**CICLO FORMATIVO DESARROLLO DE APLICACIONES WEB**

**PROYECTO DE 1ºDAW**

**GESTION DE UNA PESCADERIA**

****

**AUTOR/A:**

**Raúl Carretero Randez**

**TUTOR/A:**

**Emiliano Torres Martínez**

**FECHA: Valencia, Junio de 2022**

**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.*

**D:\Grado Superior\DAW\PROYECTO\DOCUMENTACION\Licencias\Licencia 1.png**Esta obra está bajo una [Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional](http://creativecommons.org/licenses/by-sa/4.0/).

1. Introducción

En el momento nos dijeron que teníamos que pensar en hacer un proyecto para este primer curso, no tuve ninguna duda en hacerla sobre una pescaderia.

No es común verlo ya que hoy en dia los comercios locales ya se están perdiendo de vista por la desactualización del comercio y la industrialización de las marcas y productos, pero mis padres trabajan en una pescadería en Ruzafa, os dejo aquí información sobre la pescadería en internet para hacerles un poco de publicidad (<http://mercatderussafa.com/puestos/pescados-cristina/>).

Fue la primera idea que tuve en cuanto a realizar un proyecto sobre una gestión de un comercio, ya que me he criado en el mercado y en una parada de venta de pescado. Entonces lo que pensé fue en como hacer que un comercio local, que no tiene los recursos de una gran empresa, pueda amoldarse a los tiempos de hoy sin que repercuta a la empresa económicamente y no tenga que modificarles mucho su modo de trabajo.

Tras una charla con mis padres , me decidí a ello. Les iba a preparar una aplicación para su pescaderia.

* 1. *Justificación*

El porque no es que sea muy claro, pero sobretodo mi justificación es para facilitar el trabajo que tienen, porque tienen que atender a los clientes, a los pedidos realizados con antelación, gestionar cuentas, registrar las ventas, administrar los encargos y los envios a los clientes… Como se puede ver no es un trabajo sencillo.

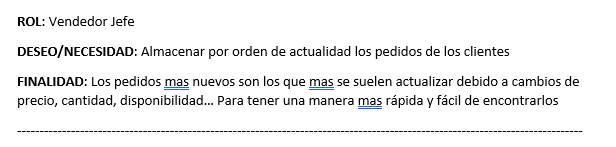
Después de ver todo eso con mis propios ojos no puede quedarme quieto. Porque si puedo ayudarles, no solo a ellos sino a todos los vendedores de comercios locales, lo intentaré

* 1. *Objetivos*

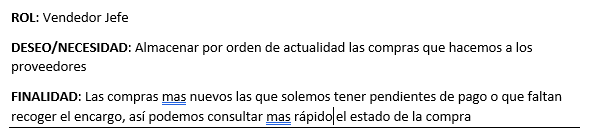
Para saber que debia implementar en mi proyecto, primero debia consultar con la empresa con la voy a colaborar para hacerles el programa, en este caso con mis padres.

Asique les mande por correo unos formularios muy básicos para que explicaran sus necesidades y que les gustaría que pudiera hacer mi proyecto. Obviamente implementaría utilidades para ver un proyecto lo mas completo posible, pero sobretodo que sea funcional y haga lo necesario para facilitar el trabajo de vendedor de pescado.

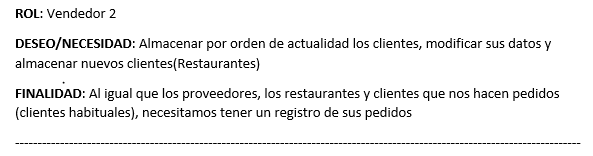
A continuacion las historias de usuario realizadas:



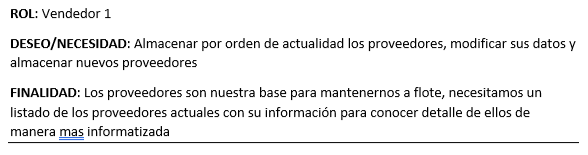
Historia de usuario 1



Historia de usuario 2



Historia de usuario 3



Historia de usuario 4

El objetivo general es que la aplicación haga reducir el tiempo de trabajo en cuanto a, gestión de las ventas diarias en la parada y en la gestión de pedidos con la restauración, base de datos con toda la información de los clientes o restaurantes y de los proveedores que tenemos a dia de hoy.

Historia de usuario 4

1. Especificación de requisitos software

La especificación de requisitos de software (ERS) es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso que describe todas las interacciones que tendrán los usuarios con el software. Los casos de uso también son conocidos como requisitos funcionales.

* 1. *Propósito*

Los requisitos lo que conseguirán es cumplir unos objetivos, algunos serán imprescindibles en el transcurso del proyecto y otros serán algo dispensables, pero todos cumplen su función que es que el proyecto y el programa sea lo más completo posible.

* 1. *Descripción general*

Hay dos tipos de requisitos que son los que introduciremos en este proyecto.

Funcionales y No funcionales, una breve explicación:

* + FUNCIONALES: Definen el comportamiento del sistema, los servicios que se espera que el sistema proveerá, sus entradas y salidas, excepciones, etc
  + NO FUNCIONALES: Tienen que ver con restricciones y exigencias de calidad. Suelen ser requisitos más críticos que los requisitos funcionales. Suelen ser difíciles de cuantificar y verificar
  1. *Requisitos funcionales*
* Registro y almacenamiento de datos de los diferentes restaurantes que sirvan al igual que el de los proveedores
* Modificación del precio en las ventas al cliente en la parada
* El sistema debe de notificar de las ausencias de pagos
* Reflejarse el sumatorio de los detalles de los pedidos y de las compras
* Consultar la cantidad de pedidos, compras, restaurantes, clientes y proveedores que disponemos
* Modificación de información detallada de los clientes, restaurantes y proveedores
* Modificación de pedidos y compras
* Actualización de los estados de los pedidos y compras
* Consulta de una ayuda (Javadoc) del programa
* Dar de baja tanto a clientes como a proveedores, al igual que pedidos y compras
  1. *Requisitos no funcionales*
* Privacidad de los datos almacenados para que sea más seguro la información, uso único por parte de los empleados
* La consultoría de todas las gestiones
* Tiene que ser un diseño claro y en los cuales los botones de selección tiene que ser numéricos o de corta escritura
* Modificación de pedidos almacenados y de clientes en la base de datos
* Elección del tipo de cliente en cada pedido
* Elección del proveedor en cada compra
* Debe de tener la opción guardar información para el caso de que sea un pedido muy elevado, guardarlo para que no se pierda ningún dato
  1. *Tabla de funcionalidad*

Resumir en una tabla las funciones del sistema organizadas en categorías:

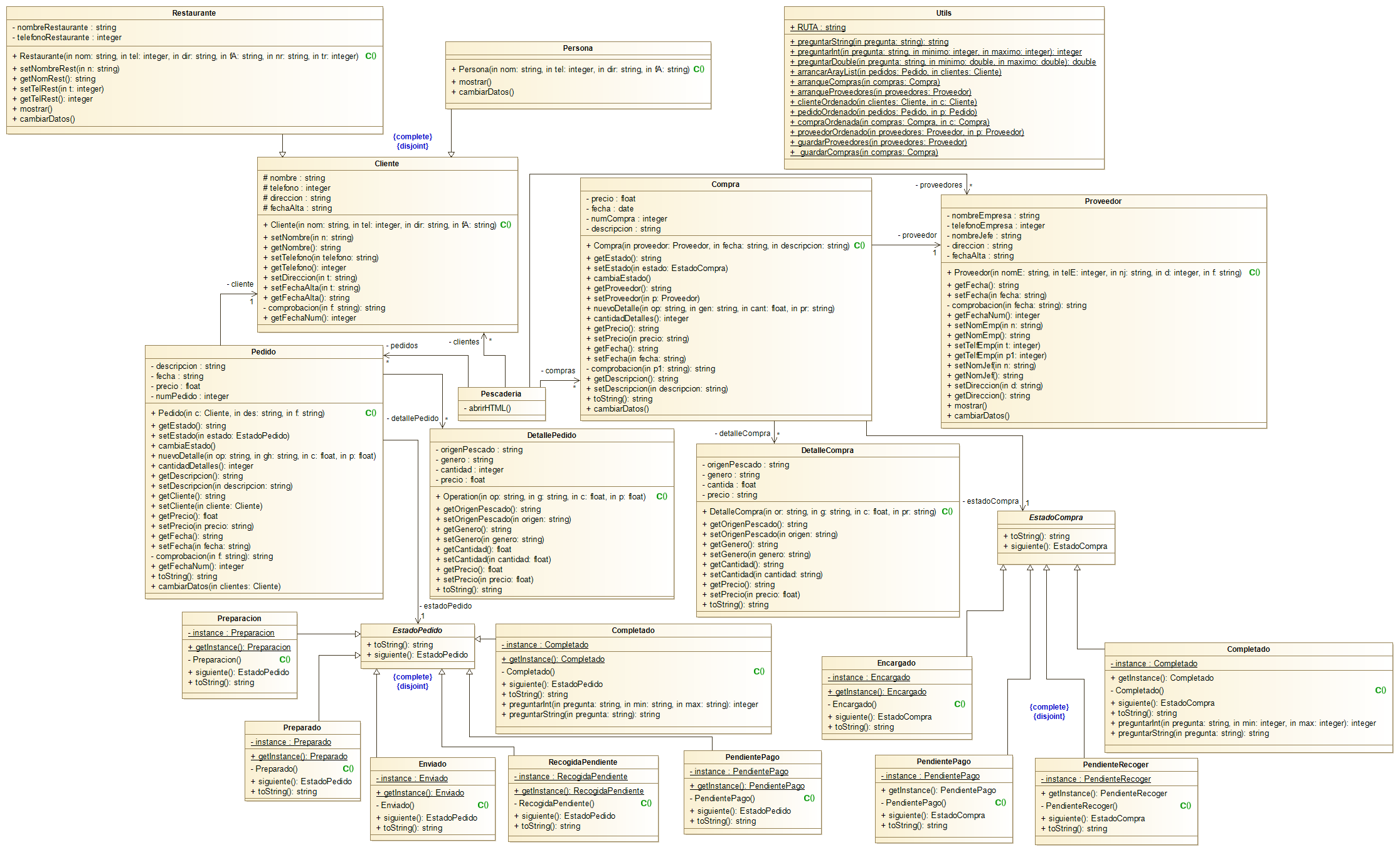
* **Evidentes**: deben hacerse y el usuario es consciente de que se hacen.
  + Almacenarse la información de los restaurantes
  + Reflejarse el sumatorio de los pedidos
  + Sacar por pantalla los resultados de la venta al momento de realizarla
* **Ocultas**: deben hacerse, pero no son visibles para el usuario.
  + Cuando se realiza una compra se almacena el sumatorio de todos sus detalles
  + Cuando compramos el género a los proveedores, se verá reflejado en el resultado de la compra
  + Cuando un restaurante tiene un impago , se vera reflejado en el estado del pedido
* **Opcionales** (adornos): pueden no realizarse y se pueden incorporar sin alterar al resto de las funciones.
  + Modificarse los pedidos de los restaurantes
  + Cuando un restaurante deja de comprarnos, se puede dejar en la lista o eliminarlo

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.

*Casos de uso*

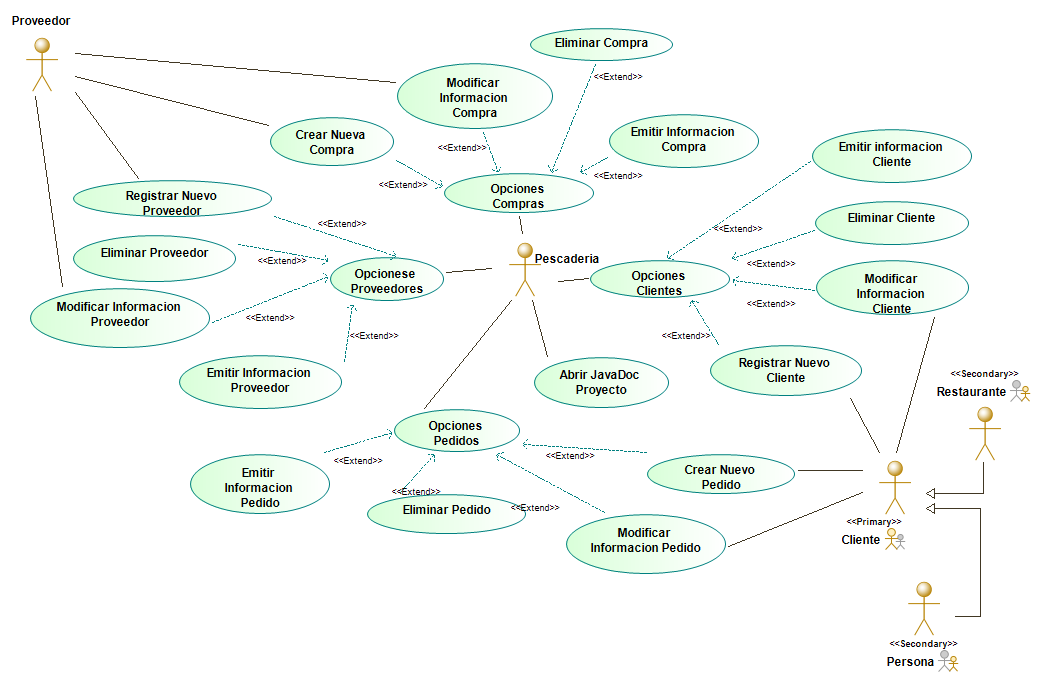
En este apartado, en primer lugar, veremos los diagramas de casos de uso para luego

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.

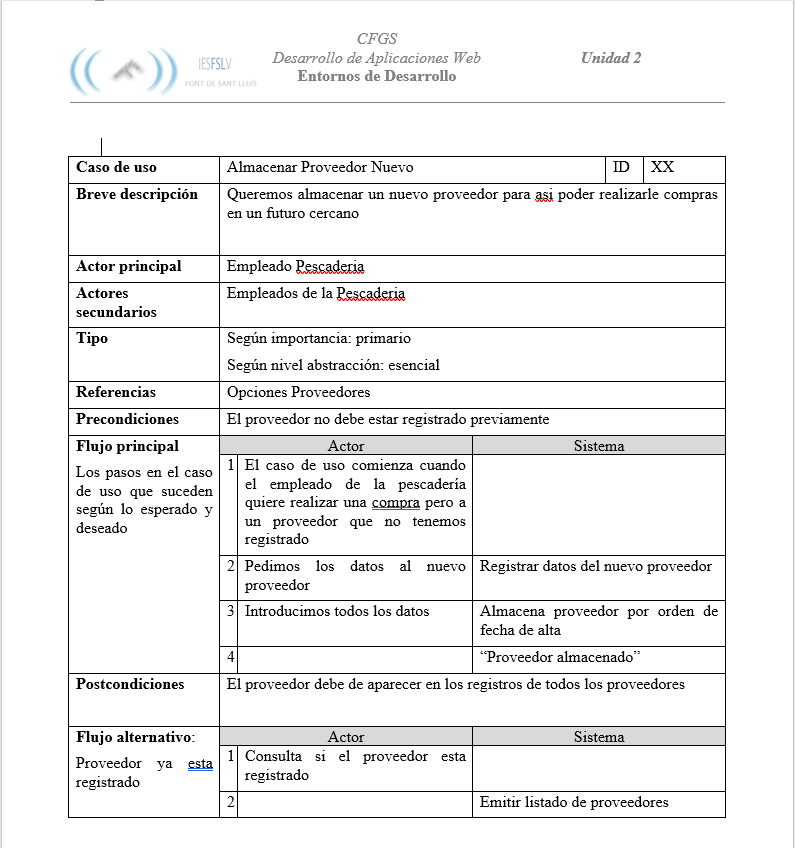
Diagrama de casos de uso de la Pescaderia:

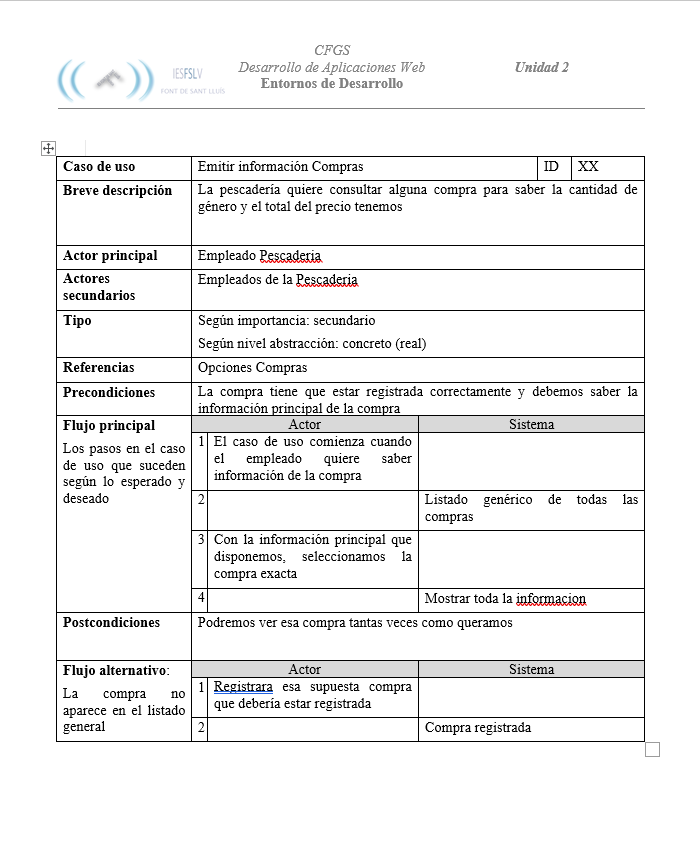
Todo diagrama de casos de uso lleva consigo unes tablas de especificacion , que se encargan de explicar mas detalladamente

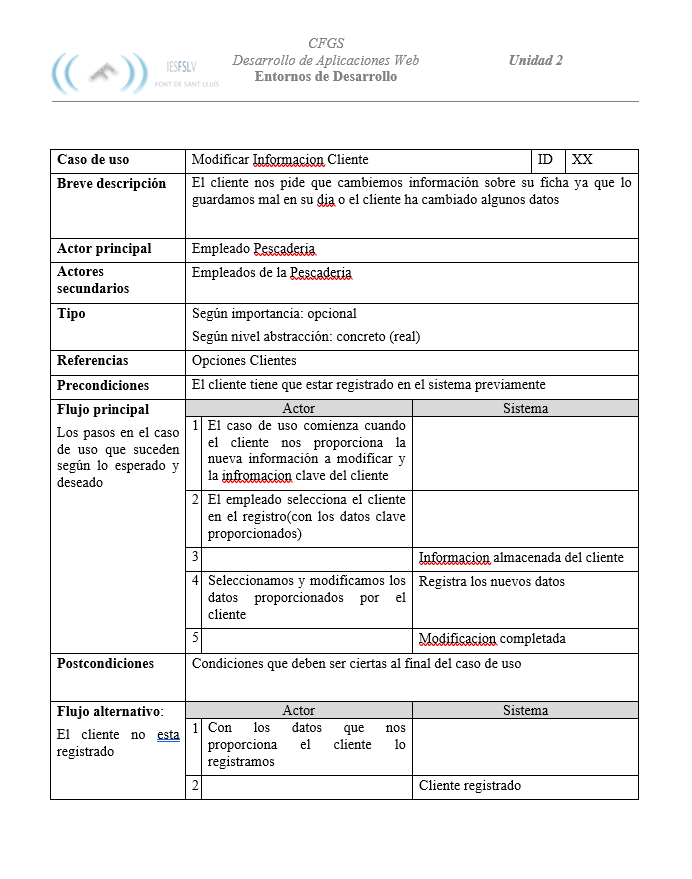
¿Qué son las especificaciones de casos de uso?

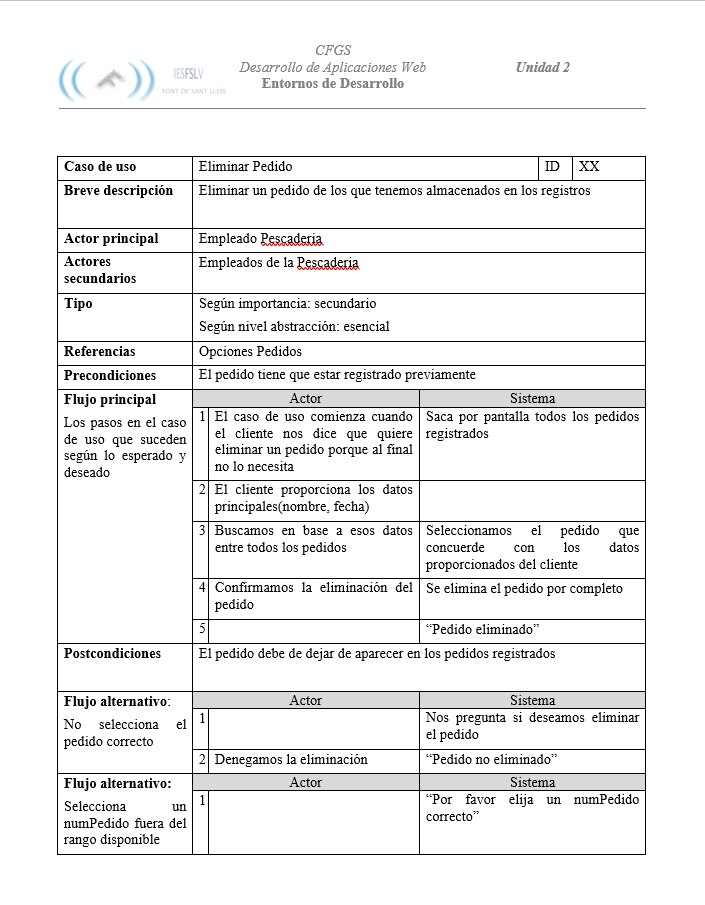
Una especificación de caso de uso proporciona detalles textuales de un caso de uso. Se proporciona una descripción de ejemplo de una especificación de caso de uso. Puede reutilizar y modificar la descripción según se requiera en una especificación de caso de uso. Indica el nombre del caso de uso.

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









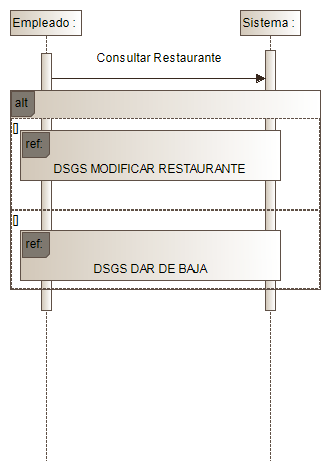
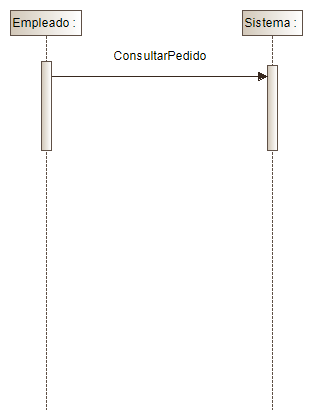
* 1. *Diagrama de Secuencias General del Sistema*

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.

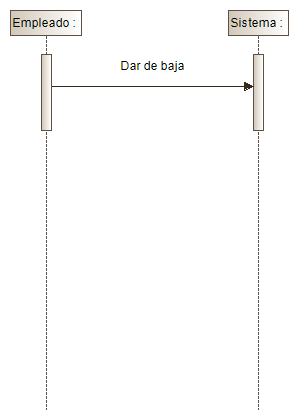
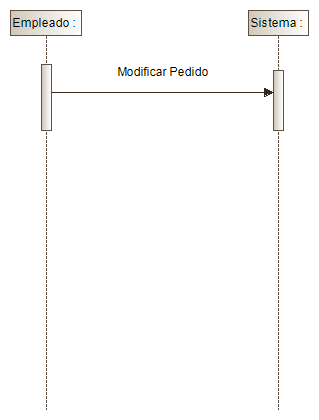
**

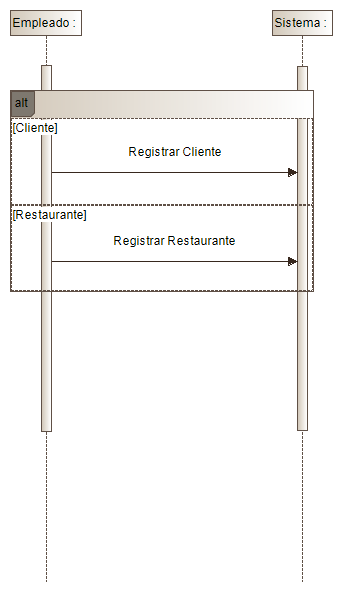
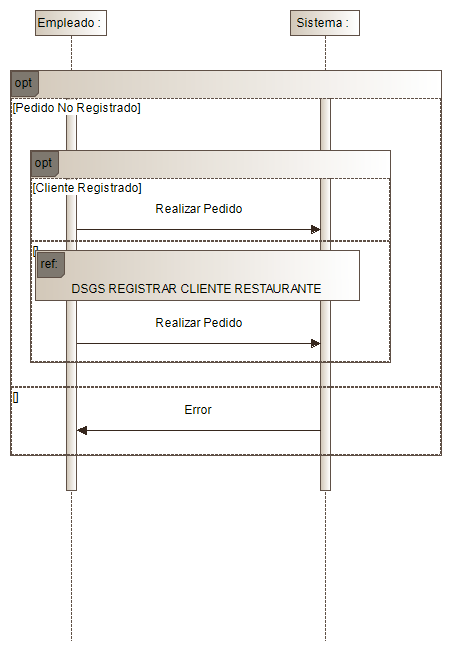
DSGS CONSULTAR PEDIDO

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.

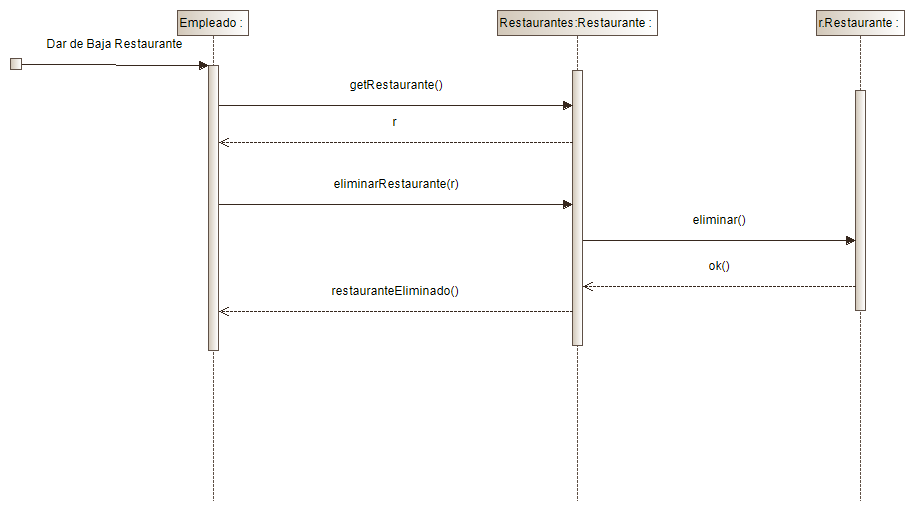
Los diagramas de secuencia se utilizan cuando queremos expresar qué objetos se relacionan con qué objetos, enfatizando el orden en que lo hacen y qué tipo de mensajes se envían entre sí.

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.

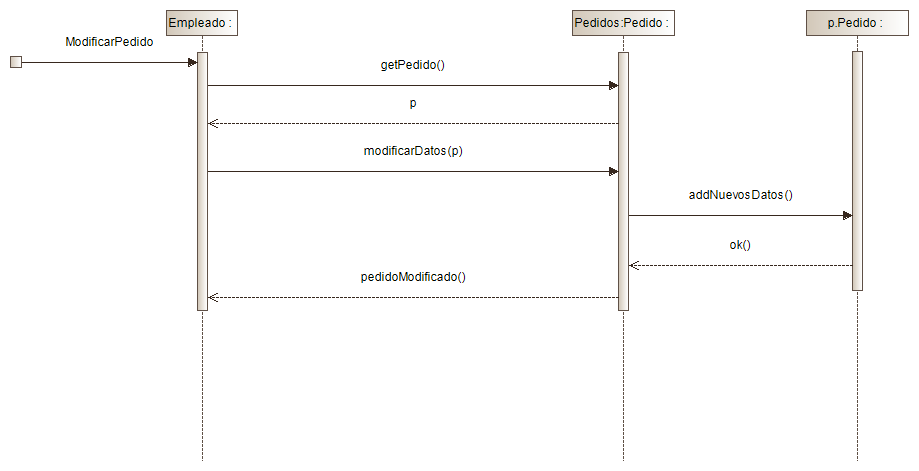
Conslusion, son los diagrames que detallan auhn mas los casos de uso, ya que son diagrames que detallan las acciones de los DSGS.

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

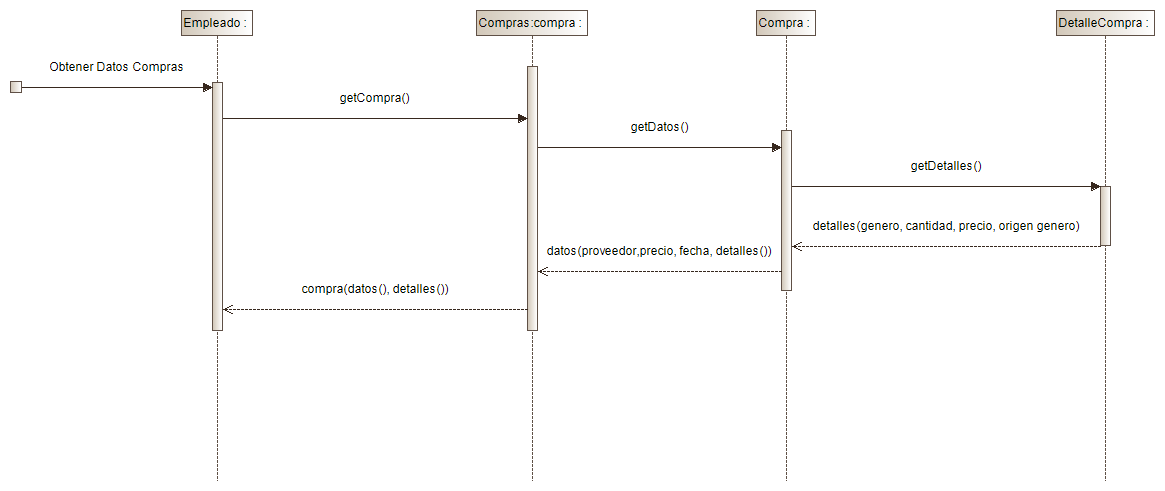
DS DAR DE BAJA RESTAURANTE

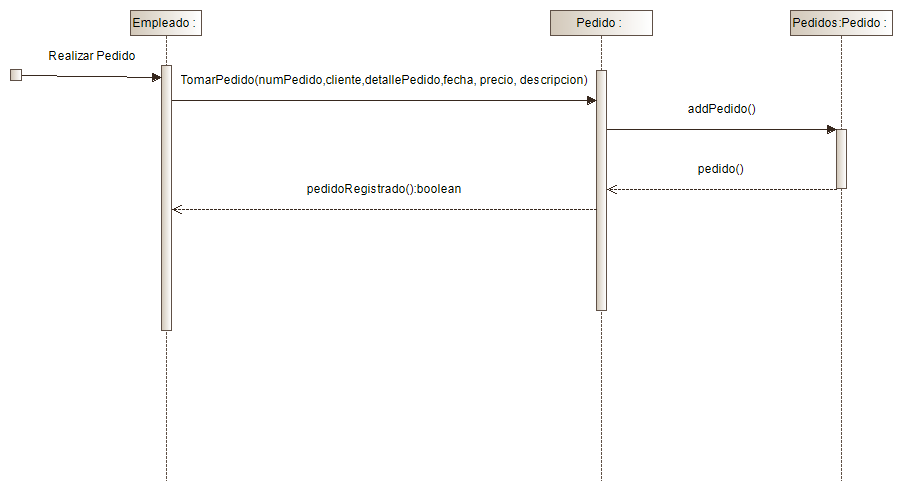


DS MODIFICAR PEDIDO



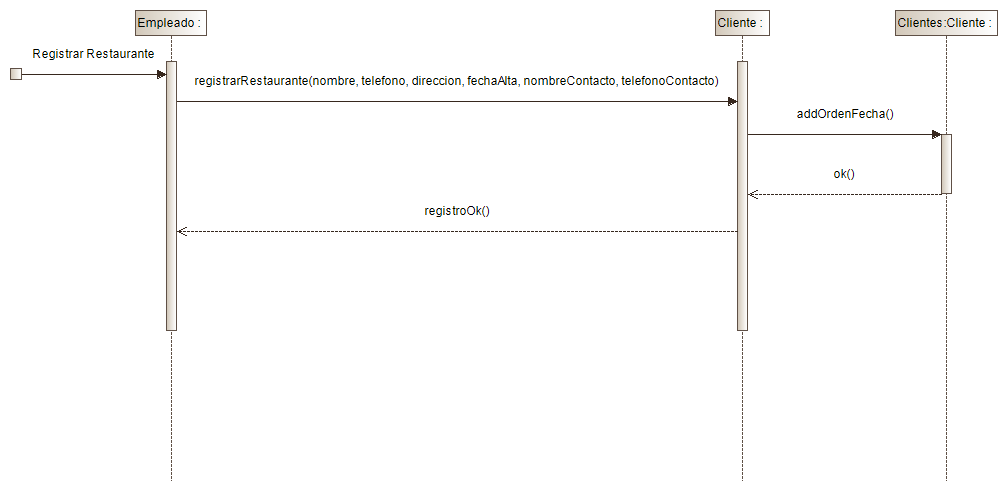
DS OBTENER DATOS COMPRAS



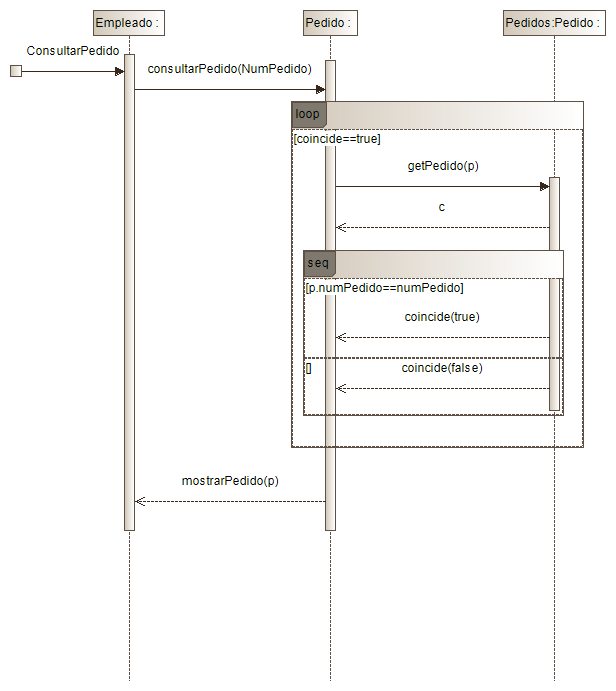


DS REALIZAR PEDIDO

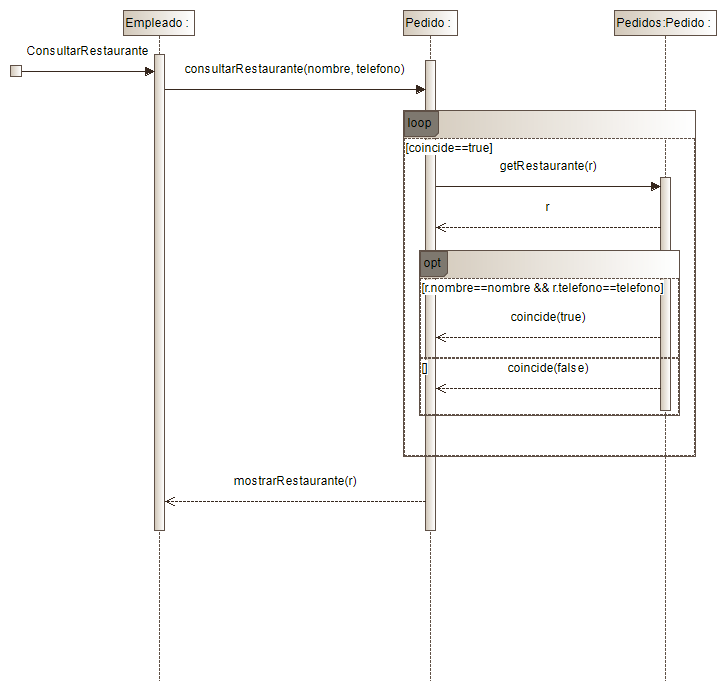
DS REGISTRAR RESTAURANTE



DS CONSULTAR PEDIDO



DS CONSULTAR RESTAURANTE



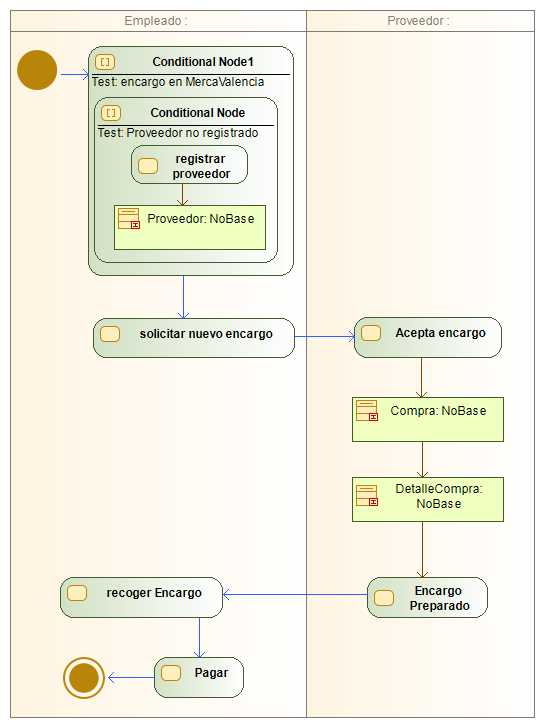
* 1. *Diagrama de Actividad*

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.

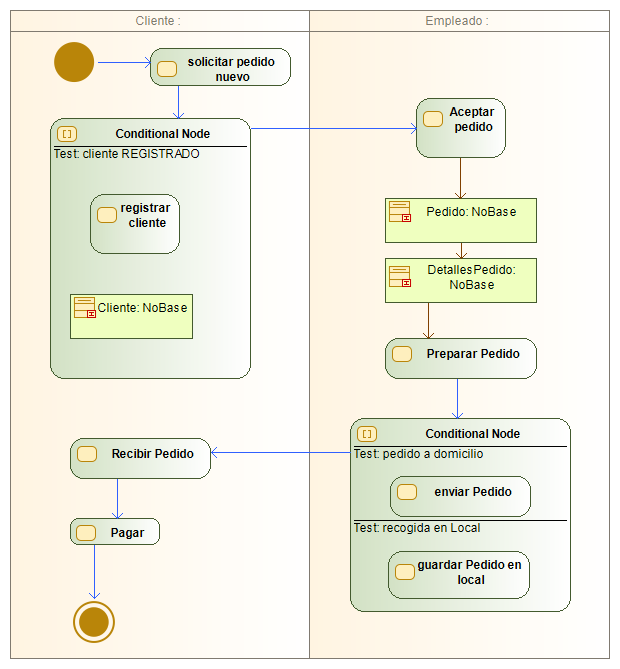
Conslusion, es un diagrama que definí la actividad que seguirà el programa para realitzar una accion concreta.

A continuacion los DA del proyecto de la pescaderia:

DA COMPRAS



DA PEDIDOS

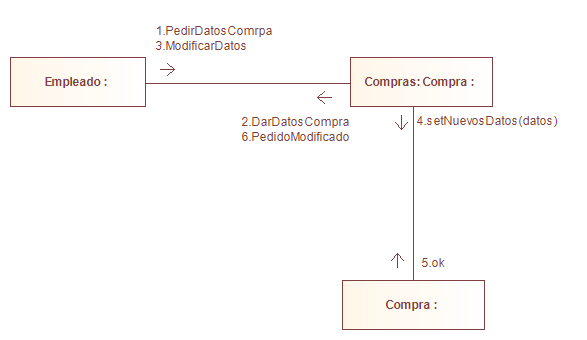


* 1. *Diagrama de comunicacion*

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.

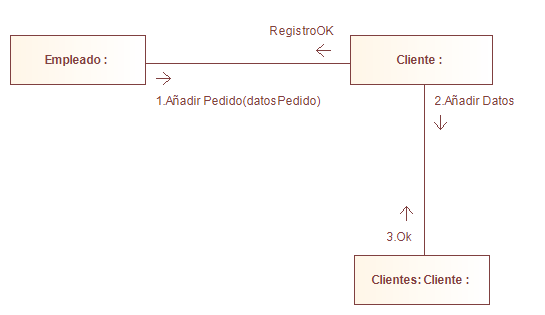
Conclusio, es un diagrama en el que se ven las llamadas que se hacen entre clases o componentes del proyecto en un diagrama de secuencia

A continuacion los DCom del proyecto Pescaderia:



DCom MODIFICAR COMRPA

DCom REGISTRAR PEDIDO



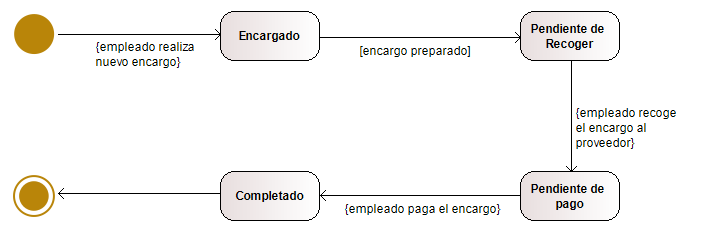
* 1. *Diagrama de estados*

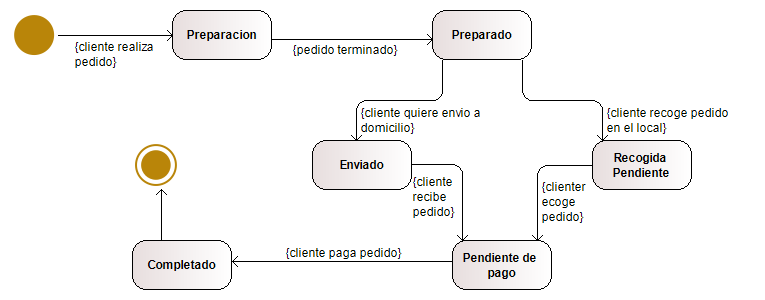
Un diagrama de estados, en ocasiones conocido como diagrama de máquina de estados, es un tipo de diagrama de comportamiento en el Lenguaje Unificado de Modelado (UML) que muestra transiciones entre diversos objetos.

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





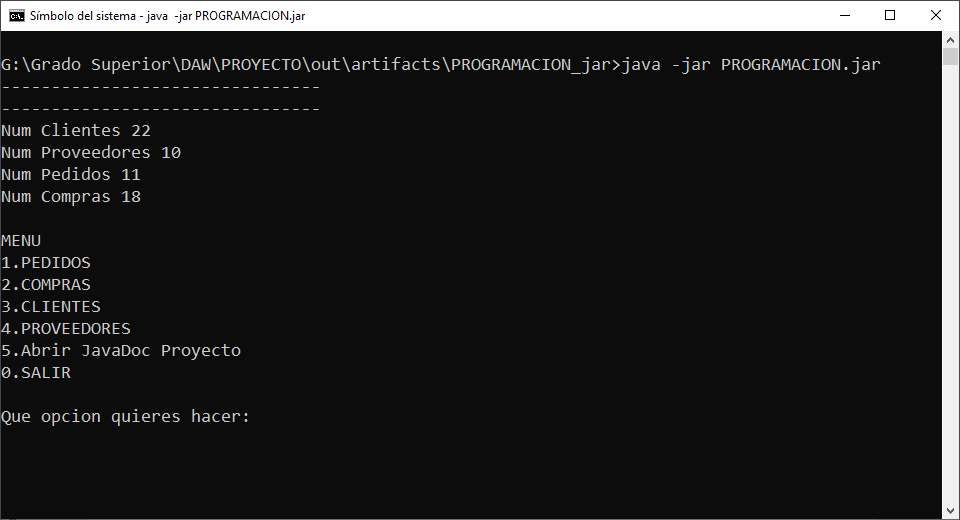
DTE PEDIDOS

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

Lo que vamos a detallar en este apartado es la visualización de la aplicación, como se ve, el porque y todos los detalles que el usuario verá en nuestra interfaz.

El proyecto se visualizará en una interfaz del cmd, la consola del sistema. Al no tener una interfaz gráfica, es la única opción que disponemos para que cualquier usuario pueda utilizar para la utilización de nuestro proyecto(actualmente).

Otras opciones que podríamos utilizar, es el uso de cualquier IDE que acepte el lenguaje de Java, pero no es para todos los públicos ya que deberíamos tener instalado el jdk y modificar configuraciones del sistema. Como se puede ver, la interfaz mas sencilla y apta para cualquier persona que no tiene conocimientos de Java o uso de IDEs , es el cmd del sistema

A continuación, una imagen de la interfaz del proyecto del CMD:

Lo que veis en esa imagen es el arranque inicial del programa, con un comando que toda esta información se detalla en el manual de usuario. En el que explicaremos como utilizar el programa y sacarle todo el provecho

Los que veríamos si usáramos un IDE en vez de la consola del sistema, es exactamente lo mismo porque nuestro programa no usa interfaz gráfica, es todo modo texto. La única diferencia que te proporciona un IDE, es la posibilidad de actualización y modificación del propio código de la aplicación, pero en este caso que el uso es para alguien sin conocimientos de programación , la mejor opción de todas es la consola del sistema.

* 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
   1. **Que IDE utilizar**

Un entorno de desarrollo integrado (IDE) es un sistema de software para el diseño de aplicaciones que combina herramientas comunes para desarrolladores en una sola interfaz de usuario gráfica (GUI).

El IDE que he utilizado para realizar el proyecto es **IntelliJ Community**. De todos los IDEs disponibles que he tenido, uno de los que mas ganes tenia de probar era este.

A continuacion una tabla detallando los motivos por los cuales utilitzar IntelliJ como IDE para proyectos con lenguaje Java(entre otros lenguajes)

**Comparativa de Entornos Desarrollo**

|  |  |
| --- | --- |
| **Nombre del testeador** | Raúl Carretero |

**Datos básicos del producto**

|  |  |
| --- | --- |
| **Datos de contacto** | JET BRAINS - IntelliJ IDEA |
| **Requisitos** | * Windows 10, 8 * 2GB libres de RAM y 8GB de RAM total recomendado * 2,5 GB espacio disco, SSD recomendado * 1024x768 mínimo resolución pantalla |

**Pruebas realizadas**

|  |
| --- |
| * Creación de proyectos nuevos. * Abrir proyectos creados en otros IDEs como Eclipse, NetBeans, BlueJ. * Abrir archivos .java ya creados para probar el compilador del IDE. * Crear paquetes con los ejercicios prácticos de PRG para comprobar su funcionalidad y ver como se reflejan los errores en el compilador. * Crear archivos de todo tipo para ver por completo las opciones que te da el IDE. |

**Tabla de evaluación**

|  |  |  |
| --- | --- | --- |
| **Aspectos ev.** | **Descripción** | **Puntuación** |
| **Linux / Windows** | * **Windows**: *.exe* fácil de ejecutar y una instalación sencilla (genial para gente sin conocimientos básicos) * **Linux**: archivo comprimido en el cual tienes que descomprimir, y ejecutar el archivo *.sh* (muy complejo para gente inicial, necesidad de documentarse previamente) | 10 / 7 |
| **Versiones** | Community // Ultimate // Edu |  |
| **Lenguajes que soporta** | * **Community**:   Java, Kotlin, Groovy, Scala, Python, Jython, Rust, Dart, HTML, XML, JSON, YALM, XSL, XPath, Markdown, CSS, Sass, SCSS, Less, Stylus   * **Ultimate**:   Lenguajes de Community\*, Cython, Ruby, JRuby, PHP, Go, SQL, JavaScript, TypeScript, CoffeeScript, ActionScript. | 7 / 10 |
| **Facilidad de uso** | Muy sencillo una vez entiendes las utilidades de los paquetes y sabes como entender los errores. Ejecución de los archivos .java muy sencillo y fácil de entender. | 9 |
| **Características** | * **Community**:   + Amplias herramientas de desarrollo integrados de JVM   + Gran variedad de control de versiones (Mercurial, GitHub …)   + Desarrollo Colaborativo   + Integración con Space   + Temas personalizados * **Ultimate**:   + Características de Community\*   + Herramientas para bases de datos y cliente HTTP   + Sincronización de ajustes a través de cuenta de JetBrains   + Integración con Sistemas de seguimiento de incidencias | 8.5 / 9.5 |
| **Debugger** | * **Community**:   + Maven   + Gradle   + Ant   + Virtualenv/Buildout * **Ultimate**:   + Debugger Community\*   + sbt, Bloop, Fury   + npm   + Webpack   + Gulp, Grunt   + Phing | 6 / 7.5 |
| **Velocidad** | Dependiendo de la densidad del proyecto o del archivo. Pero en general es un IDE muy fluido y rápido de gestión | 8.5 |
| **Documentación[[1]](#footnote-1)** | Página oficial de JetBrains  Página de Wikipedia  Foros y paginas web con experiencia del IDE | 9.5 |
| **Calidad/Precio** | * **Community**: GRATUITO * **Ultimate ORGANIZACIONES**:   + ANUAL: 499€ sin IVA     - 1º año(normal) // 2º año (20% descuento) // 3º año en adelante (40% descuento)   + MENSUAL:49.90€ sin IVA * **Ultimate PARTICULAR**:   + ANUAL: 150€ sin IVA     - 1º año(normal) // 2º año (20% descuento) // 3º año en adelante (40% descuento)   + MENSUAL:14.90€ sin IVA * **Edu**: GRATUITO | Gratuito: 10  Pago: 3.5 |
| **Puntuación final** *(Community // Ultimate)* | | **8.38 // 8.25** |

**Ventajas e inconvenientes**

|  |  |
| --- | --- |
| **Ventajas** | **Inconvenientes** |
| * Página oficial muy completa, foros y páginas web con mucha documentación y suficientes ejemplos prácticos para conseguir entender y dominar el EDE en general. * Gran personalización de estética y de organización con las carpetas de los proyectos * Prácticamente de los mejores IDEs que puedes tener gratuitamente * Viene con todo instalado de serie. Además, existe un amplio set de plugin * Integración con sistemas de control de versiones. * Sensación de fiabilidad y robustez muy superior a otros entornos. No hace las «cosas raras» que hacen otros IDEs de vez en cuando * Herramienta de refactorización extremadamente inteligente. | * La versión Ultimate me parece caro teniendo en cuenta que tenemos IDES que aceptan mas cantidad de lenguajes y son gratuitos. * La versión Community tiene un abanico muy pequeño de lenguajes soportados |

**Valoración personal**

|  |
| --- |
| A mi parecer me parece el mejor IDE que he utilizado. Eficiente, bonito, personalizable, sencillo de entender… Es un IDE que puedes utilizar teniendo un nivel de programación bajo. Lo recomendaría usar en los cursos de programación como IDE de base, como por ejemplo en 1º de DAW. |

**Bibliografía**

-https://es.wikipedia.org/wiki/JetBrains

-https://www.jetbrains.com/idea/

-https://www.jetbrains.com/es-es/products/compare/?product=idea&product=idea-ce

-https://ciksiti.com/es/chapters/2984-how-to-install-jetbrains-intellij-in-debian--linux-hint

-https://www.adictosaltrabajo.com/2012/03/26/trabajando-intellijidea/

-http://recursostic.educacion.es/observatorio/web/ca/ software/programacion/911-monografico-java

-https://dialnet.unirioja.es/servlet/articulo?codigo=6775562

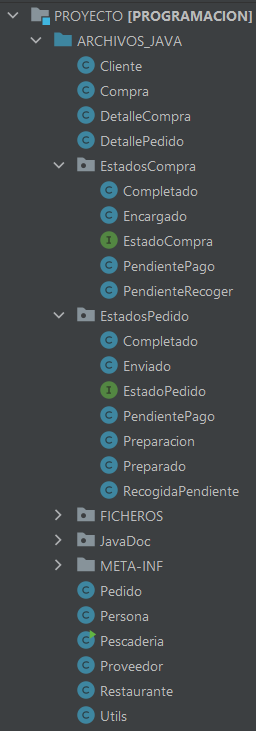
**Anexos**

No hago las comparativas con *IntelliJ Edu*, ya que es muy similar al *IntelliJ Community*, con lo cual tiene una valoración similar ambas versiones del IDE.

* 1. **Estructura de los archivos del proyecto**

Lo que vamos a explicar a continuacion es la estructuracion de los archivos Componentes del proyecto.

Os muestro una imagen con la estructuracion del proyecto:

Puntos a detallar:

Tenemos dentro de la carpeta PROYECTO, el propio proyecto llamado PROGRAMACION, para asi hacer distinción de los modulos a los que se presenta este proyecto.

Como se puede contemplar, las clases genericas como pueden ser COMPRAS, PEDIDOS, PROVEEDORES... estan visibles en la carpeta fuente del proyecto, es decir, no esta dentro de ningun paquete con lo cual todas las clases dentro de esa carpeta son completamente visibles entre si.

Tambien se puede ver que tenemos dos paquetes, llamados EstadosCompra y EstadosPedido.

Lo que almacenamos en esos paquetes son las clases con las interfases de los estados que van a tener nuestros pedidos y nuestras compras.

De tal manera que la carpeta Fuente queda bastante desahogada al tener todas esas clases organizadas por paquetes.

Lo único que deberíamos hacer para que se pueda comunicar los archivos de la carpeta fuente con los archivos de los paquetes, es importar los paquetes en las clases que vayamos a usar los paquetes a la carpeta fuente .

(import package… en la clase compra)

Por disponemos de una carpeta llamada FICHEROS que es donde almacenamos los ficheros cuya información usaremos en la clase main Pescaderia.

**En la carpeta llamada META-INF y JavaDoc, almacenamos información variada**

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.

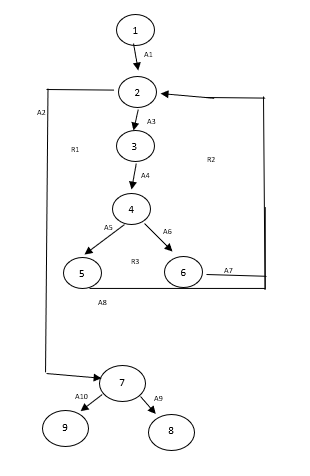
**CAJA BLANCA**

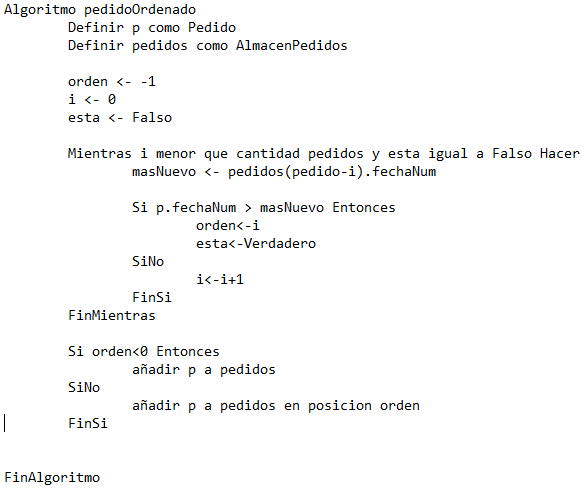
Para demostrar que las pruebas exhaustivas son impracticables se puede estudiar el clásico ejemplo de Myers de un programa de 50 líneas con 25 sentencias IF en serie. El número total de caminos contiene 33,5 millones de secuencias potenciales (contando dos posibles salidas para cada IF tenemos 225 posibles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

Las conclusiones que he sacado de este proyecto son muy claras. Los programadores deben de anticiparse a muchos inconvenientes que se generaran en el proceso de la utilización del programa, con lo cual hay que tener una paciencia y un control del código gigantesco.

Otra de las conclusiones mas grandes es la realización de un proyecto, cuando loo ves desde afuera piensas que solo tienes que ponerte a hacer código y poco mas. Pero estamos muy equivocados, lo que mas me ha enseñado este proyecto es eso, los procesos previos, los diagramas, las pruebas, los esquemas, la consultoría con la empresa, la consultoría con compañeros de oficio mismo… todo eso antes de introducirte a hacer el código del programa.

Me ha dejado bastante claro que cuando mas sabes sobre la programación, más descubres que no sabes. Es un área tan extensa que cada vez que abres los ojos, solo haces que ves más terreno desconocido.

Además, el conocimiento sobre la materia, el don de gentes y la fluidez que hay que tener para la exposición del proyecto, no sabia que era tan importante. Pero agradezco que este proyecto me haya hecho abrir los ojos con todo eso.

Y como no, el aprendizaje sobre nuevos IDEs como es mi caso y la implementación de todo lo aprendido hasta el día de hoy del lenguaje Java y sobre las aplicaciones de Entornos de desarrollo.

* 1. Futuras ampliaciones

El programa está en la versión beta, entonces le faltan muchas implementaciones que a la larga harán que el programa sea mas completo y vas fácil de usar.

La ampliación que considero fundamental para el futuro será el aplicarle una base de datos y gestionar toda la información en base a ella, nada de ficheros y ArrayList. Una aplicación java que juegue con una base de datos.

Otra de las implementaciones que me gustaría aplicarle es una pagina web comercial, ya que hoy en día una pagina de compras hace que una empresa tenga mayor reconocimiento y expansión. Además de una web para que se conozca la propia parada de mis padres, una pescadería actualizada a los tiempos.

Un objetivo a gran escala que tengo con este proyecto es el siguiente:

Me gustaría hacer una aplicación para cualquier parada de pescadería, en la cual meta la información a su disponibilidad y adaptada a cualquier pescadería. O incluso a cualquier parada de mercado(carnicería, verdulería, vinoteca, herboristería) ya que todos tienen clientes, proveedores, pedidos y compras.

La comercialización de un programa genérico para todos los puestos locales de mercado.

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>
* <http://mercatderussafa.com/puestos/pescados-cristina/>
* <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>
* <https://creativecommons.org/licenses/by-sa/4.0/>
* <https://www.google.com/search?q=pescaderia+mercado+ruzafa&rlz=1C1GCEA_enES957ES957&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjHhbWC_4n4AhUKwoUKHW90Cq8Q_AUoAnoECAEQBA&biw=958&bih=927&dpr=1&safe=active&ssui=on#imgrc=nSyDasGmJHT6lM>

## 

1. Documentación on-line e impresa [↑](#footnote-ref-1)