**CICLO FORMATIVO DESARROLLO DE APLICACIONES WEB**

**Módulo de Proyecto**

**GESTION DE UNA PESCADERIA**

**AUTOR/A:**

**Raúl Carretero Randez**

**TUTOR/A:**

**Emiliano Torres Martinez**

**TRIBUNAL:**

|  |  |
| --- | --- |
| PRESIDENTE: | VOCAL 1: |
| VOCAL 2: | **FECHA DEFENSA:**  **CALIFICACIÓN:** |

**CONTENIDOS MEMORIA PROYECTO CICLO DAW**

1. Introducción

1.1 Justificación

1.2 Objetivos

2. Especificación de requisitos software

2.1 Propósito

2.2 Descripción general

2.3 Requisitos funcionales

2.4 Requisitos no funcionales

3. Fase de análisis

3.1 Diagrama de casos de uso

3.2 Casos de uso

3.3 Diagrama de Secuencias General del Sistema

3.4 Diagrama de Secuencias

3.5 Diagrama de Actividad

3.6 Diagrama de comunicacion

3.7 Diagrama de estados

4. Fase de diseño

4.1 Capa de presentación

4.2 Capa de persistencia

4.3 Capa de lògica

5. Detalles de implementación

6. Pruebas

7. Conclusiones y futuras ampliaciones

8. Bibliografía

*En algun caso es posible que alguno de estos apartados pueda quedar en blanco, por la propia naturaleza del proyecto.*

1. Introducción

(explicar el tipo de proyecto que has hecho, sobre que es)

* 1. *Justificación*

(por que quieres hacer este proyecto)

* 1. *Objetivos*

(que queremos hacer con la aplicacion , resolviendo las cosas de la justificacion )

1. Especificación de requisitos software

(explicacion genèrica de que son los requisitos software)

* 1. *Propósito*

(por que quiero añadir esos requisitos previos)

* 1. *Descripción general*

(los motivos por el por que quiero añadir esos requisitos)

* 1. *Requisitos funcionales*

(mas de 10)

* 1. *Requisitos no funcionales*

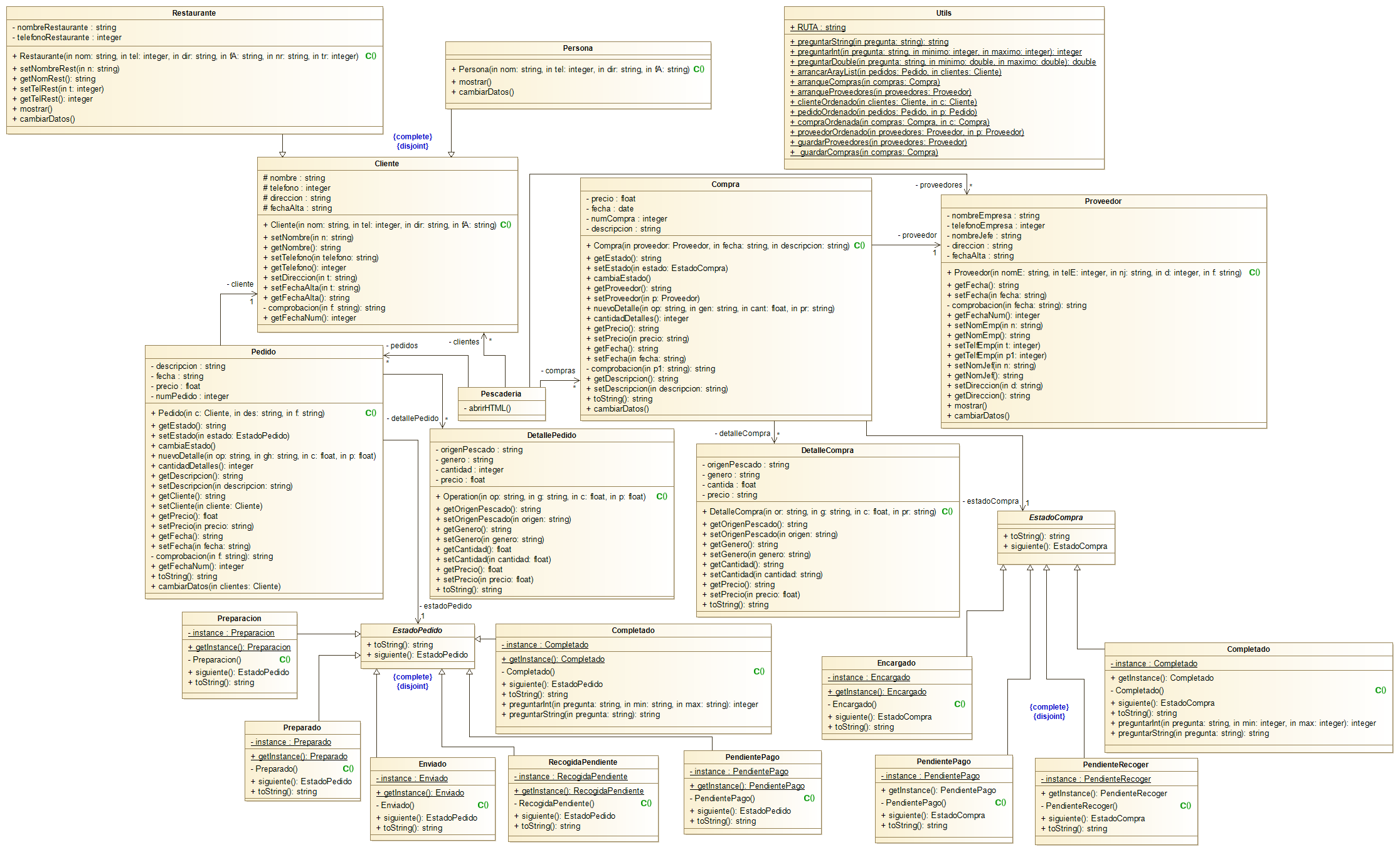
(mas de 10)

1. Fase de anàlisis
   1. *Diagrama de clases*

En este apartado mostraremos los diagrames en los cuales se organiza el proyecto por completo, del cual se basara todo el codigo java.

En terminos generales, el diagrama de clases recoge las clases de objetos y sus asociaciones. En este diagrama se representa la estructura y el comportamiento de cada uno de los objetos del sistema y sus relaciones con los demás objetos, pero no muestra información temporal.

Diagrama de clases con todos sus atributos y metodos de la Pescaderia:



Lo que debemos recalcar de esta imagen es que pescaderia es el main del proyecto, del cual tendremos una coleccion de Pedidos, Compras, Clientes y Proveedores.

Toda la gestion del proyecto se basara en el tratamiento de esas clases para su tratamiento de datos y registro.

Despues tenemos las interfases EstadoPedido y EstadoCompra que seran la implementacion de las clases cuyos estados hagan referencia a un pedido o a una compra

Otra cosa a recalcar es la classe Utils, que es nuestro contenidor de metodos para vaciar y tener un main mas limpio, tambien para que todas las clases puedan implementar sus cualquiera de sus metodos.

* 1. *Casos de uso*

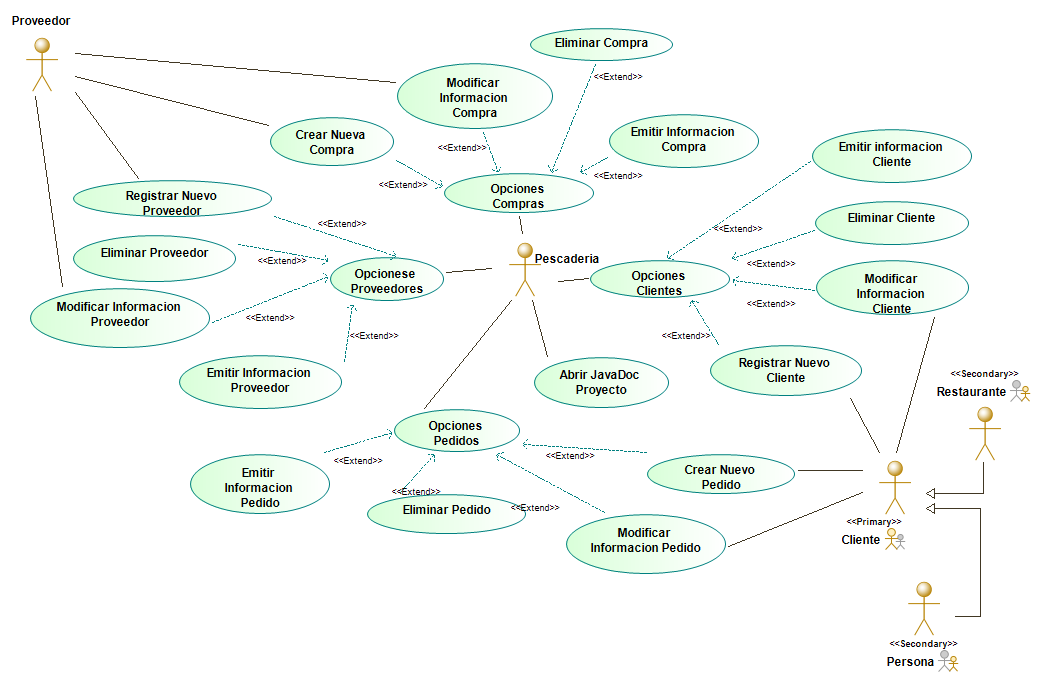
En este apartado, en primer lugar, veremos los diagramas de casos de uso para luego

hacer una descripción más detallada mediante una ficha por cada caso de uso

Lo que vamos a mostrar a continuacion, que siendo muy explicito, es el diagrama que representa las acciones que tendra nuestro proyecto.

Que hace el diagrama de casos de uso:

* El caso de uso debe describir qué debe hacer el sistema a desarrollar en su interacción con los actores y no cómo debe hacerlo.
* El nombre del caso de uso debe ilustrar el objetivo que pretende alcanzar el actor al realizarlo.

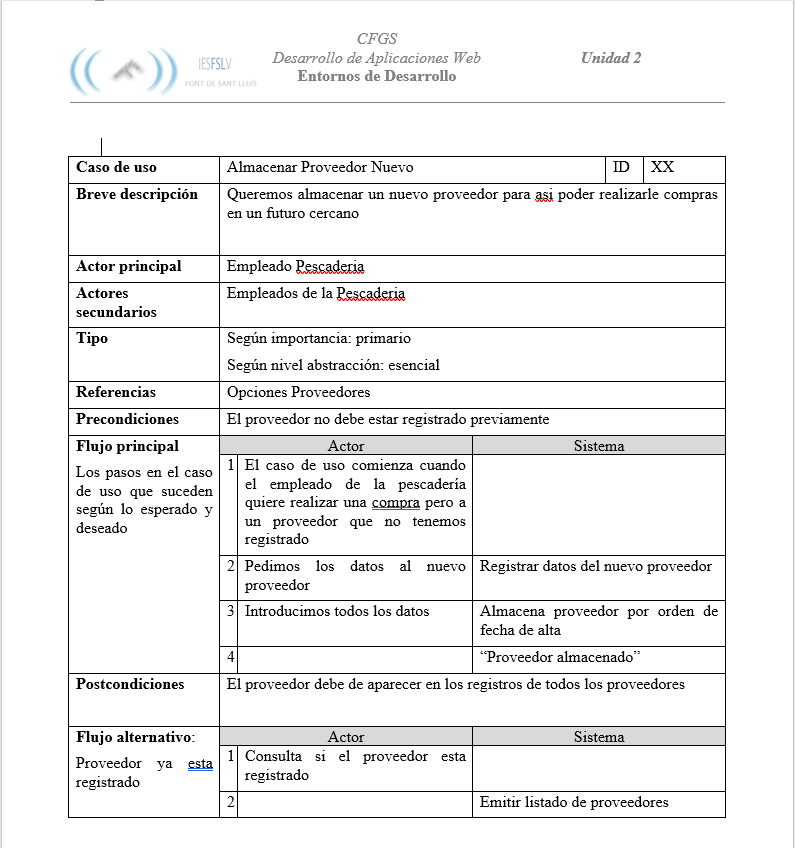
Diagrama de casos de uso de la Pescaderia:

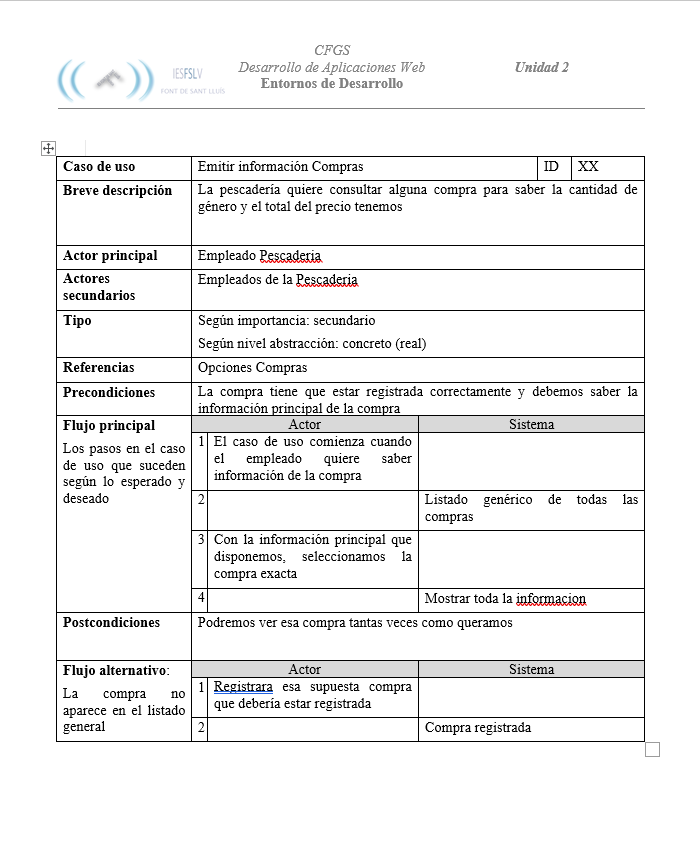
Todo diagrama de casos de uso lleva consigo unes tablas de especificacion , que se encargan de explicar mas detalladamente

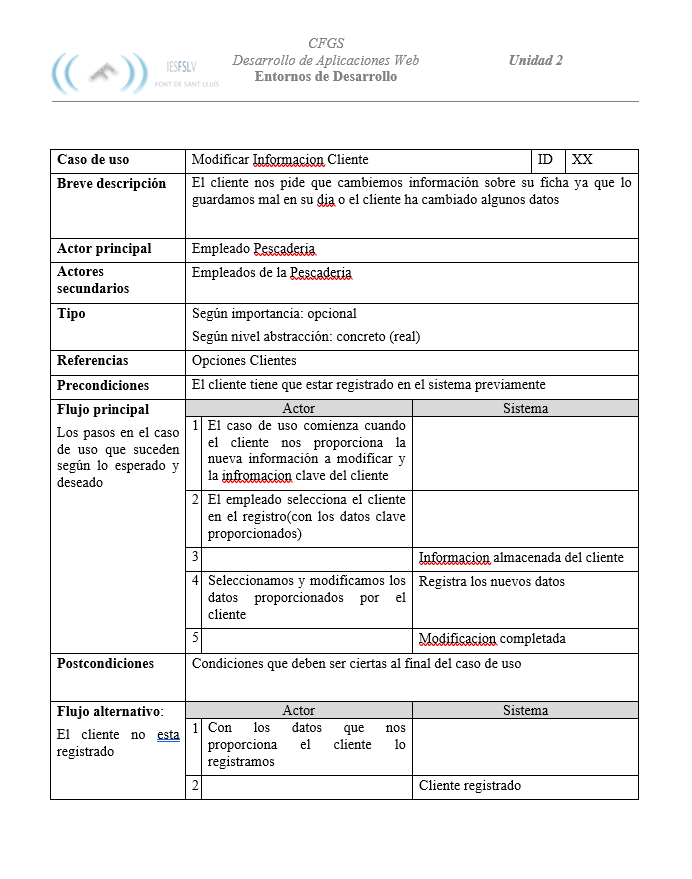
¿Qué son las especificaciones de casos de uso?

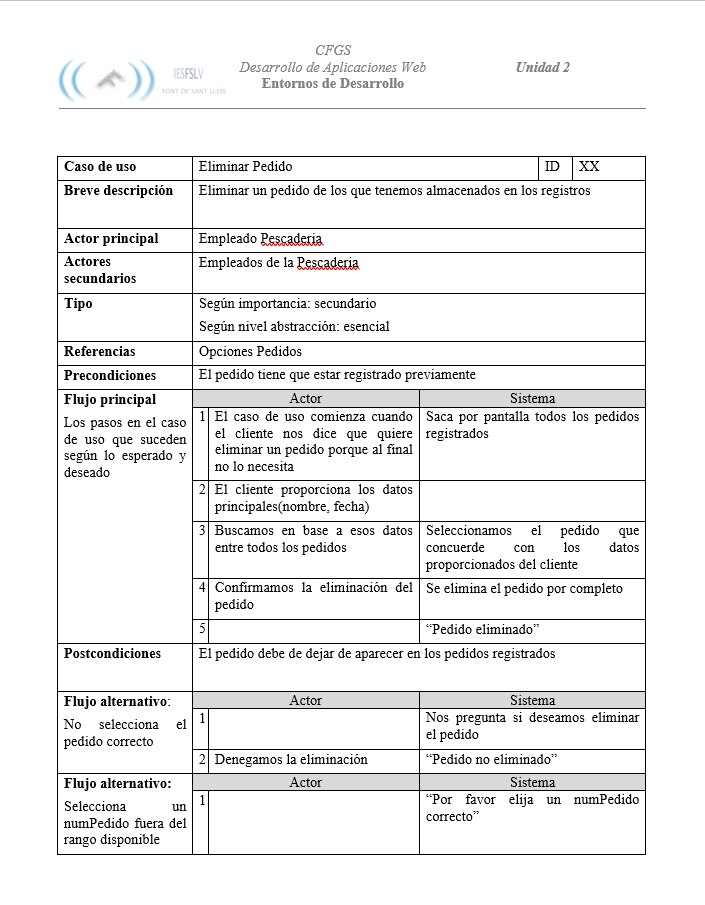
Una especificación de caso de uso proporciona detalles textuales de un caso de uso. Se proporciona una descripción de ejemplo de una especificación de caso de uso. Puede reutilizar y modificar la descripción según se requiera en una especificación de caso de uso. Indica el nombre del caso de uso.

A continuacion las tablas de esecificacion de casos de uso mas genericos del protecto:









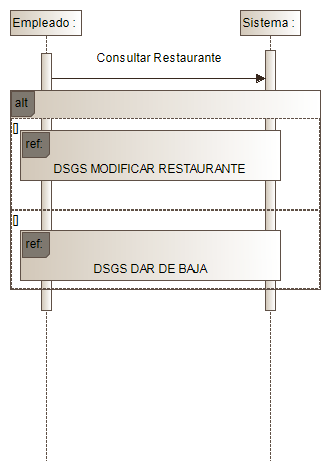
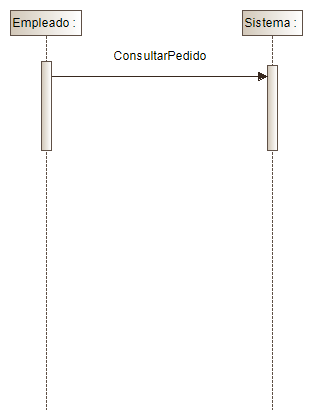
* 1. *Diagrama de Secuencias General del Sistema*

Los diagramas de secuencia muestran interacciones entre líneas de vida como una

secuencia ordenada de tiempo de eventos.

Empieza esbozando una realización de caso de uso, despues realizamos la demostracion de todas las acciones que conlleva ese caso de uso.

A continuacion mostraremos los DSGS de los casos de uso mas genericos del proyecto para que abarquen la mayor informacion posible.

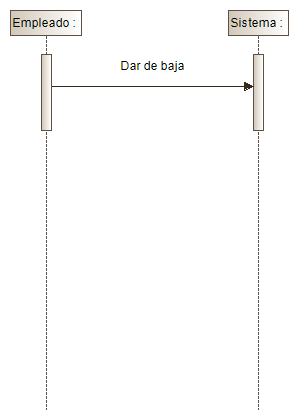
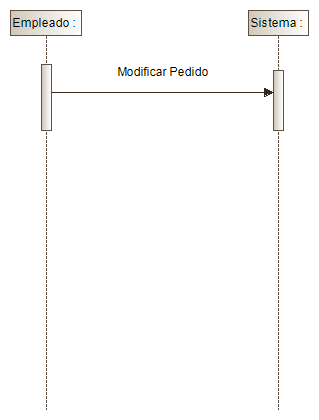
**

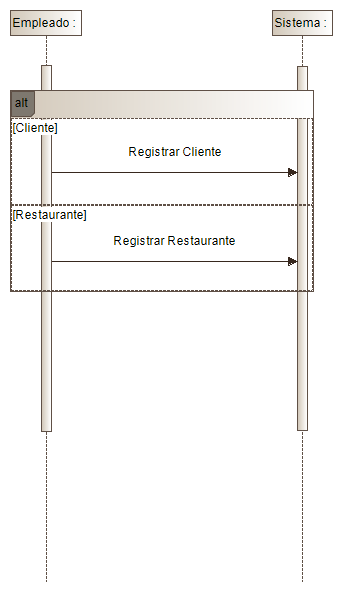
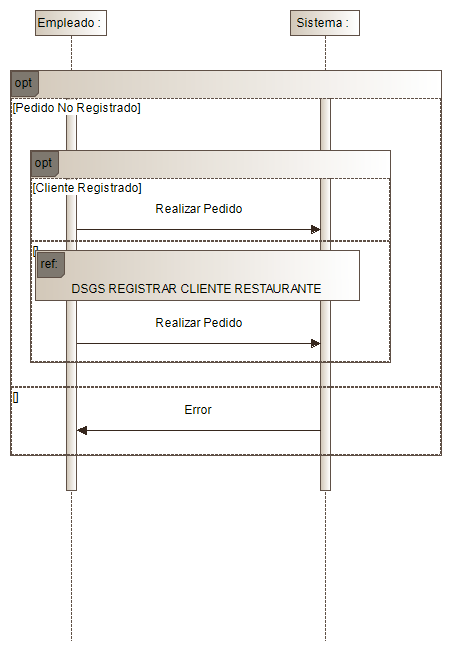
DSGS CONSULTAR PEDIDO

DSGS CONSULTAR RESTAURANTE

DSGS MODIFICAR PEDIDO

DSGS DAR DE BAJA RESTAURANTE

**

**

DSGS TOMAR PEDIDO CLIENTE

DSGS REGISTRAR CLIENTE

* 1. *Diagrama de Secuencias*

Un diagrama de secuencia es un gráfico bidimensional donde el eje vertical representa el tiempo, el eje horizontal muestra objetos del sistema y la interacción entre los objetos se representa mediante flechas que van de unos objetos a otros, ordenadas cronológicamente de arriba abajo. Son un tipo de diagramas de interacción, que a su vez es una categoría de diagramas de comportamiento.

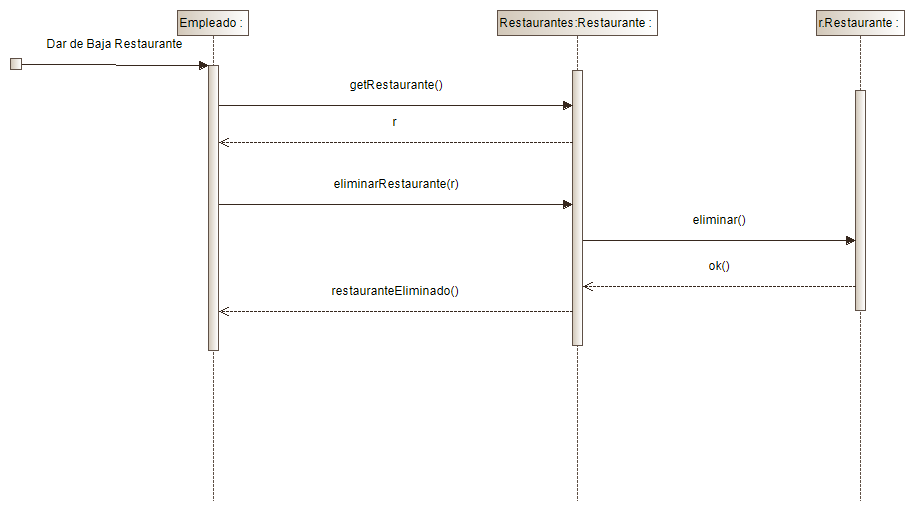
Los diagramas de secuencia se utilizan cuando queremos expresar qué objetos se relacionan con qué objetos, enfatizando el orden en que lo hacen y qué tipo de mensajes se envían entre sí.

También permiten expresar estructuras de control (condiciones y repeticiones), pero los diagramas de secuencia no están pensados para eso y debemos evitar incluir lógica procedimental compleja en ellos.

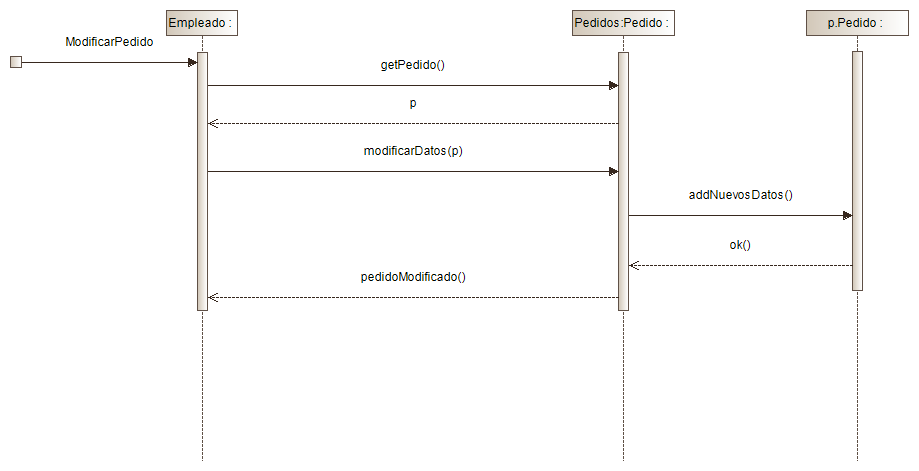
Conslusion, son los diagrames que detallan auhn mas los casos de uso, ya que son diagrames que detallan las acciones de los DSGS.

A continuacion mostraremos los DS de los DSGS mas genericos (de los casos de uso) del proyecto para que abarquen la mayor informacion posible.

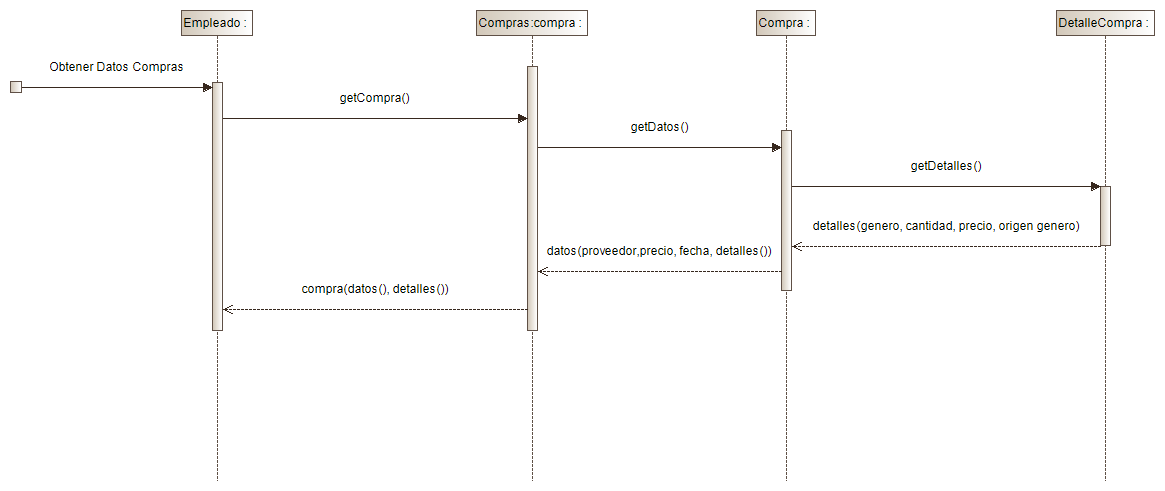
DS DAR DE BAJA RESTAURANTE



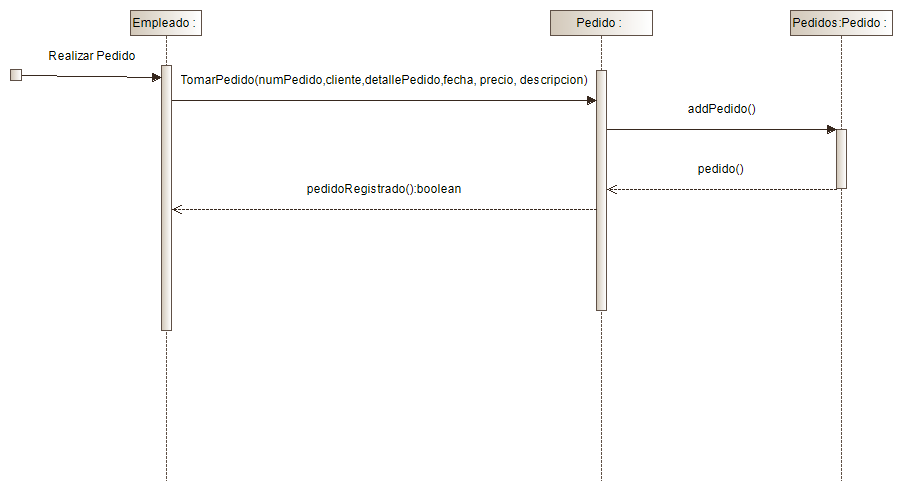
DS MODIFICAR PEDIDO



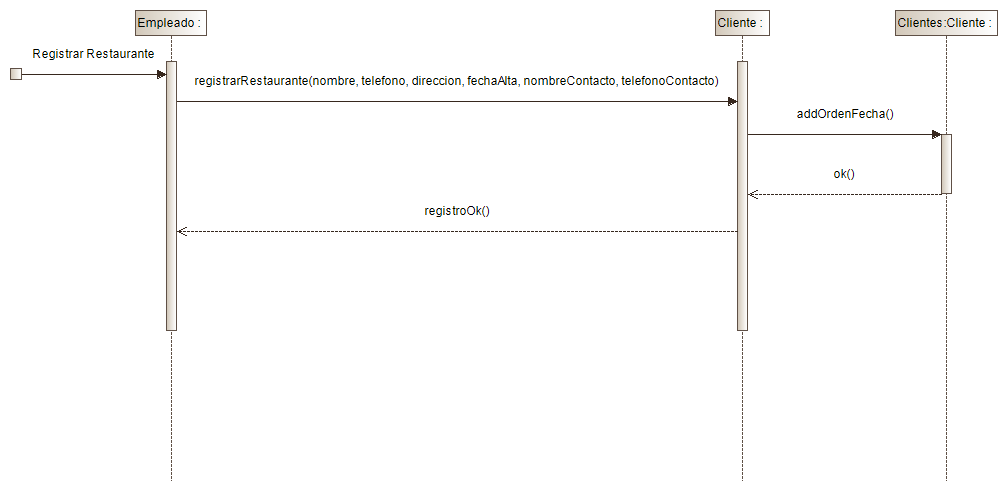
DS OBTENER DATOS COMPRAS



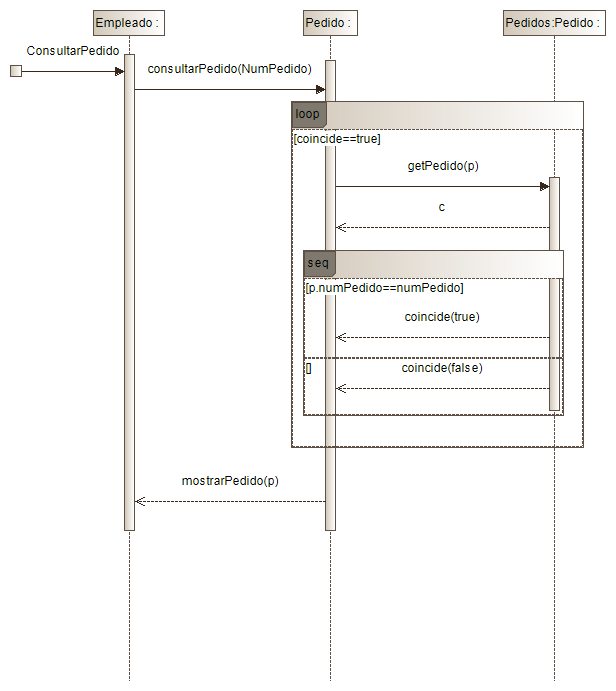
DS REALIZAR PEDIDO



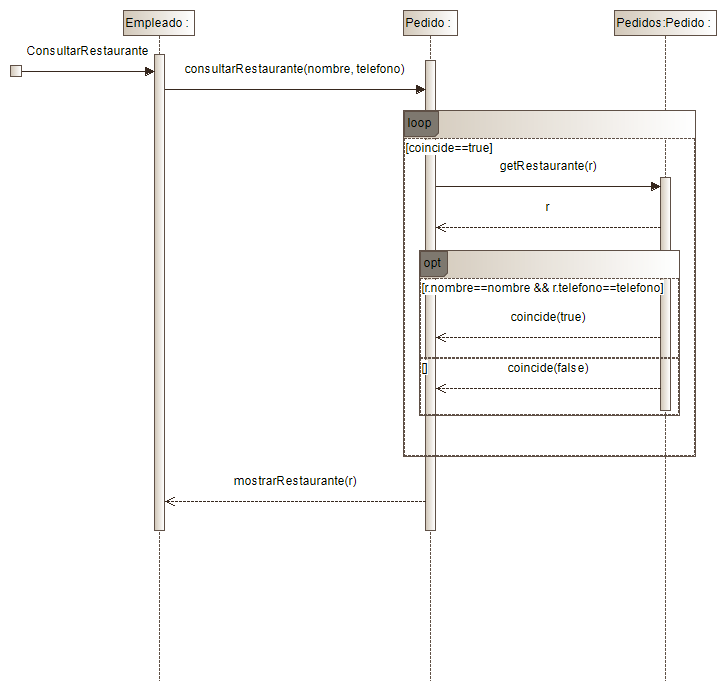
DS REGISTRAR RESTAURANTE



DS CONSULTAR PEDIDO



DS CONSULTAR RESTAURANTE



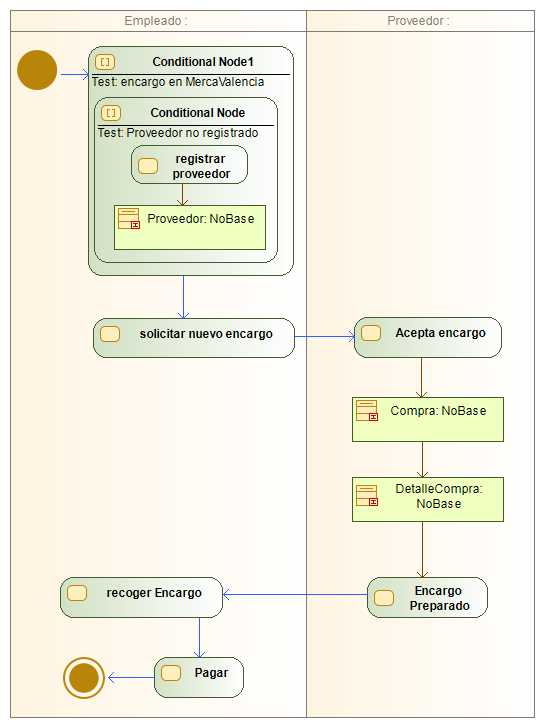
* 1. *Diagrama de Actividad*

Un diagrama de actividad UML enVisio parece un diagrama de flujo. El flujo de control se desencadena al completar acciones (o actividades)dentro del sistema. El flujo puede ser secuencial, simultáneo o ramificado, indicado por formas como calles, bifurcaciones y combinaciones.

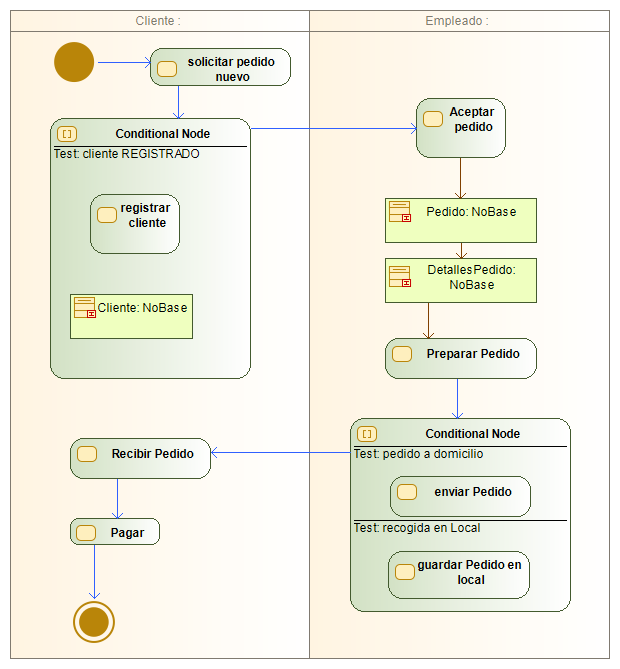
Conslusion, es un diagrama que definí la actividad que seguirà el programa para realitzar una accion concreta.

A continuacion los DA del proyecto de la pescaderia:

DA COMPRAS



DA PEDIDOS

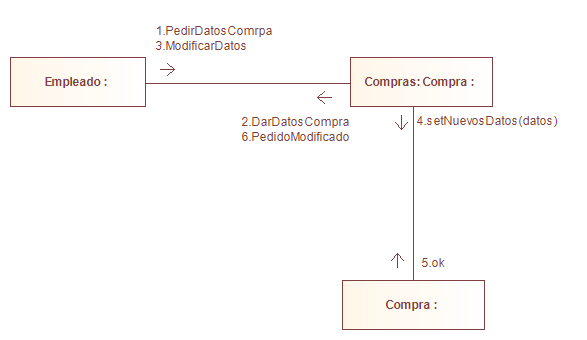


* 1. *Diagrama de comunicacion*

Un diagrama de comunicación es una forma de representar interacción entre objetos, alterna al diagrama de secuencia. Es un diagrama de clases que contiene roles de clasificador y los roles de asociación en lugar de solo clasificadores y asociaciones.

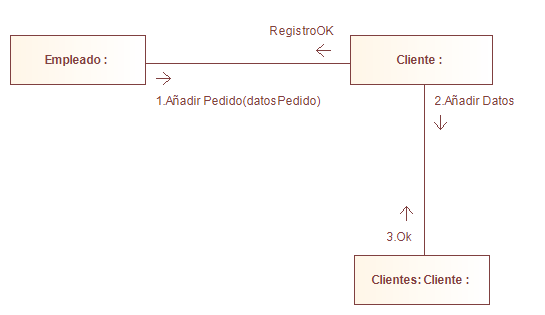
Conclusio, es un diagrama en el que se ven las llamadas que se hacen entre clases o componentes del proyecto en un diagrama de secuencia

A continuacion los DCom del proyecto Pescaderia:



DCom MODIFICAR COMRPA

DCom REGISTRAR PEDIDO



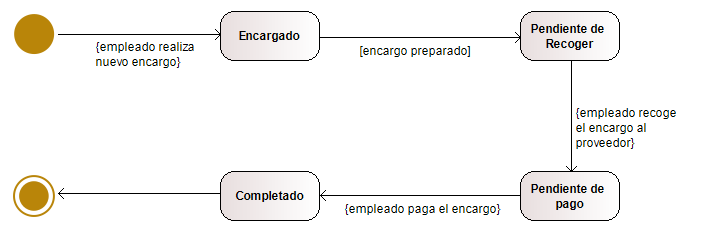
* 1. *Diagrama de estados*

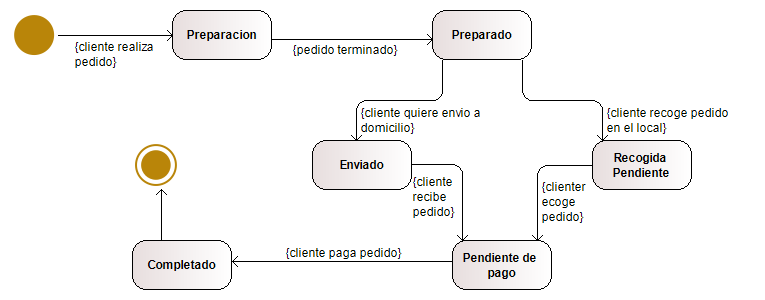
Un diagrama de estados, en ocasiones conocido como diagrama de máquina de estados, es un tipo de diagrama de comportamiento en el Lenguaje Unificado de Modelado (UML) que muestra transiciones entre diversos objetos.

Conclusion, diagrama en el eu establece los possibles estados de un objeto del proyecto y sus explicaciones entre estados.

A continuacion los DTE del proyecto Pescaderia:

DTE COMPRAS





DTE PEDIDOS

1. Fase de diseño
   1. *Capa de presentación*

(explicar la interfaz, en mi caso el cmd)

(captures del menu, de como se ve)

* 1. *Capa de logica*

En esta seccion lo que vamos a concretar es la refactorizacion del codigo, es decir, realizarle cambios en la estructura interna sin que el comportamiento externo se vea alterado.

Con esto conseguimos que el codigo sea mas legible, mas entendible en el caso de que un programador que se acaba de incorporar en el proyecto pueda entender la lògica de las funciones y del codigo, tambien que sea mas fàcil el mantenimiento y la ampliacion del proyecto en general

Refactorización a posteriori

En este apartado nos referimos con el término refactorización a posteriori a todas aquellos cambios estructurales que se realizan un tiempo después de la implementación de la funcionalidad existente. Existen varios motivos para encontrarse con esta situación. Algunos de ellos son:

* Un equipo comienza a trabajar con código desarrollado por un equipo anterior que o bien no tiene buena calidad o bien no está preparado para que se incorporen nuevas funcionalidades.
* Se ha aplicado refactorización continua pero la calidad del código o el diseño se ha degradado porque no se identificó a tiempo la necesidad de una refactorización o bien se equivocó la refactorización necesaria.
* REFACTORIZACION ESTADOS DE LA COMPRA

En este punto del código lo que queremos mejorar son los estados que pueda tener una compra. Cuando finalicemos un proceso que este dentro de cierto estado, ejecutaremos la función cambioEstado, para que se cambie al siguiente estado de la jerarquía.

**VERSION SIN REFACTORIZAR:**

public class Compra

{

private int estado;

static final private int ENCARGADO = 0;

static final private int PENDIENTE\_RECOGER = 1;

static final private int PENDIENTE\_PAGO = 2;

static final private int COMPLETADO = 3;

public void cambiaEstado()

{

if(estado== ENCARGADO) {

estado = PENDIENTE\_RECOGER;

}

else if(estado== PENDIENTE\_RECOGER) {

estado = PENDIENTE\_PAGO;

}

else if(estado== PENDIENTE\_PAGO) {

estado = COMPLETADO;

}

else if(estado== COMPLETADO) {

estado = ENCARGADO;

}

else {

throw new RuntimeException("Estado desconocido");

}

}

}

**VERSION REFACTORIZADA**

//INTERFAZ ESTADO\_COMPRA

public interface EstadoCompra

{

public EstadoCompra siguiente();

public String toString();

}

//CLASE ENCARGADO

public class Encargado implements EstadoCompra, Serializable

{

private static final Encargado instance=new Encargado();

public static Encargado getInstance()

{

return instance;

}

private Encargado(){}

public EstadoCompra siguiente()

{

EstadoCompra e=PendienteRecoger.getInstance();

String cadena=e+"";

System.out.println("Ahora el estado del pedido es:\n"+cadena.toUpperCase());

return e;

}

public String toString() {

return "Encargado";

}

}

//CLASE PENDIENTE\_RECOGER

public class PendienteRecoger implements EstadoCompra, Serializable

{

private static final PendienteRecoger instance=new PendienteRecoger();

public static PendienteRecoger getInstance()

{

return instance;

}

private PendienteRecoger(){}

public EstadoCompra siguiente()

{

EstadoCompra e=PendientePago.getInstance();

String cadena=e+"";

System.out.println("Ahora el estado del pedido es:\n"+cadena.toUpperCase());

return e;

}

public String toString() {

return "PendienteRecoger";

}

}

//CLASE PENDIENE\_PAGO

public class PendientePago implements EstadoCompra, Serializable

{

private static final PendientePago instance=new PendientePago();

public static PendientePago getInstance()

{

return instance;

}

private PendientePago(){}

public EstadoCompra siguiente()

{

EstadoCompra e=Completado.getInstance();

String cadena=e+"";

System.out.println("Ahora el estado del pedido es:\n"+cadena.toUpperCase());

return e;

}

public String toString() {

return "PendientePago";

}

}

//CLASE COMPLETADO

public class Completado implements EstadoCompra, Serializable

{

private static final Completado instance=new Completado();

public static Completado getInstance()

{

return instance;

}

private Completado(){}

public EstadoCompra siguiente()

{

EstadoCompra e = Completado.getInstance();

String cadena = "";

System.out.println("YA NO HAY MAS ESTADOS DE LA COMPRA");

System.out.println("1.Modificar el estado de la compra");

System.out.println("0.Nada, dejarlo como esta");

int n = preguntarInt("\nQUE DESEA HACER?",0,1);

if (n == 1) {

System.out.println("A QUE ESTADO DESEA ACTUALIZAR EL PEDIDO?");

System.out.println("1.Encargado");

System.out.println("2.Pendiente de Recoger");

System.out.println("3.Pendiente de pago");

System.out.println("0.SALIR");

int n1 = preguntarInt("Selecciona una accion",0,3);

switch (n1) {

case 1:

e = Encargado.getInstance();

break;

case 2:

e = PendienteRecoger.getInstance();

break;

case 3:

e = PendienteRecoger.getInstance();

break;

default:

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

}

}

cadena += e;

System.out.println("El estado actual de la compra es " + cadena.toUpperCase());

return e;

}

public String toString() {

return "Completado";

}

public static int preguntarInt(String pregunta, int min, int max) {

boolean repetir = true;

int resultado = -1;

do {

String valor = preguntarString(pregunta);

try {

resultado = Integer.parseInt(valor);

repetir = (resultado < min) || (resultado > max);

if (repetir) {

System.out.println("Error: "+resultado+" fuera de rango");

}

} catch(NumberFormatException e) {

System.out.println("Error: "+valor+" no es un numero Valido");

}

} while (repetir);

return resultado;

}

public static String preguntarString(String pregunta) {

System.out.print(pregunta+"\n ");

Scanner entrada = new Scanner(System.in);

return entrada.nextLine();

}

}

* REFACTORIZACION ESTADOS DEL PEDIDO

Estamos en una situación similar al que tenemos con las compras, pero en este caso con los pedidos que nos hacen los clientes, tenemos mas estados con lo cual hay que realizar mas código.

**VERSION SIN REFACTORIZAR:**

public class Pedido

{

private int estado;

static final private int PREPARACION = 0;

static final private int PREPARADO = 1;

static final private int ENVIADO = 2;

static final private int RECOGIDA\_PENDIENTE = 3;

static final private int PENDIENTE\_PAGO = 4;

static final private int COMPLETADO = 5;

public void cambiaEstado()

{

if(estado== PREPARACION) {

estado = PREPARADO;

}

else if(estado== PREPARADO) {

if(quiere domicilio)

estado = ENVIADO;

else

estado=RECOGIDA\_PENDIENTE;

}

else if(estado== ENVIADO) {

estado = PENDIENTE\_PAGO;

}

else if(estado== RECOGIDA\_PENDIENTE) {

estado = PENDIENTE\_PAGO;

}

else if(estado== PENDIENTE\_PAGO) {

estado = COMPETADO;

}

else if(estado== COMPLETADO) {

estado = PREPARACION;

}

else {

throw new RuntimeException("Estado desconocido");

}

}

}

**VERSION REFACTORIZADA**

//INTERFAZ ESTADO\_COMPRA

public interface EstadoPedido

{

public EstadoPedido siguiente();

public String toString();

}

//CLASE PREPARACION

public class Preparacion implements EstadoPedido

{

private static final Preparacion instance=new Preparacion();

public static Preparacion getInstance()

{

return instance;

}

private Preparacion(){}

public EstadoPedido siguiente()

{

EstadoPedido e= Preparado.getInstance();

String cadena=e+"";

System.out.println("Ahora el estado del pedido es:\n"+cadena.toUpperCase());

return e;

}

public String toString() {

return "Preparacion";

}

}

//CLASE PREPARADO

public class Preparado implements EstadoPedido

{

private static final Preparado instance=new Preparado();

public static Preparado getInstance()

{

return instance;

}

private Preparado(){}

public EstadoPedido siguiente()

{

EstadoPedido e =Preparado.getInstance();

int r;

String cadena="";

System.out.println("A Domicilio(0) o Recogida en General.Pescaderia(1)");

r=Completado.preguntarInt("\nSeleccione un numero (0 // 1)", 0,1);

switch (r) {

case 0:

e = Enviado.getInstance();

cadena += e;

System.out.println("Ahora el estado del pedido es:\n" + cadena.toUpperCase());

break;

case 1:

e = RecogidaPendiente.getInstance();

cadena += e;

System.out.println("Ahora el estado del pedido es:\n" + cadena.toUpperCase());

break;

default:

System.out.println("Por favor seleccionar la opcion correcta (0 // 1)");

}

return e;

}

public String toString() {

return "Preparado";

}

}

//CLASE ENVIADO

public class Enviado implements EstadoPedido

{

private static final Enviado instance=new Enviado();

public static Enviado getInstance()

{

return instance;

}

private Enviado(){}

public EstadoPedido siguiente()

{

EstadoPedido e= PendientePago.getInstance();

String cadena=e+"";

System.out.println("Ahora el estado del pedido es:\n"+cadena.toUpperCase());

return e;

}

public String toString() {

return "Enviado";

}

}

//CLASE RECOGIDA\_PENDIENTE

public class RecogidaPendiente implements EstadoPedido

{

private static final RecogidaPendiente instance=new RecogidaPendiente();

public static RecogidaPendiente getInstance()

{

return instance;

}

private RecogidaPendiente(){}

public EstadoPedido siguiente()

{

EstadoPedido e= PendientePago.getInstance();

String cadena=e+"";

System.out.println("Ahora el estado del pedido es:\n"+cadena.toUpperCase());

return e;

}

public String toString() {

return "RecogidaPendiente";

}

}

//CLASE PENDIENTE\_PAGO

public class PendientePago implements EstadoPedido

{

private static final PendientePago instance=new PendientePago();

public static PendientePago getInstance()

{

return instance;

}

private PendientePago(){}

public EstadoPedido siguiente()

{

EstadoPedido e= Completado.getInstance();

String cadena=e+"";

System.out.println("Ahora el estado del pedido es:\n"+cadena.toUpperCase());

return e;

}

public String toString() {

return "PendientePago";

}

}

//CLASE COMPLETADO

public class Completado implements EstadoPedido

{

private static final Completado instance=new Completado();

public static Completado getInstance()

{

return instance;

}

private Completado(){}

public EstadoPedido siguiente()

{

EstadoPedido e = Completado.getInstance();

String cadena = "";

System.out.println("YA NO HAY MAS ESTADOS DEL PEDIDO");

System.out.println("1.Modificar el estado del pedido");

System.out.println("0.Nada, dejarlo como esta");

int n = preguntarInt("QUE DESEA HACER?", 0,1);

if (n == 1)

{

System.out.println("A QUE ESTADO DESEA ACTUALIZAR EL PEDIDO?");

System.out.println("1.Preparacion");

System.out.println("2.Preparado");

System.out.println("3.Enviado");

System.out.println("4.Recogida Pendiente");

System.out.println("5.Pendiente de Pago");

System.out.println("0.SALIR");

int n1 = preguntarInt("Selecciona una accion",0,5);

switch (n1) {

case 1:

e = Preparacion.getInstance();

break;

case 2:

e = Preparado.getInstance();

break;

case 3:

e = Enviado.getInstance();

break;

case 4:

e = RecogidaPendiente.getInstance();

break;

case 5:

e = PendientePago.getInstance();

break;

default:

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

}

}

cadena += e;

System.out.println("El estado actual del pedido es " + cadena.toUpperCase());

return e;

}

public String toString() {

return "Completado";

}

public static int preguntarInt(String pregunta, int min, int max) {

boolean repetir = true;

int resultado = -1;

do {

String valor = preguntarString(pregunta);

try {

resultado = Integer.parseInt(valor);

repetir = (resultado < min) || (resultado > max);

if (repetir) {

System.out.println("Error: "+resultado+" fuera de rango");

}

} catch(NumberFormatException e) {

System.out.println("Error: "+valor+" no es un numero Valido");

}

} while (repetir);

return resultado;

}

public static String preguntarString(String pregunta) {

System.out.print(pregunta+"\n ");

Scanner entrada = new Scanner(System.in);

return entrada.nextLine();

}

}

1. Detalles de implementación

(lenguaje utilizado y por que usar IntelliJ)

(estructura del proyecto, ficheros y todo)

1. Pruebas

Todo sistema o aplicación debe ser probado mediante su ejecución controlada antes de ser entregado al cliente.

Las pruebas del software son un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, el diseño y la codificación. Se debe reservar el 40% del tiempo del proyecto.

En caso de proyectos con altos requisitos de exactitud (críticos) el tiempo debe ser de tres a cinco veces mayor que todo el resto de fases juntas.

Las pruebas miden dos aspectos fundamentales:

* Verificación: ¿se está construyendo correctamente el producto?
* Validación: ¿se está construyendo el producto correcto?

La verificación, según IEEE, es el proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase. Entonces, verificar el código de un módulo significa comprobar si cumple lo marcado en la especificación de diseño donde se describe.

Por otra parte, la validación es el proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos especificados. Así, validar una aplicación implica comprobar si satisface los requisitos indicados en la ERS.

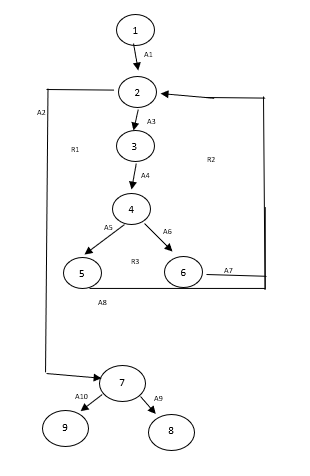
**CAJA BLANCA**

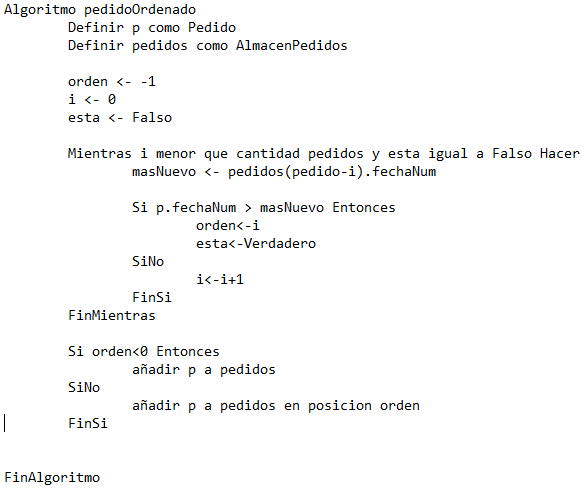
Para demostrar que las pruebas exhaustivas son impracticables se puede estudiar el clásico ejemplo de Myers de un programa de 50 líneas con 25 sentencias IF en serie. El número total de caminos contiene 33,5 millones de secuencias potenciales (contando dos posibles salidas para cada IF tenemos 225 posibles caminos).

El diseño de casos tiene que basarse en la elección de caminos importantes que ofrezcan una seguridad aceptable en descubrir un defecto, y para ello se utilizan los llamados criterios de cobertura lógica.

Aunque el uso de estas técnicas no requiere el uso de ninguna representación gráfica específica del software, es habitual tomar como base los diagramas de flujo de control (flowgraph charts o flowcharts).

La prueba estructural valida que los flujos internos de la aplicación cumplen con la planificación establecida basándose en el código de la aplicación, por ende, hablamos de una prueba de caja blanca, donde el usuario que diseña la prueba tiene acceso completo al código fuente

*COMPLEJIDAD CICLOMATICA DE MCCABE*





*COBERTURA DE SENTENCIAS*

Registro 1

-P=Juan – “Descripcion” -- **21/12/2004** // pedidos-(La cantidad de pedidos es 0)

1-2-7-9

Registro 2

-P=Juan -- “Descripcion” – **21/12/2004** // pedidos(i)-fechaNum(**12/10/2003**) (No hay mas pedidos)

1-2-3-4-5-2-7-9

Registro 3

-P=Juan -- “Descripcion” – **21/12/2004** // pedidos(i)-fechaNum(**12/10/2005**)(No hay mas pedidos)

1-2-3-4-6-2-7-8

*COBERTURA DE DECISIONES*

|  |  |  |
| --- | --- | --- |
|  | **NODO 4** | **NODO 7** |
| **VERDADERO** | REG.3 | REG.3 |
| **FALSO** | REG.2 | REG.1//REG.2 |

**CAJA NEGRA**

Esta técnica divide en un número finito de clases de equivalencia. Las cualidades que definen un buen caso de prueba son que:

* Cada caso debe cubrir el máximo número de entradas.
* La prueba de un valor representativo de una clase permita suponer razonablemente que el resultado obtenido (existan defectos o no) será el mismo que el obtenido probando cualquier otro valor de la clase.

Para identificar las posibles clases de equivalencia de un programa a partir de su especificación se estudian las condiciones de las entradas del programa, es decir, restricciones de formato o contenido de los datos de entrada.

La identificación de las clases se realiza basándose en el principio de igualdad de tratamiento: todos los valores de la clase deben ser tratados de la misma manera por el programa.

Las clases de equivalencia determinadas tendrán en cuenta los datos válidos y los datos no válidos o erróneos.

*TABLA DE EQUIVALENCIA*

|  |  |  |
| --- | --- | --- |
| **Condición de entrada** | **Clases validas** | **Clases no validas** |
| **Cliente** | (1)cliente ==clase cliente | (2)cliente!=clase Cliente |
| **Descripción** | (3)Cualquier opción es valida |  |
| **Fecha** | (4)dia y mes mínimo 1 digito y año 2 digitos separado por “/”  (5)fecha formato dd/mm/yyyy | (6)fecha sin “/”  (7)fecha sin formato  (8) dia y mes <1 digito && año <2 digitos |

*GENERAR CASOS DE PRUEBA*

* CASOS VALIDOS

-Paco // “” // 6/3/01 (1)(3)(4)

-Paco // “” // 06/03/2001 (1)(3)(5)

* CASOS NO VALIDOS

- futbol(clase distinta a cliente) // “” // 6/3/01 (2)(3)(4)

- Paco // “” // 06032001 (1)(3)(6)

- Paco // “” // 21/310/0 (1)(3)(7)

- Paco // “” // 6/3/1 (1)(3)(8)

1. Conclusiones y futuras ampliacions

(explicar que possibles mejoras podràs ponerle en el futuro)

1. Bibliografía

* <https://www.google.com/search?q=pruebas+estructurales+de+software&rlz=1C1GCEA_enES957ES957&ei=bQuVYuBdzZJrnd6MmAU&oq=PRUEBAS+ESTRUCTYURA&gs_lcp=Cgdnd3Mtd2l6EAMYATIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDTIECAAQDToECAAQQzoLCAAQgAQQsQMQgwE6EQguEIAEELEDEIMBEMcBEKMCOhEILhCABBCxAxCDARDHARDRAzoFCC4QgAQ6BQgAEIAEOggIABCABBCxAzoICAAQsQMQgwE6BQgAELEDOgsILhCABBCxAxDUAjoGCAAQHhANOggIABAeEA0QBToICAAQHhAIEA1KBAhBGABKBAhGGABQAFibGWC2JmgBcAF4AYABuAGIAeYRkgEENy4xNZgBAKABAcABAQ&sclient=gws-wiz&safe=active&ssui=on>
* <https://creativecommons.org/choose/>
* <https://www.google.com/search?q=diagrama+de+ESTADOS&rlz=1C1GCEA_enES957ES957&ei=RQqVYti2LM-YaYSXp5AB&ved=0ahUKEwiY7fnf6of4AhVPTBoKHYTLCRIQ4dUDCA4&uact=5&oq=diagrama+de+ESTADOS&gs_lcp=Cgdnd3Mtd2l6EAMyCwgAEIAEELEDEIMBMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEOgcIABBHELADOgcIABCwAxBDOgQIABAKOgQIABANOgcIIRAKEKABOgQIABBDOggIABCxAxCDAUoECEEYAEoECEYYAFDzCVjwJmD7KGgGcAF4AIABjQGIAbYKkgEDOC41mAEAoAEByAEKwAEB&sclient=gws-wiz&safe=active&ssui=on>
* <https://www.google.com/search?q=diagrama+de+comunicacion&rlz=1C1GCEA_enES957ES957&oq=DIAGRAMA+DE+COM&aqs=chrome.2.69i57j0i512l9.6613j0j9&sourceid=chrome&ie=UTF-8&safe=active&ssui=on>
* <https://www.google.com/search?q=DIAGRAMA+DE+ACTIVIDAD&rlz=1C1GCEA_enES957ES957&ei=6wKVYrbTK57g7_UP3NeuoAY&ved=0ahUKEwj29MTe44f4AhUe8LsIHdyrC2QQ4dUDCA4&uact=5&oq=DIAGRAMA+DE+ACTIVIDAD&gs_lcp=Cgdnd3Mtd2l6EAMyCwgAEIAEELEDEIMBMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEOgQIABBDOhEILhCABBCxAxCDARDHARCjAjoICAAQgAQQsQM6CAgAELEDEIMBOgoIABCxAxCDARBDOgcIABCxAxBDSgQIQRgASgQIRhgAUABYuRJg2BdoAHABeACAAbMBiAGKEJIBBDE2LjWYAQCgAQHAAQE&sclient=gws-wiz&safe=active&ssui=on>
* <https://www.google.com/search?q=diagramas+secuenciales+generales+del+sistema&rlz=1C1GCEA_enES957ES957&ei=BgGVYt7PFcf0aLyju-gP&ved=0ahUKEwje6oz34Yf4AhVHOhoKHbzRDv0Q4dUDCA4&uact=5&oq=diagramas+secuenciales+generales+del+sistema&gs_lcp=Cgdnd3Mtd2l6EAMyBQghEKABMgUIIRCgATIKCCEQHhAPEBYQHToECAAQQzoRCC4QgAQQsQMQgwEQxwEQowI6CwgAEIAEELEDEIMBOggIABCxAxCDAToICAAQgAQQsQM6CggAELEDEIMBEEM6BwgAELEDEEM6BQgAEIAEOgcIABCABBAKOgYIABAeEBY6CAgAEB4QFhAKOggIIRAeEBYQHToHCCEQChCgAUoECEEYAEoECEYYAFAAWPJdYLlfaAVwAXgAgAGqAYgBtSCSAQQzLjI5mAEAoAEBwAEB&sclient=gws-wiz&safe=active&ssui=on>
* <https://www.google.com/search?q=tablas+de+especificaciones+casos+de+uso&rlz=1C1GCEA_enES957ES957&ei=AwGVYoHzD4S-aPPNmfAJ&ved=0ahUKEwjBgND14Yf4AhUEHxoKHfNmBp4Q4dUDCA4&uact=5&oq=tablas+de+especificaciones+casos+de+uso&gs_lcp=Cgdnd3Mtd2l6EAMyBQghEKABOgcIABBHELADSgQIQRgASgQIRhgAUOUEWOUEYIgGaAFwAXgAgAF4iAF4kgEDMC4xmAEAoAEByAEIwAEB&sclient=gws-wiz&safe=active&ssui=on>
* <https://www.google.com/search?q=tablas+de+especificaciones+casos+de+uso%C3%A7&rlz=1C1GCEA_enES957ES957&oq=tablas+de+especificaciones+casos+de+uso%C3%A7&aqs=chrome..69i57j33i160l2.9977j0j7&sourceid=chrome&ie=UTF-8&safe=active&ssui=on>
* https://www.google.com/search?q=explicacion+diagrama+de+casos+de+uso&rlz=1C1GCEA\_enES957ES957&ei=9\_yUYuKVMIqSa7SRlqgN&ved=0ahUKEwii7e-H3of4AhUKyRoKHbSIBdUQ4dUDCA4&uact=5&oq=explicacion+diagrama+de+casos+de+uso&gs\_lcp=Cgdnd3Mtd2l6EAMyBggAEB4QFjIGCAAQHhAWOgcIABBHELADOgUIABCABDoICAAQHhAPEBY6BQghEKABOggIIRAeEBYQHUoECEEYAEoECEYYAFCjBVjHGWDmG2gCcAF4AIABmQGIAc8MkgEEMC4xM5gBAKABAcgBCMABAQ&sclient=gws-wiz&safe=active&ssui=on

## 