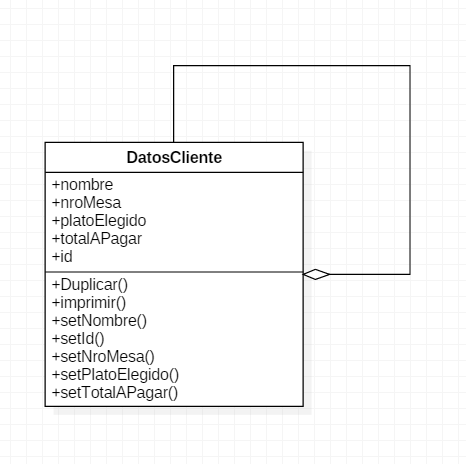
}

}

}

}

## PROPOTYPE



public class Propotype {

public static void main(String[] args) {

       int N\_documentos = 0;

       int itinerancia =0;

       ArrayList<DatosCliente> Doc = new ArrayList<DatosCliente>();

       Doc.add(itinerancia,new DatosCliente());

       JOptionPane.showMessageDialog(null,"Creando un nuevo cliente....");

       do{

       Doc.get(itinerancia).Imprimir();

       PreguntarModificar(Doc.get(itinerancia));

       if(Integer.parseInt(JOptionPane.showInputDialog(null, "1.Crear un nuevo documento"+"\n"+"2.Salir"))==1 ){

           Doc.add(itinerancia+1,Doc.get(itinerancia).duplicar());

           N\_documentos = Doc.size();

       }

       itinerancia++;

       }while(itinerancia < N\_documentos);

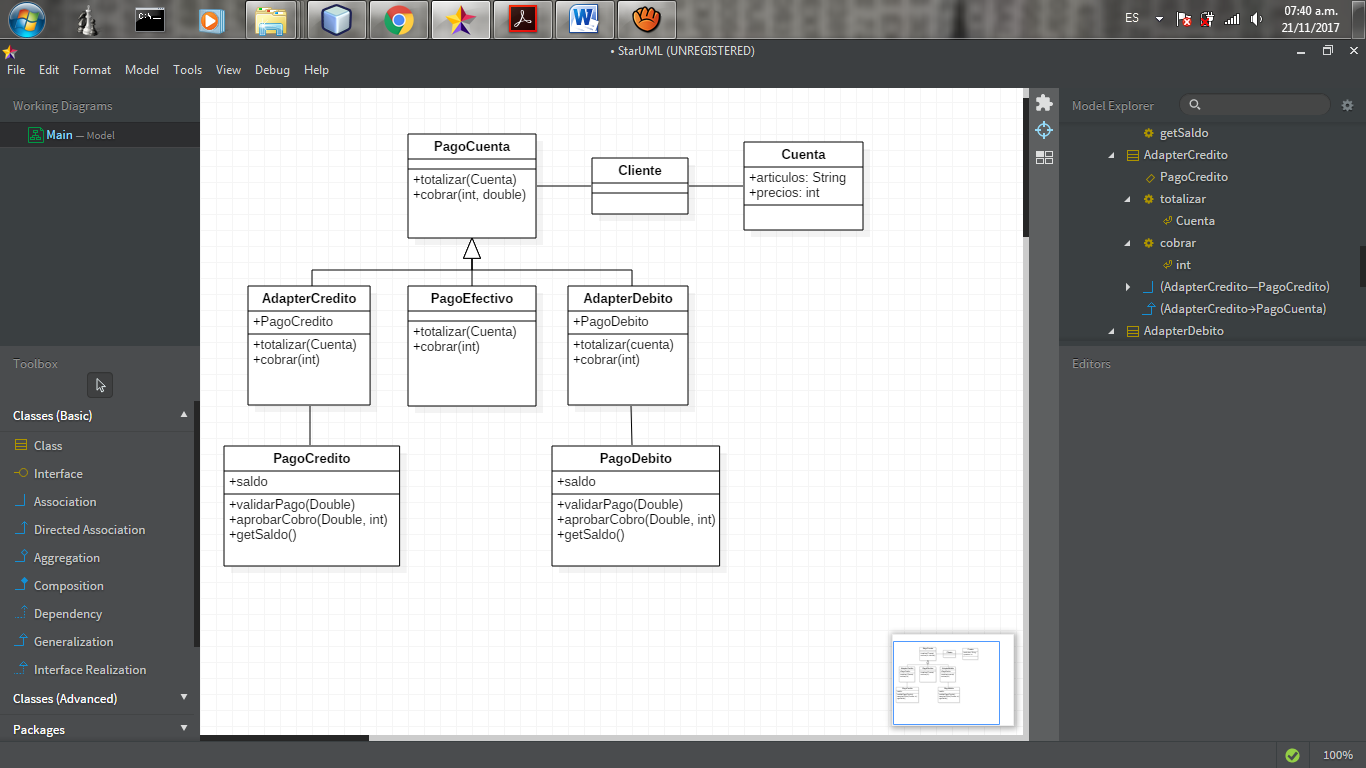
       PreguntarImprimir(Doc);

   }

}

# PATRONES DE COMPORTAMIENTO

## ADAPTER



public class Cliente {

public static void main(String[] args) {

PagoCuenta pagoCuenta;

Cuenta cuenta;

int opcion, cantidad, cuotas;

String producto, productos[];

int salir = 0;

double total, precio, precios[], saldo = 0;

while (salir != 1) {

System.out.println("-------------------------");

System.out.println("BIENVENIDO AL RESTAURANTE");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

cantidad = Integer.parseInt(JOptionPane.showInputDialog(null, "Pago De Cuenta \n"

+ "Digite cantidad de productos: \n"));

productos = new String[cantidad];

precios = new double[cantidad];

for (int i = 0; i < cantidad; i++) {

producto = JOptionPane.showInputDialog(null, "Nombre Del Producto: ");

precio = Integer.parseInt(JOptionPane.showInputDialog(null, "Precio: "));

productos[i] = producto;

precios[i] = precio;

}

cuenta = new Cuenta(productos, precios);

boolean cancelado = false;

do {

opcion = Integer.parseInt(JOptionPane.showInputDialog(null, "Tipo DE Pago: " + "\n"

+ "1.Efectivo" + "\n"

+ "2.Tarjeta De Credito" + "\n"

+ "3.Tarjeta Debito" + "\n"

+ "4.Salir" + "\n"));

switch (opcion) {

case 1:

try {

saldo = Integer.parseInt(JOptionPane.showInputDialog(null, "Efectivo: "));

pagoCuenta = new PagoEfectivo(saldo);

total = pagoCuenta.totalizar(cuenta);

pagoCuenta.cobrar(total, 1);

cuenta.imprimirCuenta();

cancelado = true;

System.out.println("Efectivo:" + saldo);

System.out.println("Cambio:" + pagoCuenta.getEstado());

} catch (Error e) {

JOptionPane.showMessageDialog(null, "Intentelo De Nuevo... Error:" + e);

cancelado = false;

}

break;

case 2:

try {

saldo = Integer.parseInt(JOptionPane.showInputDialog(null, "Saldo: "));

pagoCuenta = new AdapterCredito(saldo);

total = pagoCuenta.totalizar(cuenta);

cuotas = Integer.parseInt(JOptionPane.showInputDialog(null, "# Cuotas: "));

pagoCuenta.cobrar(total, cuotas);

cuenta.imprimirCuenta();

cancelado = true;

System.out.println("Nuevo Saldo:" + pagoCuenta.getEstado());

} catch (Error e) {

JOptionPane.showMessageDialog(null, "Intentelo De Nuevo... Error:" + e);

cancelado = false;

}

break;

case 3:

try {

saldo = Integer.parseInt(JOptionPane.showInputDialog(null, "Saldo: "));

pagoCuenta = new AdapterDebito(saldo);

total = pagoCuenta.totalizar(cuenta);

pagoCuenta.cobrar(total, 1);

cuenta.imprimirCuenta();

cancelado = true;

System.out.println("Nuevo Saldo:" + pagoCuenta.getEstado());

} catch (Error e) {

JOptionPane.showMessageDialog(null, "Intentelo De Nuevo... Error:" + e);

cancelado = false;

}

break;

case 4:

salir = 1;

break;

}

} while (cancelado == false);

}

}

}

## Bridge

## COMPOSITE

## DECORADOR

## FACADE

## PESO LIGERO

## PROXY

# PATRONES DE COMPORTAMIENTO

## PLANTILLA

## ESTRATEGIA

## CADENA DE RESPONSABILIDAD

## COMANDO

## INTERPRETER

## ITERATOR

## MEDIATOR

## MEMENTO

## OBSERVER

## STATE

## VISITOr

# CONCLUSIONES

# REFERENCIAS BIBLIOGRÁFICAS

* http://www.juntadeandalucia.es/servicios/madeja/contenido/subsistemas/desarrollo/patrones-diseno