CURSO 2013 - 2014

ACTIVIDAD EVALUABLE Y CALIFICABLE

1. Portada con:

Asignatura: 71013035 – Diseño de Software

Año de la práctica:

Centro Asociado al que pertenece:

Tutor que le da asistencia:

Datos personales: Nombre y apellidos:

DNI o número de expediente:

Contacto (teléfono o correo electrónico):

Localidad de residencia:

Datos de coste: Horas de dedicación al estudio de los contenidos:

Nº de actividades no evaluables realizadas y horas de dedicación:

Horas de dedicación para realizar esta actividad:

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

swdesign@issi.uned.es

2. El enunciado y planteamiento del caso de estudio.

El dominio del problema es un **sistema de información para las comandas y gestión de almacén en la cocina de un restaurante** (KitchCommander). Las comandas se originan en la sala, a través de los terminales inalámbricos que tienen los camareros. Tras atender a un cliente, el camarero envía la selección a un receptor central en cocina. Cuando una comanda llega al receptor central, el sistema deduce de los artículos en almacén las cantidades aproximadas que se han consumido y se imprime una orden de cocina (OrdenDeCocina) con los siguientes datos:

- Número de mesa (MesaID)
- Camarero (TerminalID)
- Número de comanda (CommandN) y hora de emisión (TimeStamp)
- Relación de platos (PlatoID), según orden de salida (entrantes, primeros…), con las variaciones y observaciones solicitadas para cada uno (guarnición, ingredientes añadidos o eliminados, cocción de la carne, etc.) y tiempo estimado de elaboración (o espera) acumulado para cada fase de salida.

El jefe de cocina utiliza la orden impresa para coordinar el trabajo de los cocineros. Según se elaboran los platos, se colocan en el mostrador de salida y el camarero los retira, marcándolos como 'servidos' en su terminal.

Si el sistema detecta la falta de algún ingrediente insustituible, emite una alarma en el terminal central y bloquea la oferta de los platos afectados en los terminales de sala.

Cuando el cliente pide la cuenta, el camarero marca la comanda como 'terminada' para que el sistema envíe las consumiciones definitivas de cocina a la caja central, que confecciona el recibo de cobro.

Se pretende que el sistema tenga un sencillo planificador de tareas de forma que, en función de los tiempos de elaboración aproximados y el personal de cocina en esa sesión, pueda calcular el acoplamiento en las tareas y detectar (cuando hay retrasos excesivos) si hay algún problema en cocina.

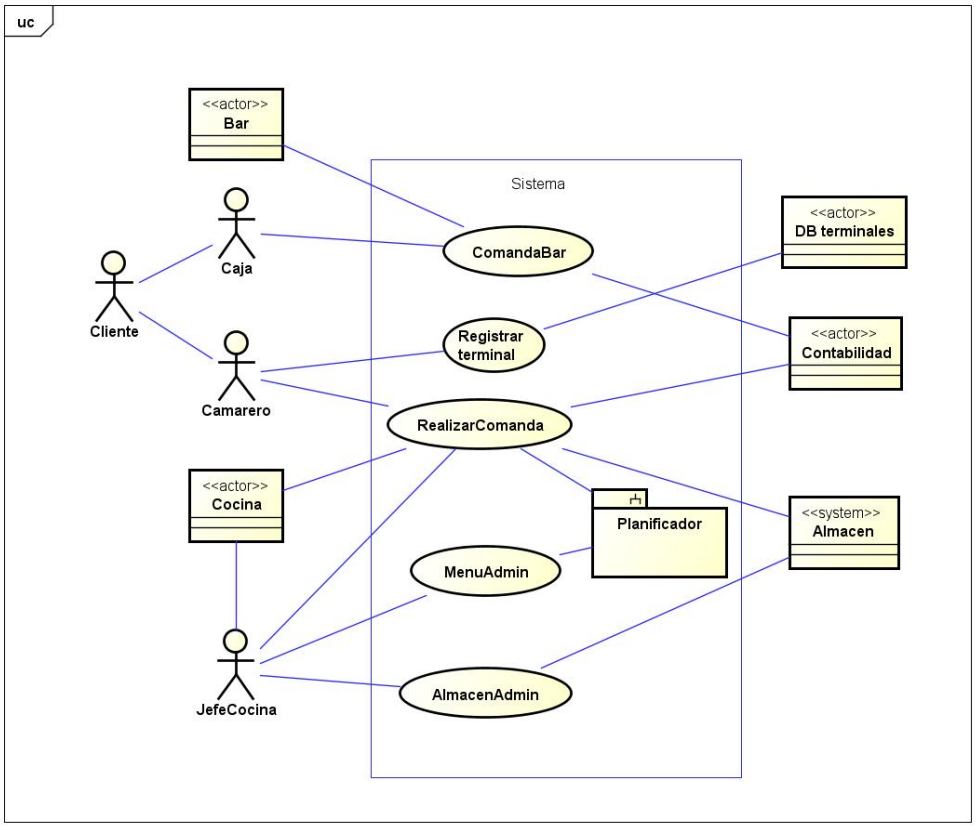
Los detalles y simplificaciones admitidas son:

- Los platos que contiene la carta son fijos. Los datos de sus ingredientes, necesidades y tiempos de elaboración están incluidos en el sistema. Sin embargo, la mayoría de los elementos de la carta admiten variaciones que están reconocidas por el sistema y contempladas en la confección de la comanda.
- Cada camarero tiene su terminal, intransferible en su turno. Cada mesa es atendida por un único camarero, desde el principio al final de la comida.
- Las comandas y el sistema recogen consumos de cocina, únicamente. Las bebidas y consumiciones de cafetería y bar, se recogen aparte y se tramitan en la caja central.
- Si un plato se devuelve a cocina, el sistema debe permitir que el jefe de cocina actualice la comanda en el terminal central (sin que repercuta en el precio final).
- El sistema debe permitir cancelar una comanda o modificarla. Se deja a su juicio en qué casos se hace, cómo se realiza y las repercusiones que tenga para el funcionamiento del sistema.
- En el caso de que el sistema presente una alarma y bloquee los platos correspondientes a un ingrediente agotado y, sin embargo, aún quede suficiente cantidad del ingrediente, el jefe de cocina debe poder modificar esa información del almacén y revertir la situación de alarma y bloqueo.
- El perfil de Administrador es el único que puede hacer las tareas de mantenimiento y, normalmente, dicho perfil lo ocupa el jefe de cocina. El sistema tiene un modo de mantenimiento que solamente se puede ejecutar en los períodos de reposo, entre sesiones de cocina. En el mantenimiento se actualiza:
    - La información del almacén.
    - Topología del comedor (número de mesas, etc.)
    - El tiempo, utensilios y otros recursos compartidos, necesarios para elaborar cada plato.
    - La disponibilidad de utensilios, recursos y equipamiento de cocina (por ejemplo, la avería de un horno)
    - La oferta de la carta. Se puede quitar, modificar o añadir un plato, pero es necesario incluir toda la información asociada (ingredientes, tiempos, recursos de elaboración…), además de las variantes de comanda admitidas.

3. **El enunciado de cada cuestión y las respuestas.** Para cada cuestión, incluirá los desarrollos, listados, diagramas y las argumentaciones que estime necesarios.

Evaluación de ***Casos de Uso***

1. (0'5 puntos) Identifique al menos 4 casos de uso primarios y sus actores correspondientes. Represente los resultados en un diagrama de casos de uso de UML.



2. (1 punto) Escriba el caso de uso <<*RealizarComanda*>> en un estilo y formato completo, esencial y breve. Incluya tanto el escenario principal de éxito (flujo básico correspondiente a que un camarero transmita el pedido de una mesa, hasta que el cliente pida la cuenta) como 2 extensiones o flujos alternativos que pudieran ser frecuentes. No escriba un encabezamiento demasiado elaborado del caso de uso (es decir, omita *propósito*, *resumen*…); en su lugar, afronte directamente el transcurso típico de los acontecimientos.

Caso de uso: **Realizar Comanda**

Estilo informal, extendido y esencial.

Evolución típica de los acontecimientos

Acciones del actor	Respuesta del sistema
1. El caso de uso comienza cuando los comensales de una mesa, tras consultar la carta, solicitan al camarero que les tome nota. El camarero identifica la mesa e introduce el número de comensales que van a comer.	2. El sistema actualiza la situación de la carta en el terminal y presenta un menú de selección.
3. El camarero anota los platos solicitados en el orden de salida que se desee. También anota la cantidad cuando varios comensales coinciden en su elección (normalmente 1) y, en algunos casos, la información relevante (cocción de la carne, etc.). Normalmente se seleccionan platos estándar, sin modificaciones.	4. El sistema permite cancelar y modificar (en el terminal) la selección de cualquier plato ofertado o sus opciones y variantes.
5. Una vez acordados todos los platos, el camarero envía la comanda al terminal central.	6. El sistema recibe la relación de platos, opciones y orden de la comanda y la organiza en un formato adecuado. La información se envía al Planificador que calcula las secuencias de tareas en cocina, los tiempos y las coordina con las que ya están en ejecución. El sistema recibe la secuenciación del Planificador e imprime una Orden de Cocina correspondiente a la comanda.
	7. El subsistema Cocina elabora los platos y, según se terminan, se disponen en el mostrador de salida.
8. El camarero recoge el plato en cocina, lo sirve y lo marca como ‘servido’ en esa comanda.	9. El sistema recibe la actualización de la comanda y la transmite al Planificador. A su vez, éste actualiza la secuencia de tareas, las existencias en el Almacén y, si es necesario, envía orden al sistema (que genera un aviso en los terminales) para bloquear un plato del menú. Se registra la

consumición en el sistema.

10. El cliente pide la cuenta. El camarero marca la comanda como ‘terminada’.
11. Se envía a Caja la relación de consumiciones de la comanda. La Caja recaba la relación de consumiciones de bar, pendientes de esa mesa. Se imprime la lista acumulada de las consumiciones y coste total de la cuenta.
12. La Caja informa del coste total y solicita el pago al cliente.
13. El cliente paga en metálico.
14. La Caja registra el pago en el sistema.
15. Se genera el recibo.
16. La Caja da el recibo al cliente, que se va.

Alternativas

- Paso 3: Algún comensal requiere variaciones sobre el plato estándar. El camarero le informa de cada opción y las anota.
- Paso 3: Si algún comensal selecciona un plato o una variación agotada, el camarero le informa para cambiar la selección.
- Paso 6 bis. El cliente devuelve un plato, cambia su elección o cancela el plato. El camarero modifica la comanda y envía la modificación al terminal central. El sistema responde en el Paso 9 bis.
- Paso 9 bis. El sistema recibe la modificación de la comanda y la transmite al Planificador. A su vez, éste actualiza la secuencia de tareas, las existencias en el Almacén y, si es necesario, envía orden al sistema para cancelar un plato del menú. Se imprime la Orden de Cocina que sustituye a la anterior, con alertas para todas las tareas que se vean afectadas. Se continúa en el Paso 7.
- Paso 9 bis-bis. En el caso de una falsa alarma en el bloqueo de un plato, el Jefe de Cocina puede acceder al sistema en ‘Modo Mantenimiento’ de emergencia. En este caso, se ‘congelan’ las operaciones en los terminales y en el Planificador, se cancela la alarma, se desbloquea el plato y se modifica la información de Almacén. Tras confirmación, se continúa con la actividad del Planificador, se actualizan los terminales y se desbloquean. El sistema continúa normalmente.

A la hora de elaborar cualquier aplicación, toda la bibliografía coincide en que «**es fundamental comprender *el negocio* del cliente**». Para eso está el análisis. El hecho de que la asignatura siga el desarrollo a través de un ejemplo, en detalle, ofrece al estudiante la posibilidad de, una vez comprendido el sistema PdV, hacer analogías con un gran número de situaciones y aplicar los conocimientos a esos otros problemas.

En este ejemplo, el negocio que se plantea es la venta de comidas en un restaurante. Por tanto, desde el punto de vista de la gestión de las ventas en sí, la situación es idéntica a la de PdV y, en lugar de artículos, lo que se vende son platos de comida.

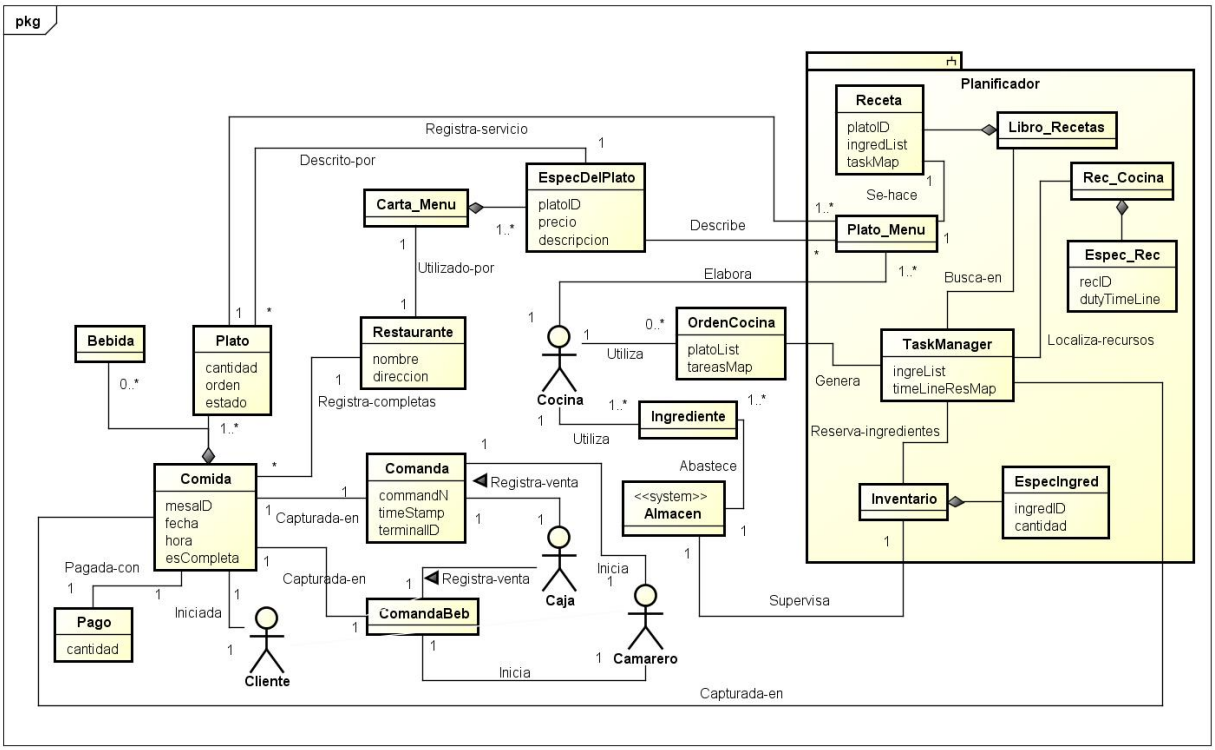
La reflexión a la que obligan estos dos primeros apartados conduce fácilmente a la conclusión de que la comanda (Registro) captura la selección de una comida (Venta), por acción del camarero (Cajero) y a instancias del Cliente. A su vez, cada comida está formada por una lista de bebidas y platos (Líneas de Venta). Al igual que las líneas de venta, cada plato es una información que contiene el plato elaborado y su precio (Especificación del Producto), información que está recogida en el menú-carta (Catálogo).

La singularidad estriba en que los platos (artículos), no están elaborados sino que, una vez pedidos (comanda), se '*fabrican*' en cocina. Continuando con las analogías, se podría considerar que el problema sugiere el desdoblamiento del sistema en dos: el de la venta de comidas y el de la elaboración de los platos (cocina).

El refinamiento de estos conceptos en los siguientes pasos, permitirá resolver el problema.

Evaluación del ***Modelado Conceptual***

3. (2 puntos) En relación al caso de uso anterior <<*RealizarComanda*>>, construya un Modelo de Dominio y represéntelo en notación UML. Represente los objetos conceptuales, las asociaciones y los atributos.



La parte izquierda del modelo es una imagen, *traducida*, de PdV. Para aclarar las ideas, establecer paralelismos y relaciones con el problema, se recomienda empezar por esa parte. Se puede observar que, aunque se selecciona un plato en la carta-menú, la información con la que se maneja el objeto está en su especificación (EspecDelPlato).

Es extremadamente importante comenzar a identificar ya qué información tiene cada elemento (objeto) porque, más adelante, al aplicar el principio de Experto, cada objeto sólo debe manejar la información contenida en él, la que conoce. A su vez, esto favorecerá enormemente el acoplamiento bajo.

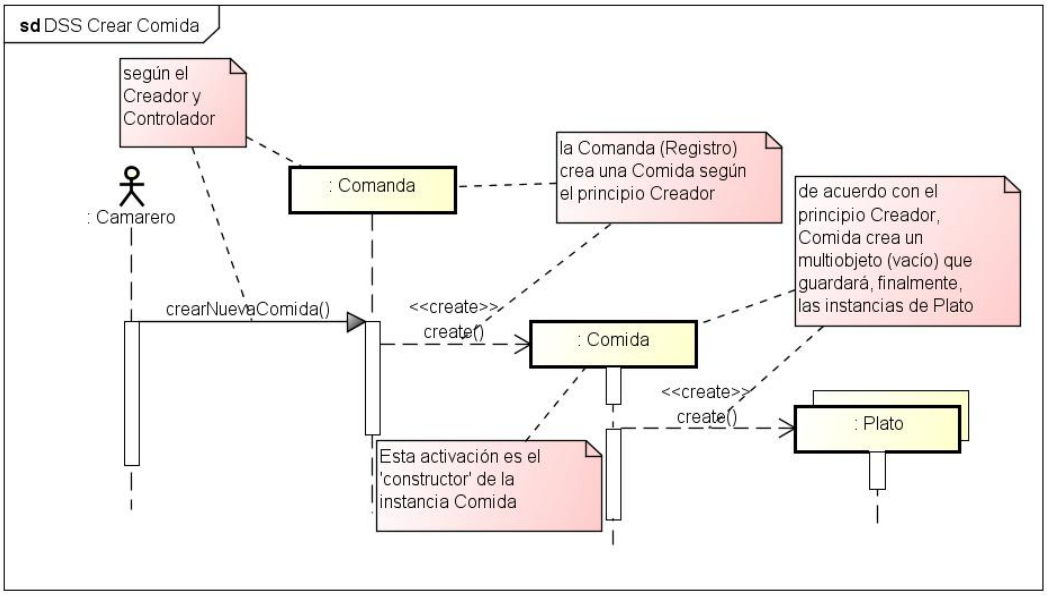
Por consiguiente, si el planificador va a tener como tarea establecer las instrucciones de cocina para la elaboración de cada plato, habrá que pensar qué información necesita. Un plato se elabora con ingredientes, que se procesan de una manera ordenada utilizando ciertos recursos (utensilios) de cocina. Esta descripción coincide con lo que se llama 'receta'. De igual forma que es necesario un catálogo de platos, también lo será un catálogo de recetas (Libro_Recetas). Igual ocurre con el catálogo de recursos de cocina (mapa de ocupación, Rec_Cocina), su especificación (Espec_Rec) y el de disponibilidad de los ingredientes (Inventario) y su especificación (EspecIngred). Todos ellos son objetos permanentes cuya información va a necesitar el planificador, cuando capture la comanda, para generar la orden de cocina.

Evaluación de los ***Eventos del Sistema***

4. (0'5 puntos) Circunscrito al caso de uso anterior *<<RealizarComanda>>*, construya un Diagrama de Secuencia del Sistema (DSS) en UML. Represente los actores y los eventos del sistema. A partir de este DSS, especifique los contratos de dos operaciones principales: *<<se omite la operación A>>* y *<<se omite la operación B>>*.

   ATENCIÓN: de aquí en adelante, lo que hay entre corchetes <<se omite...>> es un ejemplo, usted lo debe sustituir por su propia operación.

   Siguiendo la analogía con Pdv se han elegido *crearComida* e *introducirPlato*, para analizar los importantes roles de Controlador, Creador y Experto.



Operación *crearNuevaComida*

Contrato

Nombre: crearNuevaComida ( )

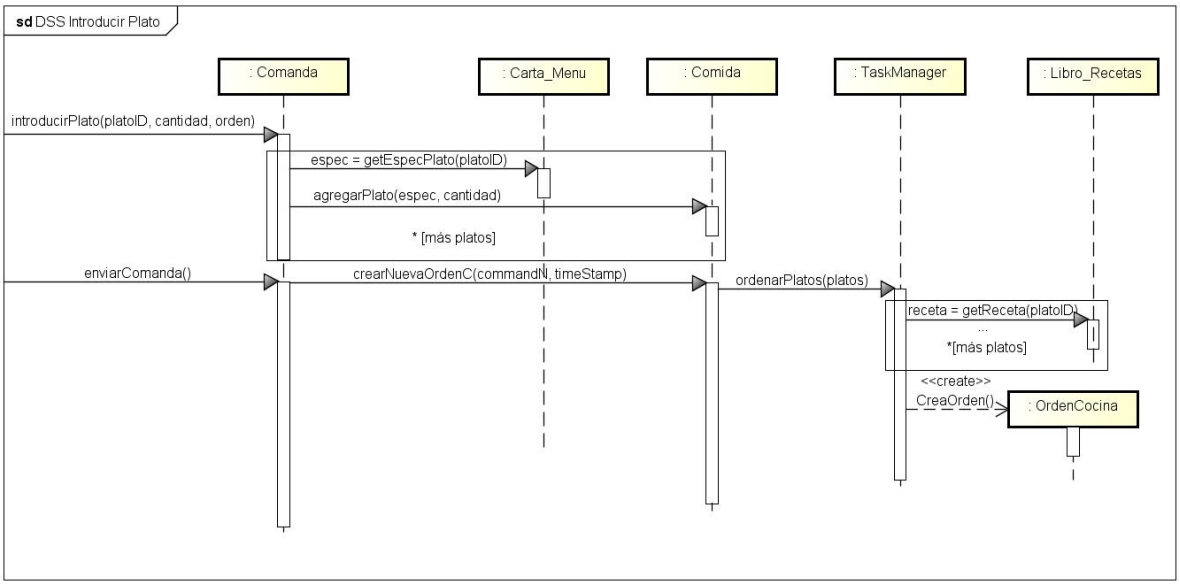
Responsabilidades: Crear un artefacto capaz de contener los elementos de la comida (platos y bebidas). Además, las operaciones realizadas deben poder ser capturadas con el controlador principal (*comanda*, controlador de sesión).

Referencias cruzadas: Caso de Uso: Realizar Comanda

Precondiciones: El controlador *comanda* no tiene en proceso otras transacciones del servicio de comida.

Postcondiciones:

- Se creó una instancia de *Comida* llamada *comida* (creación de instancias).
- *comida* se asoció con la *comanda* (formación de asociaciones) y los comensales (mesaID). Se inicializaron los atributos de *comida* (constructor). Esto significa, en este caso, que se creó una colección vacía de su contenido (*platos*).



Por resultar importante para ilustrar la manera en la que se imbrica la parte del planificador de cocina en el resto del sistema, se ha incluido la operación *enviarComanda()* en el DSS. El evento lo genera el camarero y lo recibe el Controlador *Comanda*. El objetivo será 'disparar' la ejecución de *TaskManager* (mediante la recepción de la lista de platos) para que elabore, con cada plato, una lista de procedimientos, para cocinar el plato, sincronizada con la de los otros platos y con las de las comandas en curso. Al final, debe actualizarse el inventario de ingredientes y el mapa de uso de los recursos de cocina, crear la orden de cocina con un objeto múltiple, compuesto y vacío, que albergará, para cada plato, los pasos de elaboración, ingredientes, cantidades, uso de recursos y tiempos. Como *TaskManager* requiere la lista de platos, el Experto en esa información es, precisamente, *Comida*.

Con estos supuestos, el contrato de la operación *enviarComanda()* se escribiría de manera análoga al de *introducirPlato()*, que aparece a continuación.

Operación introducirPlato

Contrato	
Nombre:	introducirPlato (platoID:PlatoID, cantidad:int)
Responsabilidades:	Registra la selección de un plato y lo añade a la comida curso. Presenta el

nombre del plato, opciones seleccionadas, indicaciones de cocina, número de platos pedidos y precio.

Referencias Cruzadas: Caso de Uso: Realizar Comanda

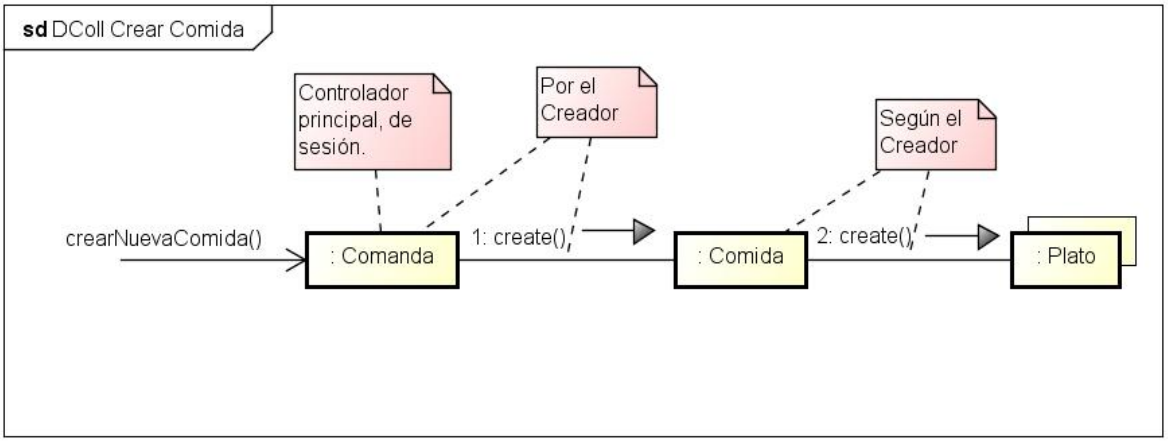
Precondiciones: platoID es un dato conocido para el sistema. Hay una *Comida* que ya se ha creado y está en curso.

Postcondiciones:

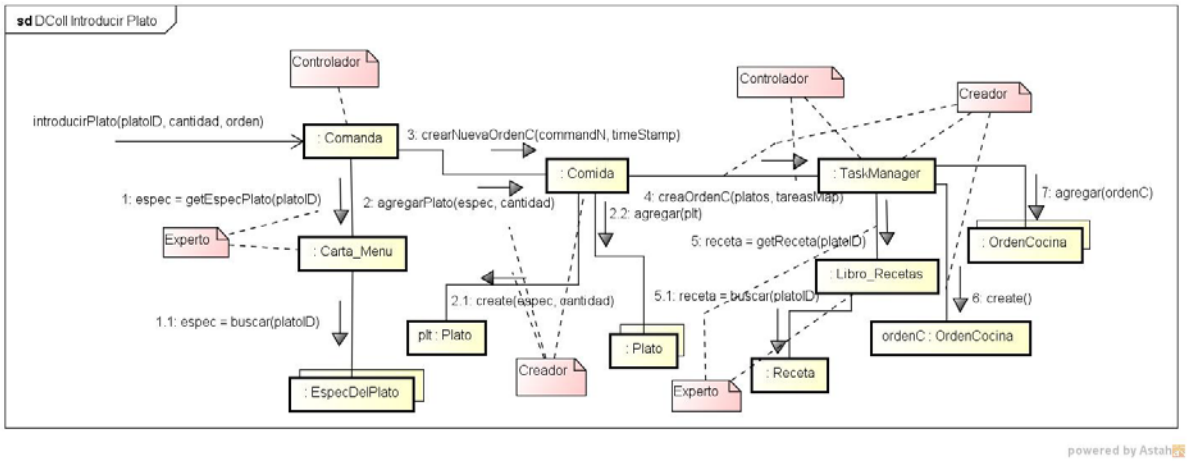
- Se creó una instancia, *plato*, de *Plato* (creación de instancias).
- *plato* se asoció a la *Comida* actual (formación de asociaciones).
- *plato.cantidad* pasó a ser *cantidad* (modificación de atributos)
- *plato* se asoció con una *EspecDelPlato* debido a la coincidencia de su identificador platoID (formación de asociaciones).

Evaluación de la ***Asignación de Responsabilidades*** y ***Diseño de Colaboraciones***

5. (2'5 puntos) A partir del contrato de la operación *crearComida* del punto 4, complete el diagrama de colaboración en UML. Consigne cada mensaje con los patrones GRASP (Experto, Creador, etc.) o cualquier otro que lo justifique. Si añade responsabilidades no explicitadas en el contrato (porque crea que es importante señalarlas), explíquelas brevemente.



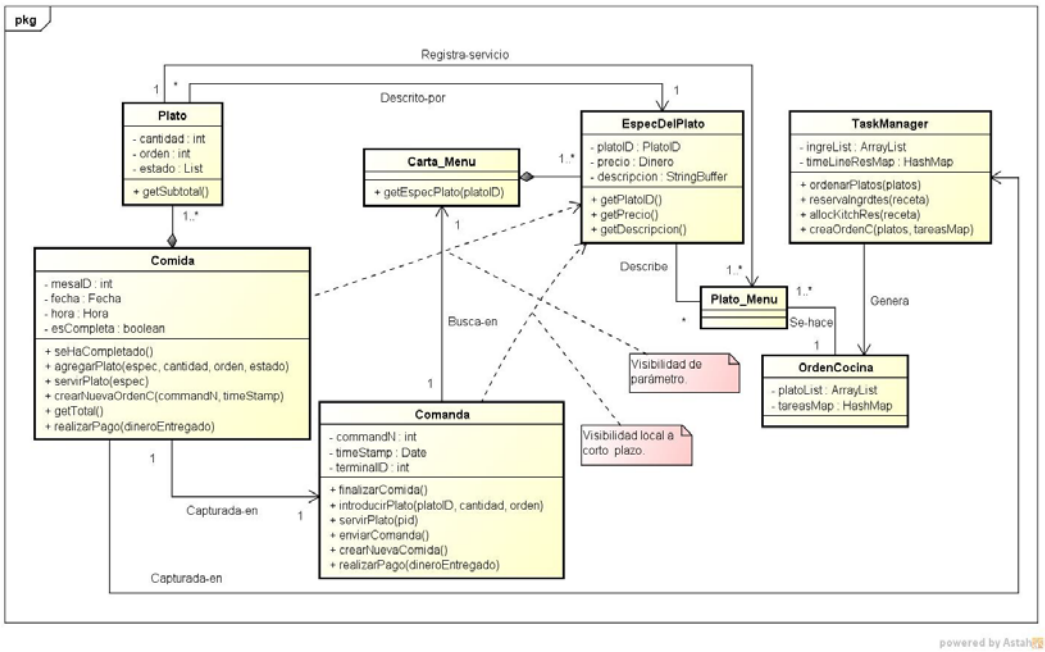
6. (2’5 puntos) A partir del contrato de la operación *introducirPlato* del punto 4, complete el diagrama de colaboración en UML. Consigne cada mensaje con los patrones GRASP (Experto, Creador, etc.) o cualquier otro que lo justifique. Si añade responsabilidades no explicitadas en el contrato (porque crea que es importante señalarlas), explíquelas brevemente.



A partir del mensaje 3, se incluye la operación *enviarComanda()*. El evento lo genera el camarero y lo recibe el Controlador *Comanda* (en el diagrama falta esa flecha, el evento), que lo envía al Experto en la información que necesita *TaskManager*: la lista de platos.

Evaluación de los ***Diagramas de Clases*** de diseño

7. (0’5 puntos) Elabore un diagrama de clases parcial: sólo para las clases <<*Plato*>> y <<*Comanda*>>, analizadas en el Modelo de Dominio. Represente los nombres de todos sus atributos, asociaciones (con la navegabilidad) y métodos.



Evaluación de la ***Transformación del Diseño en Código***

8. (0'5 puntos) A partir de los anteriores diagramas de clases y colaboraciones, elabore y defina la clase <<*Comanda*>>. Incluya las definiciones de todas variables que la componen (miembros), pero escriba solamente la definición completa del cuerpo para el método (función miembro): <<*se omite el método*>>. Ignore los pequeños detalles de sintaxis —el objetivo es evaluar la capacidad fundamental para transformar el diseño en código—. Utilice la sintaxis de Java.

   Aunque sólo se solicita la clase *Comanda* se incluye un desarrollo (mucho más detallado que el pedido) de esa clase y de las clases relacionadas que aparecen en el DCD.

Clase Pago

```
public class Pago
{
    private Dinero cantidad;

    public Pago ( Dinero dineroEntregado )
    {
        cantidad = dineroEntregado;
    }

    public Dinero getCantidad() { return cantidad; }
}
```

Clase Carta_Menu

```
public class Carta_Menu
{
    private Map especificacionesdePlatos = new HashMap();

    public Carta_Menu()
    {
        // datos de ejemplo
        PlatoID pid1 = new PlatoID( 100 );
        PlatoID pid2 = new PlatoID( 200 );
        Dinero precio1 = Dinero( 10 );
        Dinero precio2 = Dinero( 12 );

        EspecDelPlato epl;
        epl = new EspecDelPlato( pid1, precio1, "Plato 1" );
        especificacionesdePlatos.put( pid1, epl );
        epl = new EspecDelPlato( pid2, precio2, "Plato 2" );
        especificacionesdePlatos.put( pid2, epl );
    }
```

```
    public EspecDelPlato getEspecPlato ( PlatoID pid )
    {
        return (EspecDelPlato)especificacionesdePlatos.get(pid);
    }
}
```

Clase Comanda

```
public class Comanda
{
    private int terminalID;
    private int mesaID;
    private int commandN;
    private Date fecha;
    private Time hora;
    private ArrayList timeStamp = new ArrayList();
    private Carta_Menu carta;
    private Comida comida;

    public Comanda( int terminalID, int mesaID, int commandN, Carta_Menu carta )
    {
        this.terminalID = terminalID;
        this.mesaID = mesaID;
        this.commandN = commandN;
        this.carta = carta;
    }

    public void finalizarComida()
    {
        comida.seHaCompletado();
    }

    public void introducirPlato( PlatoID pid, int cantidad, int orden )
    {
        EspecDelPlato espec = carta.getEspecPlato( pid );
        comida.agregarPlato( espec, cantidad, orden, “pedido”);
    }

    public void servirPlato( PlatoID pid )
    {
        EspecDelPlato espec = carta.getEspecPlato( pid );
        comida.servirPlato( espec );
    }

    public void enviarComanda( )
    {
        fecha = new Date(); timeStamp.add(fecha);
        hora = new Time(); timeStamp.add(hora);
        comida.crearNuevaOrdenC ( commandN, timeStamp );
```

```
    }

    public void crearNuevaComida( )
    {
        comida = new Comida ( int mesaID, int commandN );
    }

    public void realizarPago( Dinero dineroEntregado)
    {
        comida.realizarPago( dineroEntregado );
    }
}
```

Clase EspecDelPlato

```
public class EspecDelPlato
{
    private PlatoID pid;
    private Dinero precio;
    private String descripcion;

    public EspecDelPlato (PlatoID pid, Dinero precio, String descripcion)
    {
        this.pid = pid;
        this.precio = precio;
        this.descripcion = descripcion;
    }

    public PlatoID getPlatoID() { return pid; }

    public Dinero getPrecio() { return precio; }

    public String getDescripcion() { return descripcion; }
}
```

Clase Comida

```
public class Comida
{
    private int mesaID;
    private int commandN;
    private List platos = new ArrayList();
    private Date fecha = new Date();
    private Time hora = new Time();
    private TaskManager planificador;
    private boolean esCompleta = false;
    private Pago pago;

    public Comida ( int mesaID, int commandN )
```

```
    {
        this.mesaID = mesaID;
        this.commandN = commandN;
    }
    public void seHaCompletado() { esCompleta = true; }

    public boolean esCompleta() { return esCompleta; }

    public void agregarPlato (EspecDelPlato espec, int cantidad, int orden, String estado)
    {
        platos.add ( new Plato( espec, cantidad, orden, estado) );
    }

    public void servirPlato (EspecDelPlato espec)
    {
        for ( int i = 0; i < platos.size(); i++)
        {
            Plato plato = (Plato)platos.get(i);
            If ( plato.getEspec() = espec )
            {
              plato.setEstado("servido");
              platos.set( i, plato );
            }
        }
    }

    public void crearNuevaOrdenC( int commandN, ArrayList timeStamp )
    {
        planificador.crearOrdenC( commandN, timeStamp, platos );
    }

    public Dinero getTotal()
    {
        Dinero total = new Dinero();
        Iterator i = platos.iterator();
        while ( i.hasNext() )
        {
            Plato plato = (Plato)i.next();
            total.add( plato.getSubtotal() )
        }
        return total;
    }

    public void realizarPago ( Dinero dineroEntregado )
    {
        pago= new Pago( dineroEntregado );
    }
}
```

Clase Plato

```
public class Plato
{
    private int cantidad;
    private EspecDelPlato especPlato;
    private int orden;
    private String estado;

    public Plato (EspecDelPlato espec, int cantidad, int orden, String estado)
    {
        this.especPlato = espec;
        this.cantidad = cantidad;
        this.orden = orden;
        this.estado = estado;
    }

    public Dinero getSubtotal()
    {
        return especPlato.getPrecio().times( cantidad );
    }

    public EspecDelPlato getEspec()
    {
        return espec;
    }

    public String getEstado()
    {
        return estado;
    }

    public void setEstado(String estadoP)
    {
        estado = estadoP;
    }
}
```

Clase TaskManager

```
public class TaskManager
{
    private OrdenCocina ordenC;
    private ArrayList ordenesC = new ArrayList;

    public TaskManager ()
    {

    }
```

```
    public void crearOrdenC( int commandN, ArrayList timeStamp, ArrayList platos )
    {
        ordenesC.add( new OrdenCocina( command, timestamp, platos, tareasMap ) );
    }
}
```

Clase Restaurante

```
public class Restaurante
{
    private Carta_Menu carta = new Carta_Menu();
    private ArrayList comandas = new ArrayList();
    private Comanda comanda = new Comanda ( 1, 3, 0, carta );

    public void crearNuevaComanda( int terminalID, mesaID, Carta_Menu carta )
    {
        comanda = new Comanda (terminalID, mesaID, comandas.size()+1, carta );
        comandas.add(comanda);
    }


    public Comanda getComanda() { return comanda; }
}
```

A continuación se incluye un posible desarrollo para la clase *Dinero*, con abstracción para la gestión de la divisa.

Clase Dinero

```
import java.util.*;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.math.MathContext;

/**
 * Class Dinero - clase la divisa de los pagos.
 * Escrito en BlueJ para el examen de febrero 2013.
 * @author  José Félix Estívariz
 * @version 1.0, marzo 2013
 */
public class Dinero
{
    // inicializa variables de la instancia
    private double cantidad;
    private Locale pais;              //init-> = new Locale("es");
    private Currency divisa;          // init-> = Currency.getInstance(pais);

    /**
     * Constructor para objetos de la clase Dinero
     */
    public Dinero ( double nuevaCantidad )
    {
        // inicializa variables de la instancia
        this.cantidad = nuevaCantidad;
        this.divisa = divisa;
    }

    /**
     * Métodos.
     */
    /**
     * Cambia la cantidad de dinero.
     */
    public void setCantidad ( double nuevaCantidad )
    {
        cantidad = nuevaCantidad;
    }

    /**
     * Añade una cantidad de dinero.
     */
    public Dinero add(Dinero nuevaCantidad)
    {
        //cantidad += nuevaCantidad;
        //        |de double a BigDecimal|     |de double a BigDecimal| |de Dinero a double| |de BigDecimal a double|
        cantidad = (new BigDecimal(cantidad)).add(new BigDecimal(nuevaCantidad.getCantidad())).doubleValue();
        return new Dinero(cantidad);
    }
```

```
    /**
     * Multiplica la cantidad de dinero.
     */
    public Dinero times(int veces)
    {
        // BigDecimal veces2 = new BigDecimal(veces, new MathContext(2));
        // BigDecimal cantidad2 = new BigDecimal(cantidad);
        // cantidad2 = cantidad2.multiply(veces2);
        // cantidad = cantidad2.doubleValue();
        cantidad = (new BigDecimal(cantidad)).multiply(new BigDecimal(veces, new MathContext(2))).doubleValue();
        return new Dinero(cantidad);
    }

    /**
     * Devuelve la cantidad de dinero.
     */
    public double getCantidad()
    {
        return cantidad;
    }

    /**
     * Cambia la divisa.
     */
    public void setDivisa(String nuevoPais)
    {
        pais = new Locale(nuevoPais);
        divisa = Currency.getInstance(pais);
    }

    /**
     * Devuelve la divisa de Dinero.
     */
    public String getDivisa()
    {
        return divisa.getCurrencyCode();
    }
}
```