

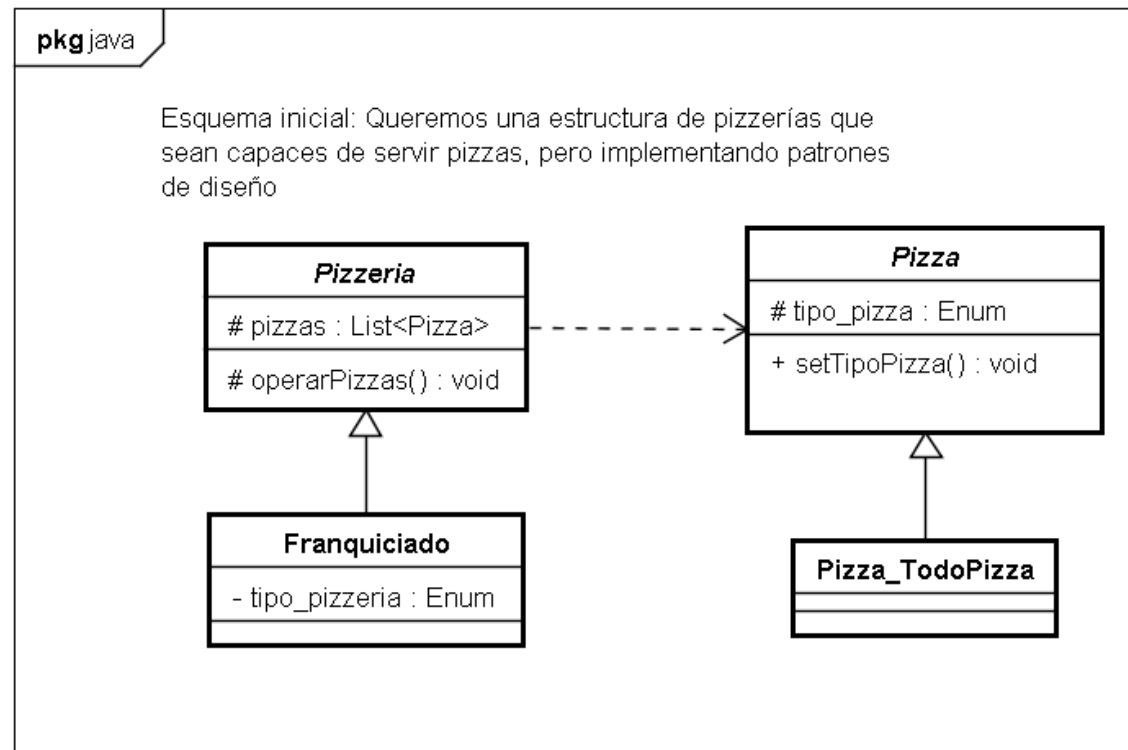
Práctica de Diseño de Aplicaciones Orientada a Objetos – Junio 2012

GONZALO SÁNCHEZ PLA

Introducción

La primera parte de la práctica consiste en crear unas clases sencillas con las que podamos crear las pizzerías de la franquicia TodoPizza, las cuales deben encapsular la gestión de las pizzas.

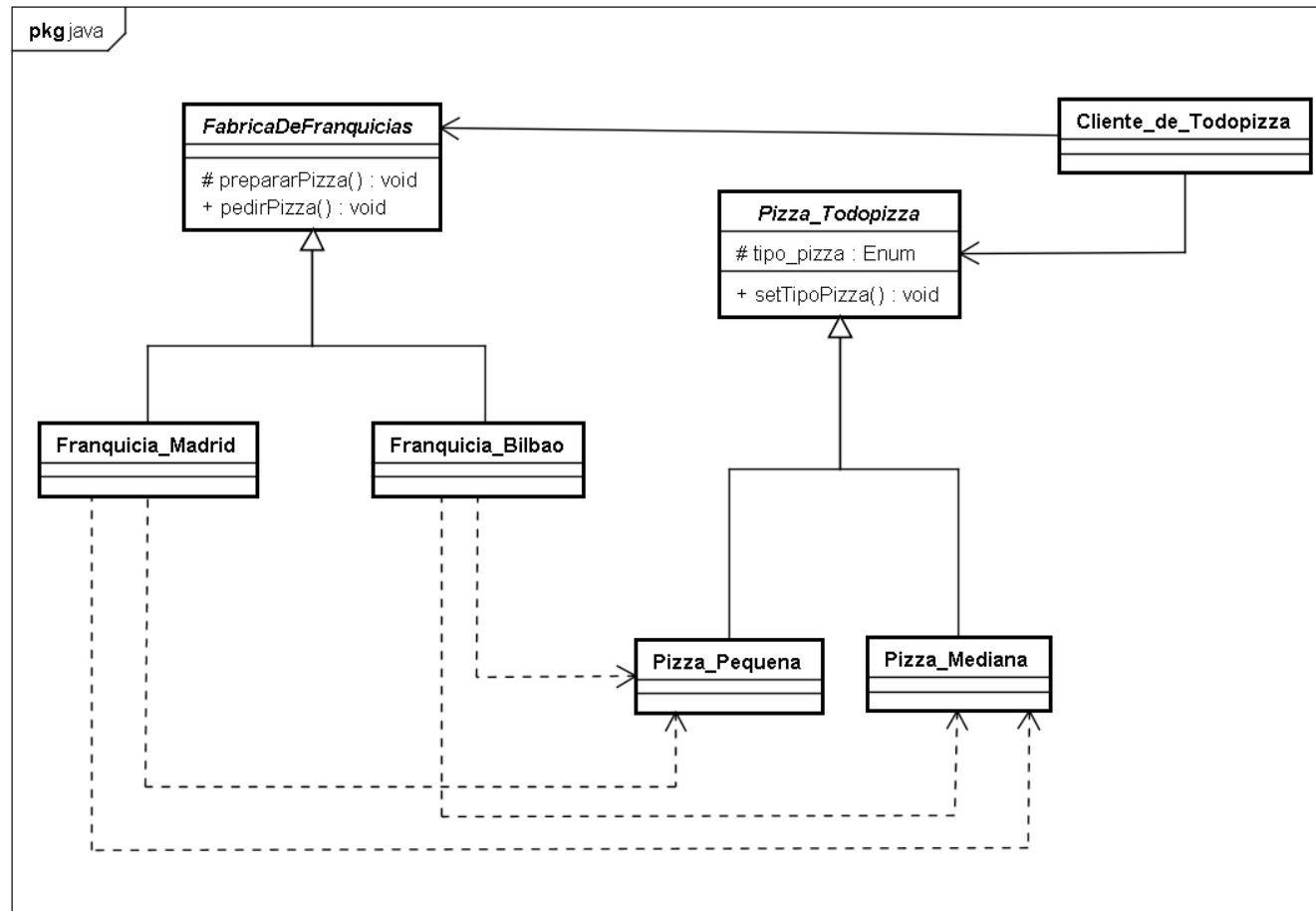
El diagrama siguiente muestra un borrador inicial de una primera propuesta elemental con cuatro clases básicas que aún no incorporan ningún patrón de diseño de los propuestos.



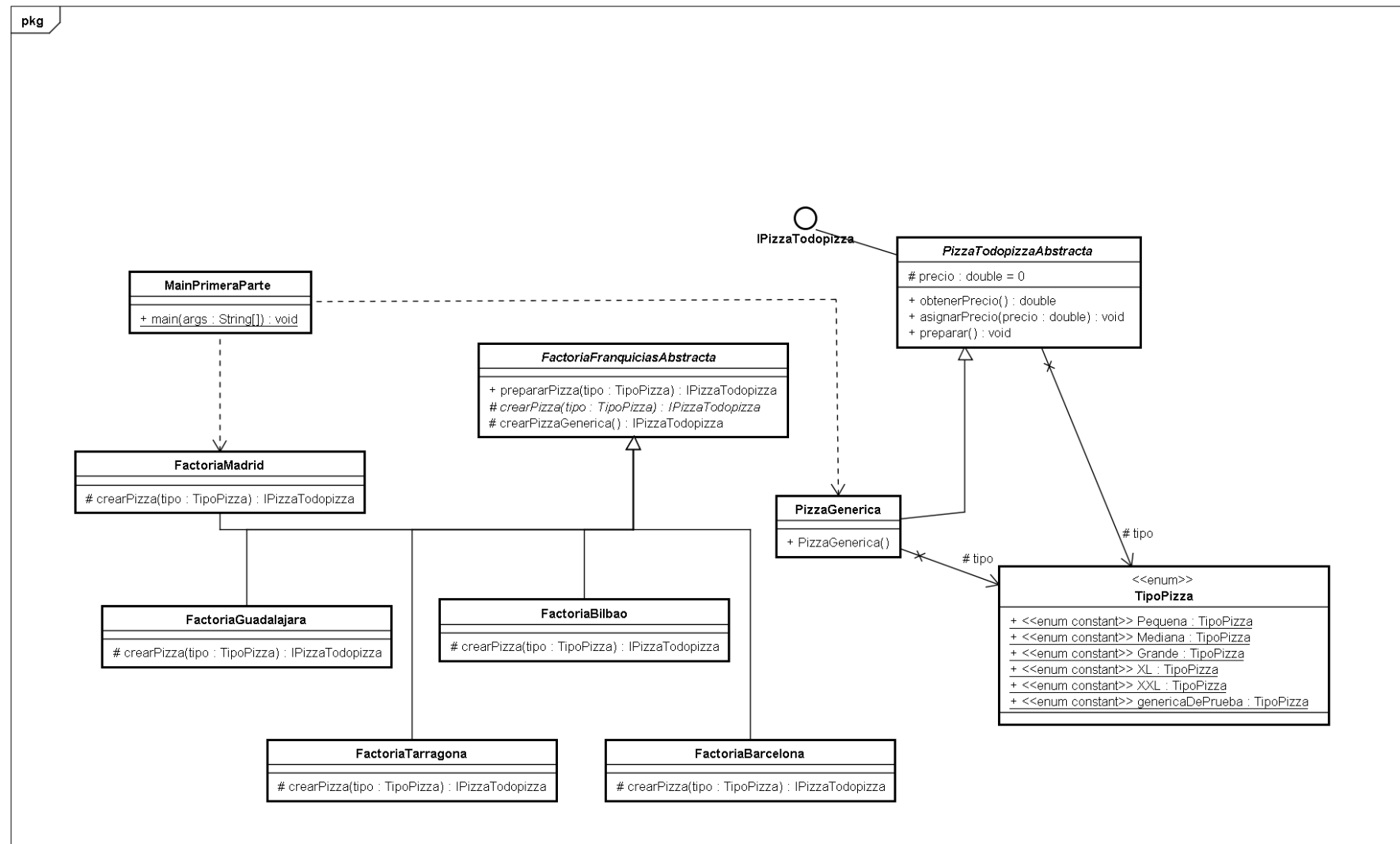
Esquema básico de la funcionalidad buscada

Parte 1

Queremos poder **crear distintas pizzerías franquiciadas** de TodoPizza, que tendrán características diferenciadas y que a su vez deberán respetar el modelo de negocio de TodoPizza. Estamos hablando una familia de patrones de diseño de **creación** sobre los que tendremos que decidir. Un patrón adecuado para resolver la creación de pizzerías **es Abstract Factory o Kit**, ya que prepara una interfaz para crear una familia de objetos relacionados, pero sin terminar de especificar la clase concreta a la que pertenecerán, y se acopla bien con el patrón Factory Method relacionado con la generación de productos desde la factoría.



La primera parte de la práctica, ya desarrollada, muestra este esquema:



Cada franquicia puede generar un tipo de producto (pizza), que en este caso hemos definido como abstracto, aunque a la hora de implementarlo, tendremos que recurrir al patrón *Factory Method* para que sea la propia pizza la que implemente la creación de las pizzas de su familia.

El cliente no tiene porqué conocer la división interna de las pizzerías y pizzas, y puede ser desde una web de pedidos a una persona que acude a una pizzería. El cliente representa el mundo exterior a la propia estructura de pizzerías y pizzas, y puede ser cualquier agente interesado en el uso de las dos clases principales: Franquicias y pizzas.

El **test de esta primera parte** está en el paquete **main**, en el cual hay una clase **Main()** la cual al instanciarse crea una pizzería mediante el patrón *Abstract Factory*. Tras ello esta pizzería crea una pizza especial de tipo *genérica* de un modo trivial sin implementar ningún otro tipo de patrón remarcable.

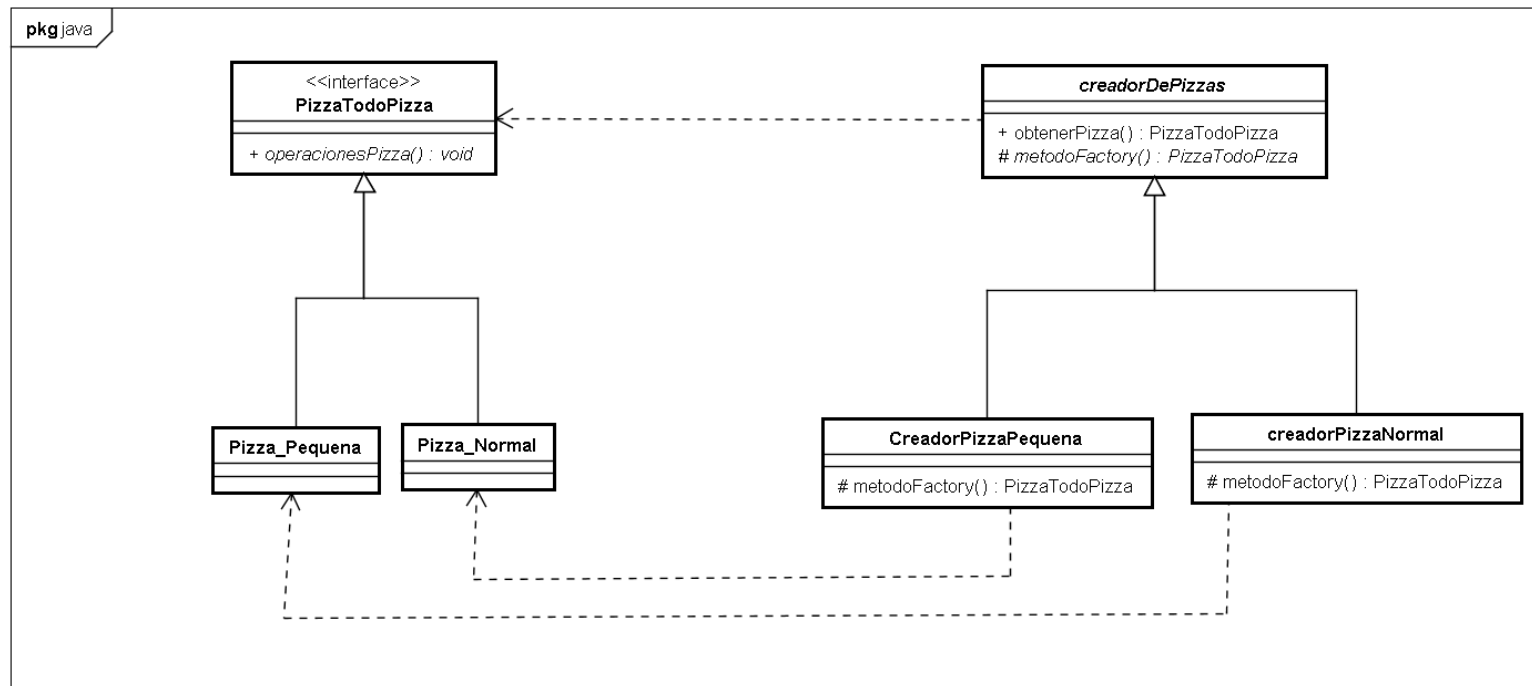
A efectos de la práctica, la superclase del modelo Abstract Factory declara un método abstracto que es el que tendrán que implementar las subclases de tipo Factory y se llama *crearPizza*.

Debido a que solo se está mostrando el patrón abstract factory y que todas las pizzas son iguales en todas las pizzerías de la franquicia, no se llega a apreciar mucho la filosofía de este patrón, que sería que pudiéramos hacer distintas implementaciones de *crearPizza* para cada franquicia.

Un caso muy genérico de fábricas abstractas son las clases de acceso a bases de datos, que implementan los métodos de acceso a datos utilizando las cadenas de conexión y los comandos DDL, DCL, DML y TCL propios de cada gestor de bases de datos cuando estos difieren del estándar ANSI.

Parte 2

Utilizaremos dos patrones. Primero se delegará en distintas subclases decidir qué tipo de pizza se crea. Para ello, tenemos el patrón **Factory Method**, que precisamente proporciona una forma de delegar en subclases los tipos (de productos) a crear, como punto de partida de la segunda parte de esta práctica y complementa la construcción de la factoría de pizzas y los patrones de creación de la práctica.

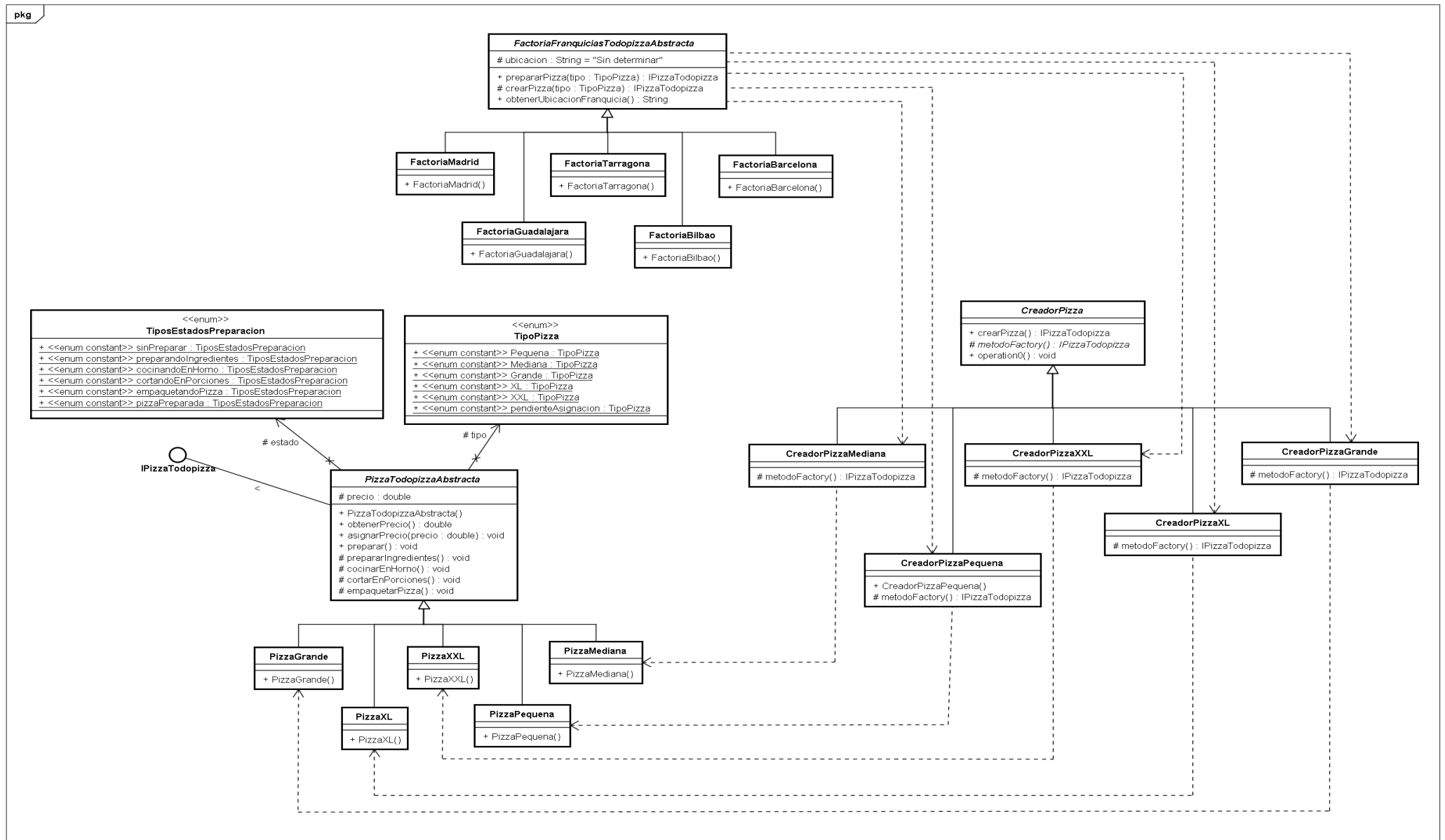


Factory Method Todopizza

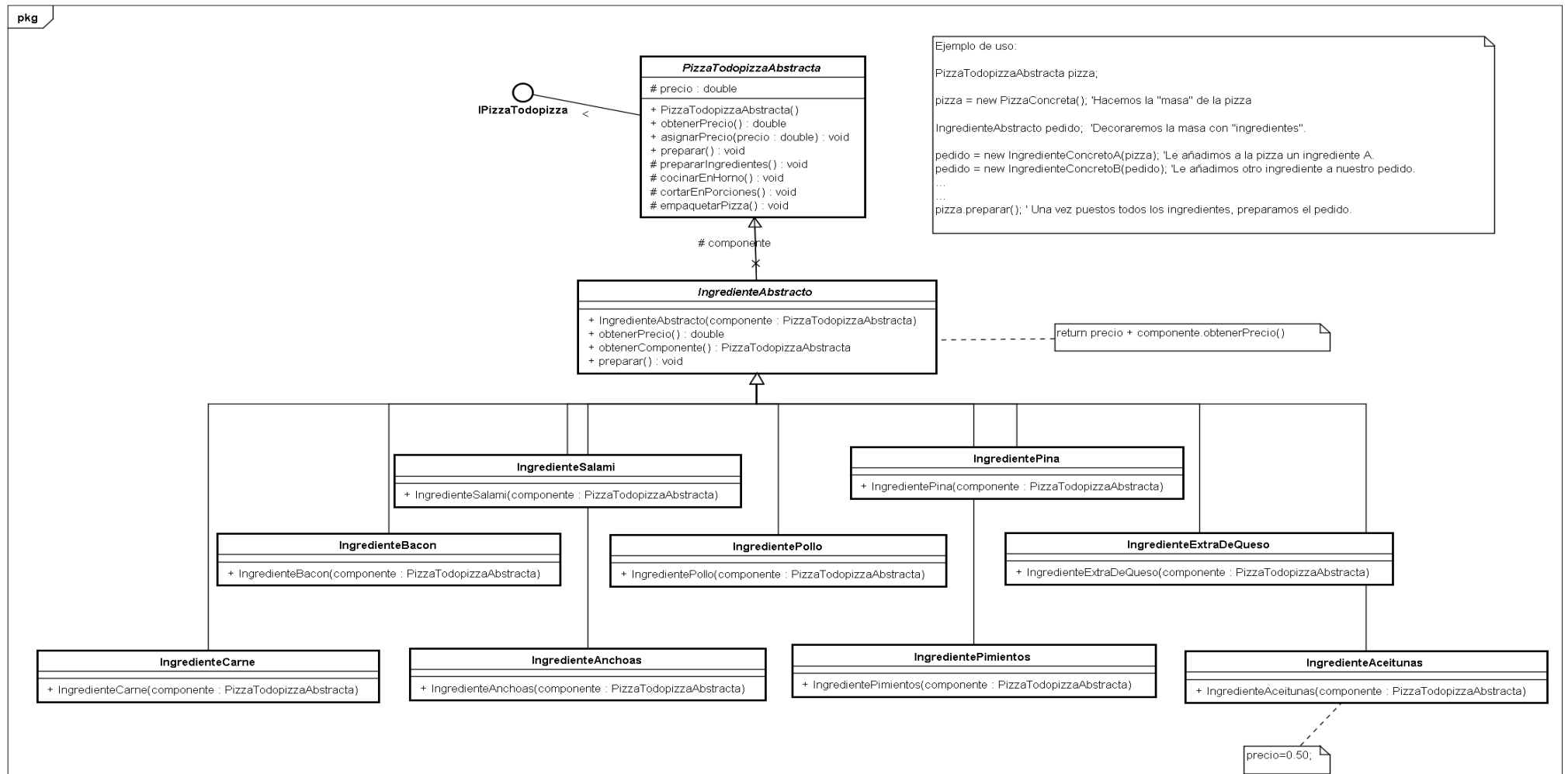
Ahora, en vez de acudir directamente a la superclase o interface que represente las pizzas de la franquicia, nuestro objeto de franquicia llamará a un creador de pizzas, el cual implementará un catálogo de pizzas según el tipo de creador que escojamos y asignemos para cada franquicia.

Como la creación de pizzas incluye la gestión de ingredientes, también será conveniente implementar un **patrón estructural**, en este caso el **Decorator** para los ingredientes. El motivo de escoger *decorator* es que se pretende que la adición de ingredientes a la base sea transparente a la clase que representa las pizzas y que se realice de manera análoga a una superposición de capas.

La figura de la página siguiente ilustra el esquema de funcionamiento del par Abstract Factory con Factory Method implementado (tiene buena resolución, hacer zoom si no se distingue algún detalle).



El patrón **decorator** lo implementamos de este modo:



El componente (ingrediente) del patrón hereda la clase que va a decorar para no tener que modificarla y que implemente las modificaciones del decorador.

El componente guarda una referencia al objeto que decora, de manera que estamos definiendo una secuencia sencilla, a la que añadimos un método que de forma recursiva podrá acceder a las clases envueltas con anterioridad. Útil para calcular el precio, ver de forma individualizada los ingredientes, etc.

Además de los patrones, en esta parte de la práctica también se implementan las fases de la preparación de la pizza y el tiempo de cada una de ellas. La forma es a solicitando al hilo donde se ejecuta el programa que duerma el tiempo requerido (`Thread.sleep(int milisegundos)`).

Al igual que en la primera parte, se ha creado un paquete llamado "main" que contiene una clase que se puede instanciar para hacer una prueba.

Parte 3

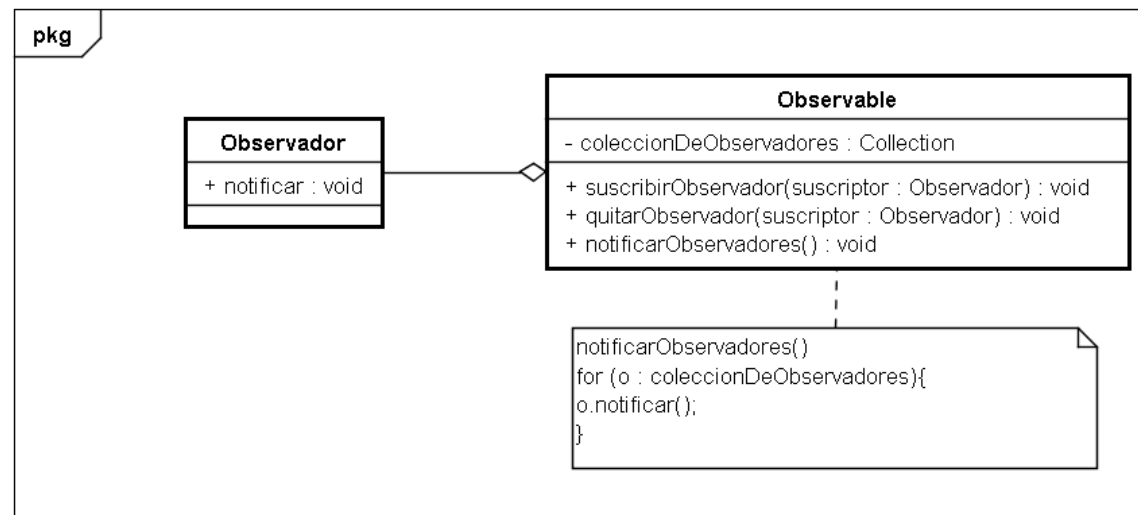
En los apartados anteriores hemos realizado los siguientes desarrollos:

- 1) Creado una fábrica de pizzas (las franquicias) mediante el patrón Abstract Factory.
- 2) Dotado de un método fábrica (Factory Method) a las pizzas para gestionar mejor su producción en las fábricas de pizzas.
- 3) Implementar un patrón Decorator para añadirle ingredientes a nuestras pizzas.

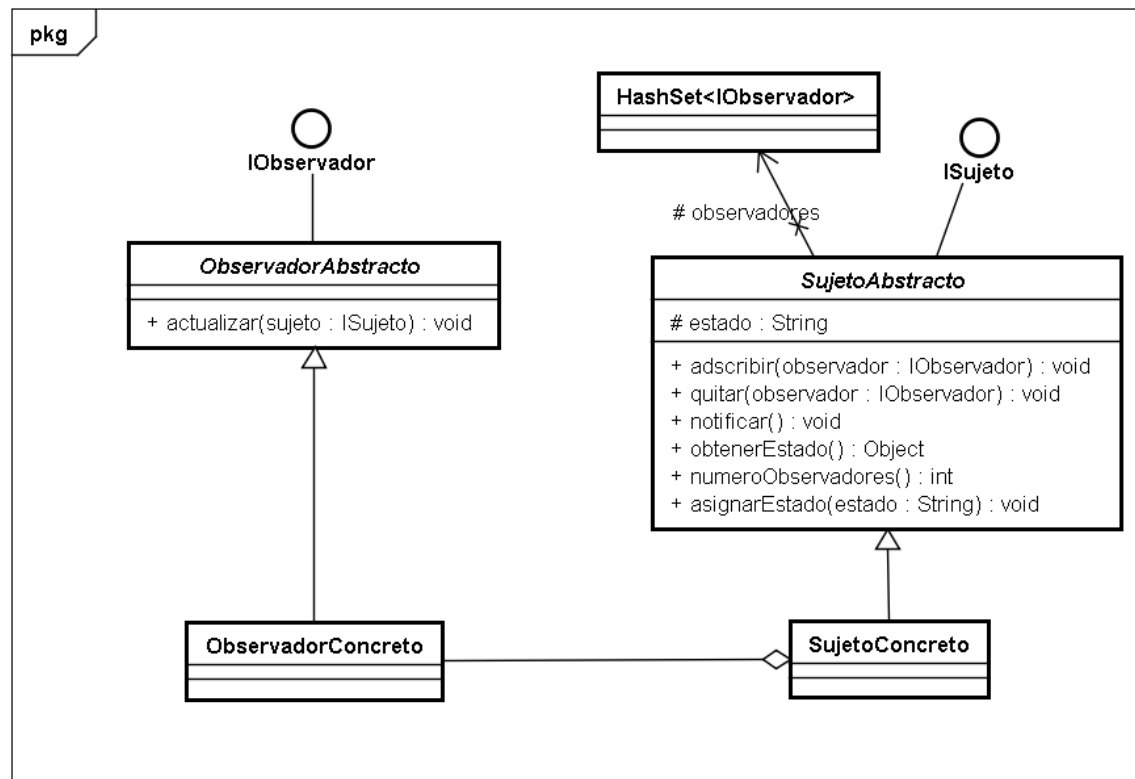
En el presente apartado vamos a demostrar cómo crear un modo de comunicación entre la producción de pizzas y las fábricas de pizzas para que en todo momento las fábricas conozcan de los cambios de estado en el proceso productivo que afectan a cada pizza individual.

Para no redundar la información, nos centraremos solo en la implementación del modelo y el canal de comunicación, representando de forma muy esquemática el resto de tipos.

El tipo de patrón de diseño que necesitamos es de la familia de patrones de **comportamiento**, y en concreto **el patrón Observer**, porque necesitamos que un objeto notifique a otros sus cambios de estado. El esquema genérico del patrón observer sencillo es:



Tal como se ha indicado, vamos a representar de manera muy simple las clases no relativas a este patrón para hacer más entendible el diagrama y la clase de prueba, que es la clase **Main** del paquete **main** de este proyecto.



Patrón Observer de Todopizza preliminar

En la práctica:

- 1) El observador será la clase factoría, cuya clase abstracta extenderá **ObservadorAbstracto**.
- 2) El sujeto observable será la clase pizza (producto) cuya clase abstracta extenderá **SujetoAbstracto**.

Otra opción sería generar clases internas, tal como se ha hecho en el ejemplo de esta parte de la práctica, aunque debido a la sencillez del proyecto no parece necesario y por ese motivo simplemente haremos uso de la herencia de clases para encapsular el comportamiento dentro de cada agente.

Parte 4 – Integración

En esta fase, la clase Fábrica heredaré observador y la clase Pizza (y componente a través de Pizza) heredaré observable.

El resto de patrones: Abstract Factory, Factory Method y Decorator ya han sido implementados integrándose uno con otro en las distintas partes de la práctica, tal como se ha visto en cada refactorización.

Lo que nos queda es mostrar el diagrama de clases relativo a la implementación de observador y adaptar un poco mejor este patrón a la filosofía que he ido siguiendo durante el desarrollo del programa.

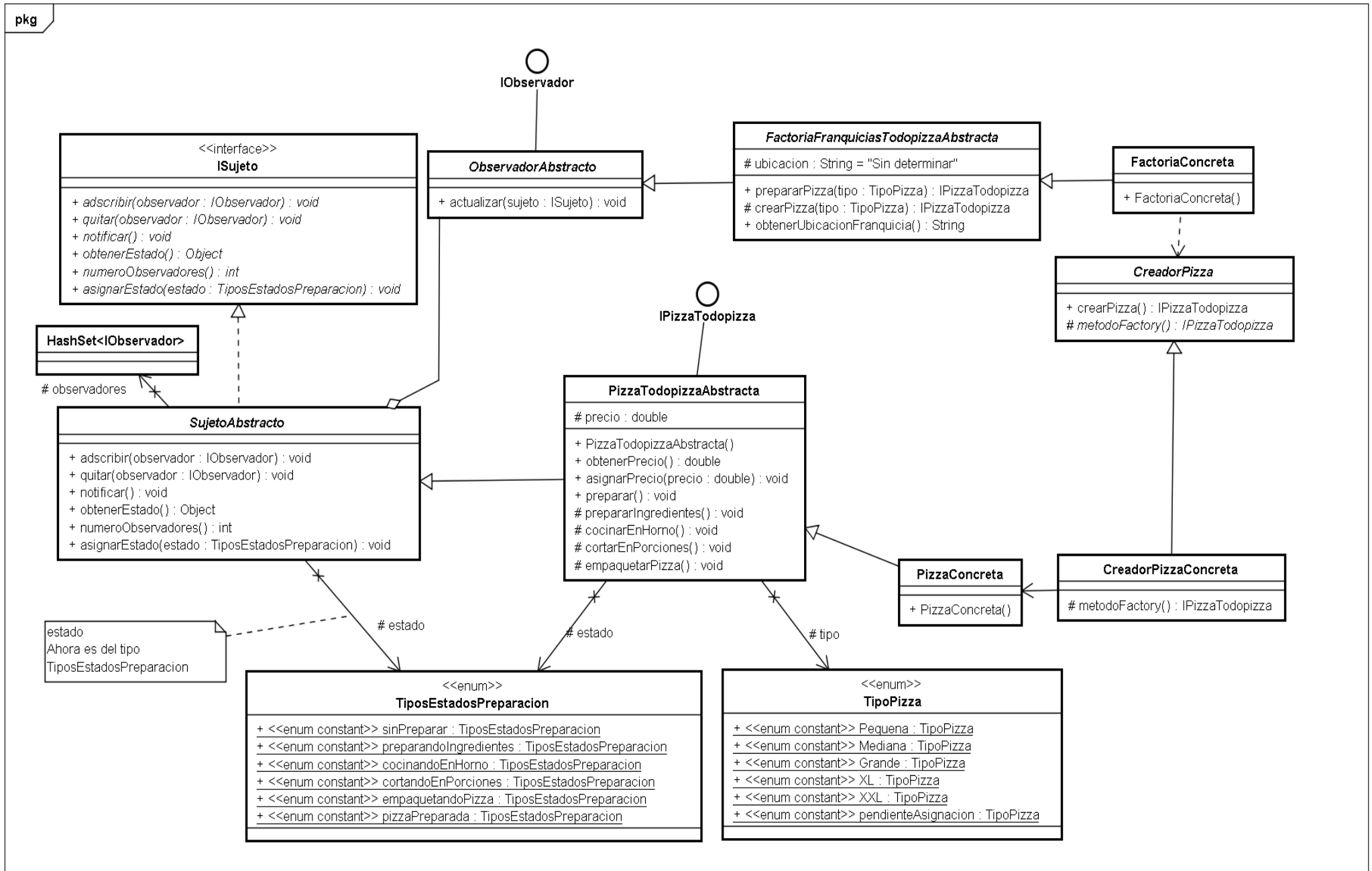
El primer planteamiento a la hora de abordar esta parte ha sido **cambiar los estados notificables** a enumerados en vez de cadenas de texto, a **desacoplar** un poco más las clases y a **prepararnos** la práctica **para** la posterior incorporación de la **interfaz gráfica** en Swing.

La clase pizza pasa a heredar hereda la implementación del sujeto observable y el sujeto observable cambia su tipo de estado, de String a un enumerado igual que el que utilizan las clases fábrica y pizza para describir los estados de preparación de las pizzas.

A su vez, las fábricas de pizza heredarán la clase Observador de manera que puedan recibir las notificaciones de los sujetos. Este planteamiento se cambiará conforme se avanza en la cuarta parte, delegando la responsabilidad de gestionar los cambios de los sujetos observables a una nueva clase y quitándole a la fábrica esta responsabilidad, ya que desde un punto de vista conceptual, la fábrica debe precisamente producir, y como mucho ser transmisora de cambios, pero no gestora de los mismos de cara al cliente que solicita la pizza.

En la refactorización de esta parte se cambia la gestión de cambios a una nueva clase y se mantiene al margen de ello a la clase fábrica (franquicias), que en un refinamiento posterior sí que debería ser, al menos, notificada de los cambios del producto (con un modelo que incluya, por ejemplo, una clase de **gestión de cambios** del patrón observer) para hacer más consistente el diseño.

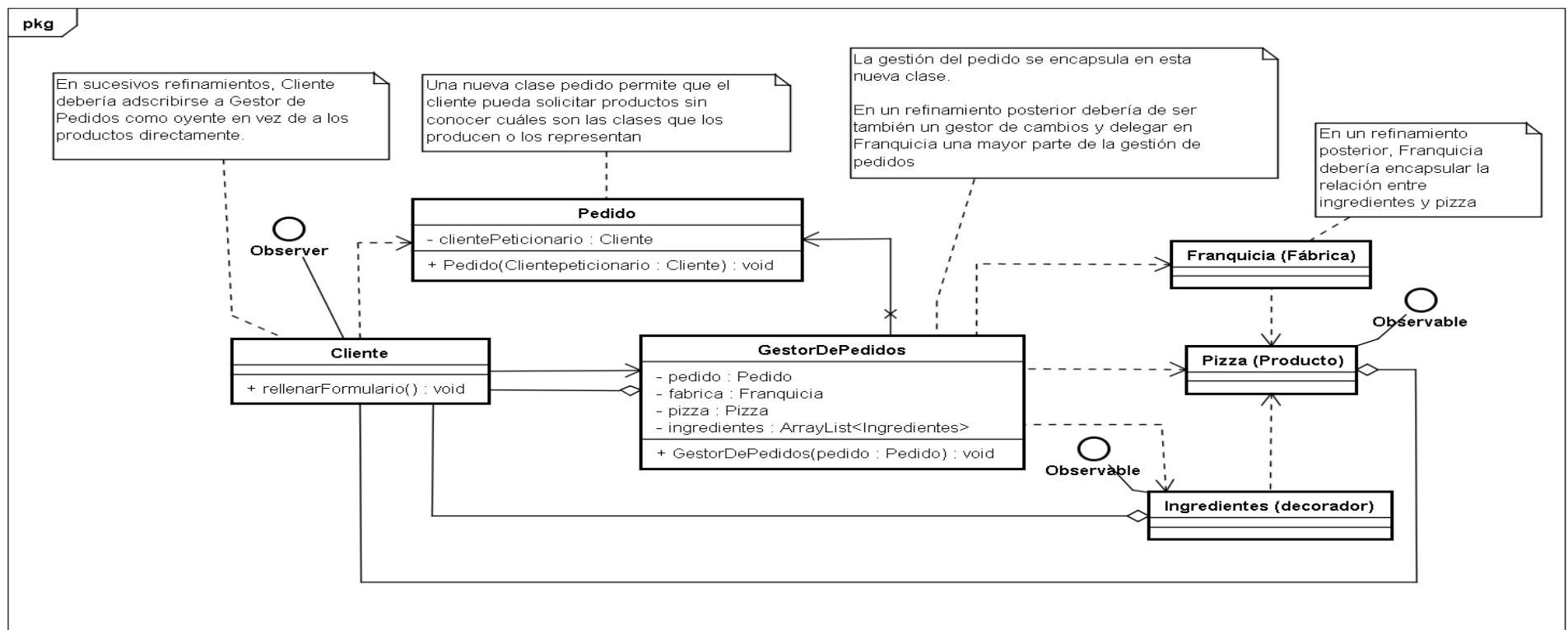
En el diagrama de la página siguiente se puede apreciar la nueva estructura tal y como se plante por primera vez al iniciar esta fase de la práctica, cuando aún la clase fábrica abstracta se intuye como responsable de recibir directamente las notificaciones de los cambios en las pizzas, sea a través del objeto pizza o bien de los objetos decoradores ingredientes.



A efectos de funcionalidad, el esquema anterior se demuestra eficaz, aunque requiere que la clase que gestione la estructura realice muchas funciones que deberían estar encapsuladas, como la creación de la factoría o las pizzas. En el siguiente refinamiento se modifica el esquema anterior para dar paso al definitivo de esta parte, el cual se detalla a continuación.

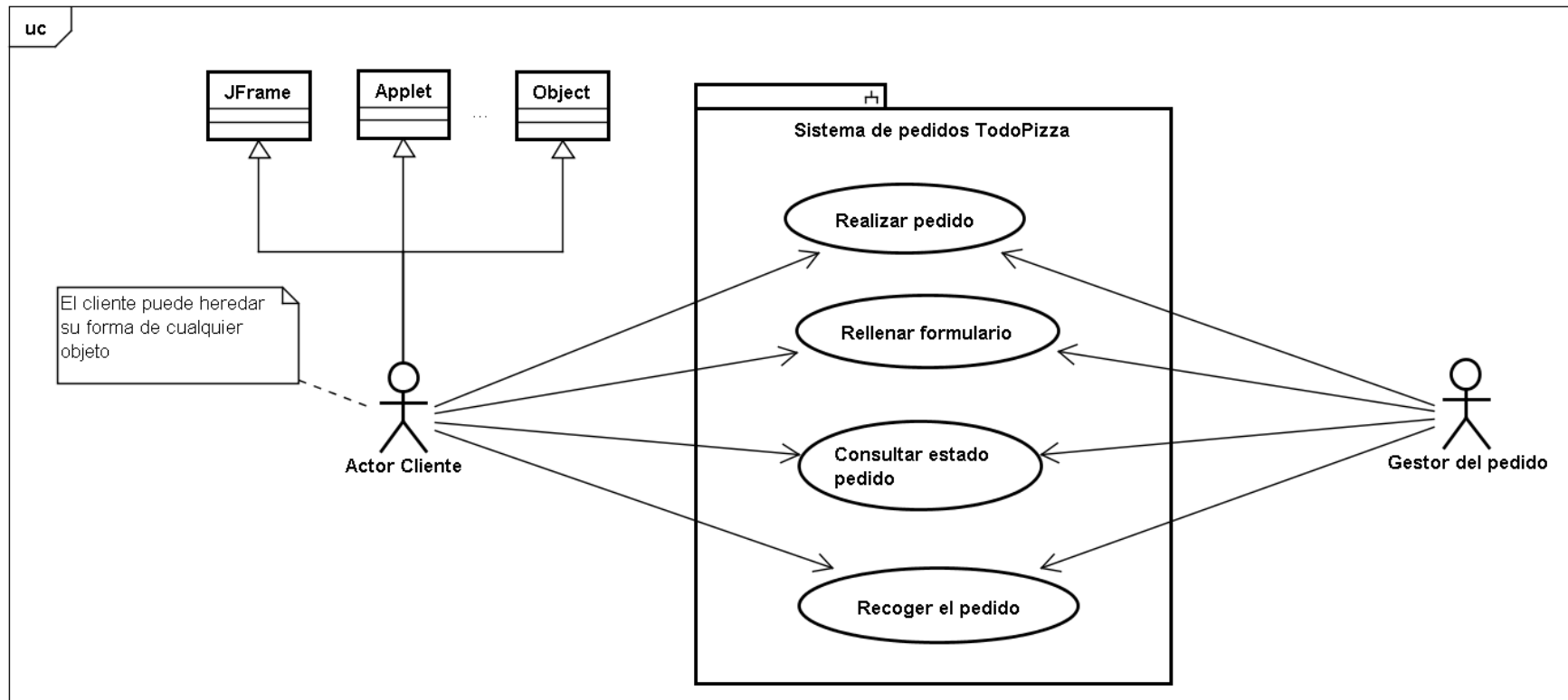
Si pensamos en un patrón de N capas, tan usual hoy en día, tenemos que reconsiderar la estructura de nuestras clases. Y también debemos recordar que tenemos en el siguiente paso que implementar una interfaz gráfica, lo que aproxima el diseño a un sistema de MVC de tres capas basado en objetos y eventos. Ahora mismo tenemos los datos separados mediante enumerados, la lógica concentrada en las clases, y hemos tomado la decisión de desacoplar el cliente del resto de clases.

El cliente debería saber qué pizzas existen por referencia, no por valor, dónde las puede solicitar y qué ingredientes combinar; los precios y el curso de su pedido. Por analogía con el mundo real, deberíamos facilitarle un **formulario de pedido**. Y dado que el cliente no tiene porqué conocer los procesos internos de nuestra franquicia, deberíamos también de crear **una clase** capaz de **solicitar a la fábrica que realice los pasos necesarios** y los encapsule, de manera que no tengamos que programarlos en la clase que utilicemos como punto de entrada al programa, tal como se ha hecho hasta ahora en los tests, lo cual se aproxima más a la programación estructurada que a la programación orientada a objetos. Con este nuevo enfoque se ha desarrollado la integración, dejando para refinamientos sucesivos varios aspectos que se detallan en el diagrama global siguiente:



En mayor o menor medida casi todas las clases se han cambiado, por lo que es conveniente repasar el estado de las mismas.

El punto de entrada es un objeto cliente y la filosofía tal como se muestra en el diagrama anterior es que la implementación esté desacoplada del cliente.

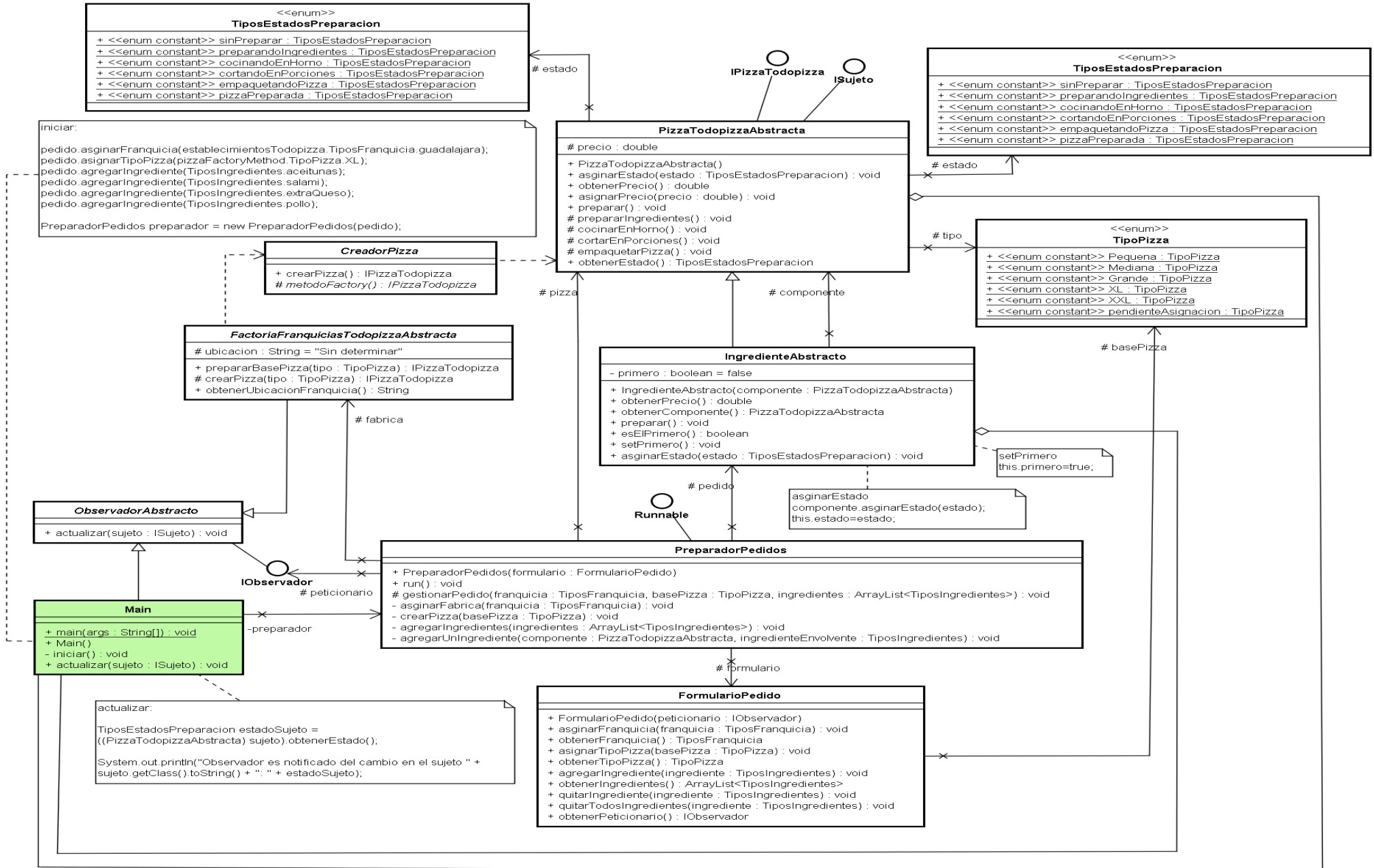


En concreto, vamos a repasar los principales cambios realizados en detalle.

De nuevo, se ha creado un paquete llamado **main**, con una clase **Main** que es el punto de entrada a esta parte de la práctica y ejerce de Actor cliente.

El diagrama de la página siguiente muestra los cambios tras la última refactorización de esta parte de la práctica. La clase gestora de pedidos se ejecuta a sí misma a través del constructor. En una siguiente refactorización será provista de un método adicional para poder decidir cuándo debe empezar a procesar un pedido. Asimismo, las fases de producción que se notifican ocurren después de que se genere realmente el pedido, por lo que habría también que refinar ese punto en el futuro. Esto es transparente, no obstante para el cliente, y no afecta al *look and feel* de la práctica.

pkg

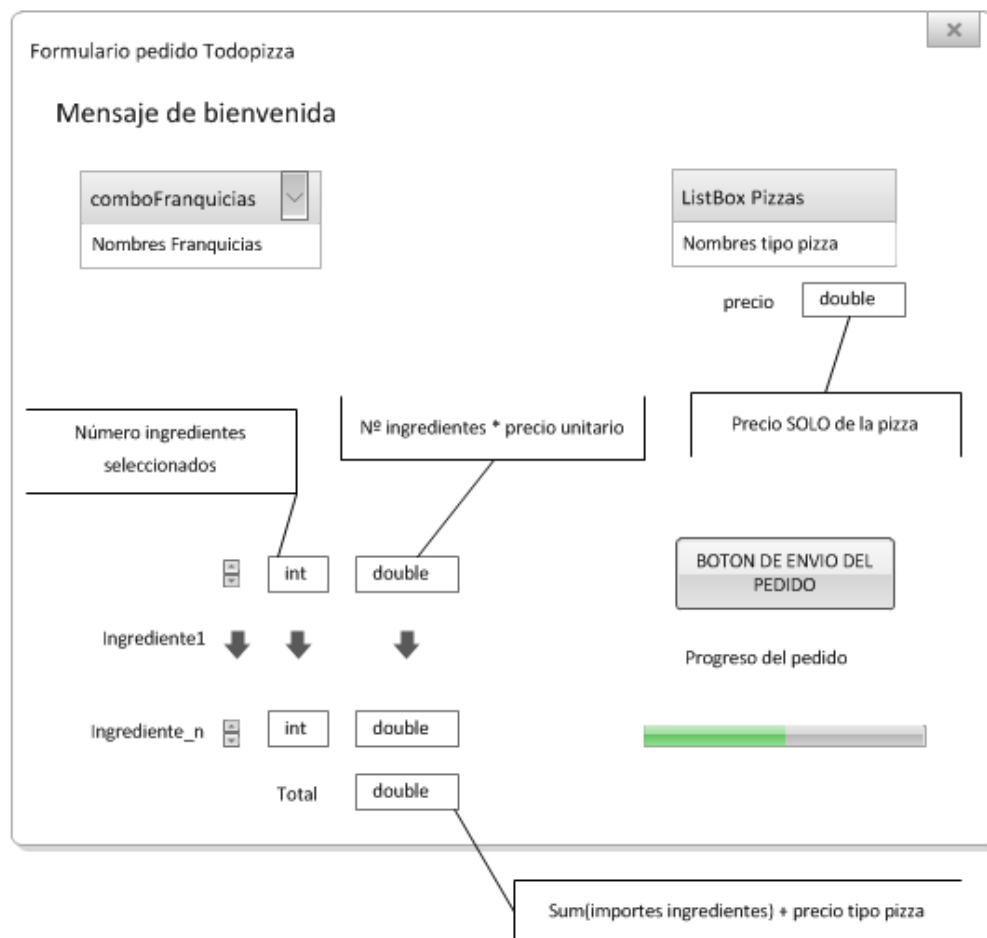


Parte 5 Interfaz gráfica

Para seguir con la optimización del código, aunque este apartado no requiere de un patrón de diseño determinado, he intentado mantener lo mejor diseñada la aplicación, aplicando multitud de variaciones (respetando siempre los patrones de diseño implementados) y ampliando el programa.

En especial he creado varias clases que adaptan el formulario y sus eventos a la lógica de programa ya realizada y he realizado un diseño separado por capas lo máximo posible de una forma razonable para la práctica.

El prototipo para la IGU está constituido por un formulario (JFrame) y tres diálogos, dos de los cuales tienen la misma forma. El formulario principal parte del siguiente boceto:



El boceto está dividido en cuatro áreas funcionales:

1. Selección de la franquicia.
2. Selección del tipo de pizza
3. Selección de los ingredientes.
4. Envío y confirmación del pedido.

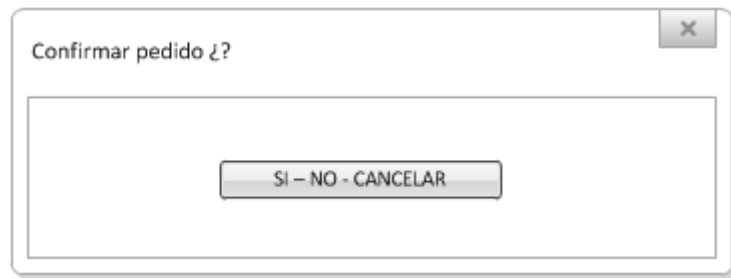
Todas las partes se basan en la premisa de que el usuario necesite la menor concentración posible para realizar el pedido y de esa manera se minimice la probabilidad de error.

Por ello, el único error posible que puede cometer el usuario es cerrar involuntariamente el formulario o confirmar el envío del pedido habiendo seleccionado por error algo que en realidad no desea. Es decir, sólo debe estar pendiente de dos detalles.

Para conseguirlo he encapsulado la mayor parte de la funcionalidad en el código de la siguiente manera:

1. El usuario no tiene que rellenar ningún campo de texto (TextField). El programa se encarga de calcular todo.
2. El usuario no tiene que escribir nada. El programa se encarga de traducir a lenguaje humano todos los objetos del pedido y de facilitar los controles para que el usuario sólo deba seleccionar lo que desea con un clic de ratón.
3. El botón de confirmar pedido sólo se activa cuando hay una pizza, un ingrediente y una franquicia seleccionadas y deja de estarlo si se no hay ninguno de estos elementos seleccionados. La pizza pequeña está seleccionada por defecto.
4. Cada acción del usuario es verificada mediante cuadros de diálogo para que el usuario pueda corregirla si es necesario.
5. El botón de confirmar pedido se desactiva de forma automática cuando se confirma el envío de pedido. Como la ventana de confirmación es modal, no hay peligro de que el usuario utilice el formulario principal hasta que no confirme el envío.
6. Todos los controles están convenientemente etiquetados para que el usuario sepa qué función representa cada uno de ellos.
7. En la pantalla, el usuario puede ver en todo momento su pedido detallado, incluyendo la franquicia, el tipo de pizza y su precio, las unidades de cada ingrediente, su importe y el importe total del pedido.
8. Para modificar la cantidad de cada ingrediente, por cada ingrediente hay dos botones, uno con el signo más y otro con el signo menos que hacen aumentar o disminuir el número de unidades de cada ingrediente al ser pulsados en un rango de valores [0,+15].
9. En el área de confirmación de pedido, se ha provisto un campo de texto que informa al cliente de estado del pedido.

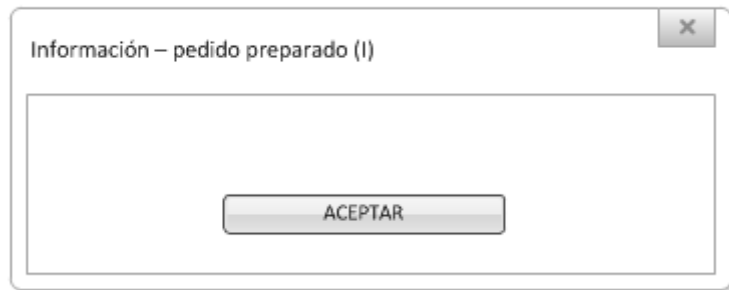
Diálogo de confirmación



Este diálogo permite que el usuario pueda detener el proceso de pedido y seguir modificando el pedido en curso .

Dispone de los botones sí, no y cancelar para dar uniformidad a todos los diálogos de confirmación posibles.

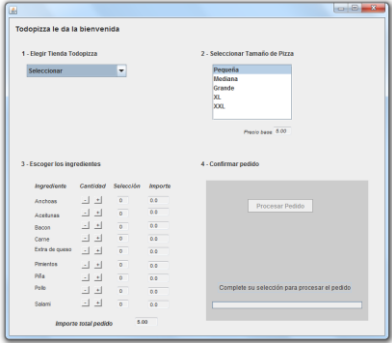
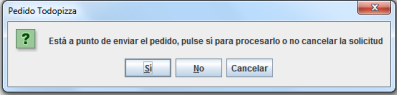
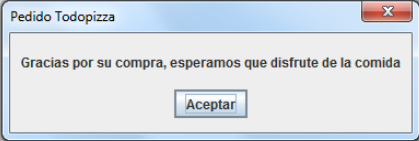
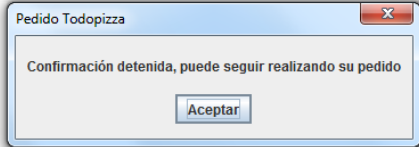
Diálogos Informativos.



Los diálogos informativos (hay dos) tendrán el formato de modales y un solo botón de aceptar.

A continuación veremos los tres niveles de formularios y diálogos de modo global y luego un detalle de cada uno de ellos.

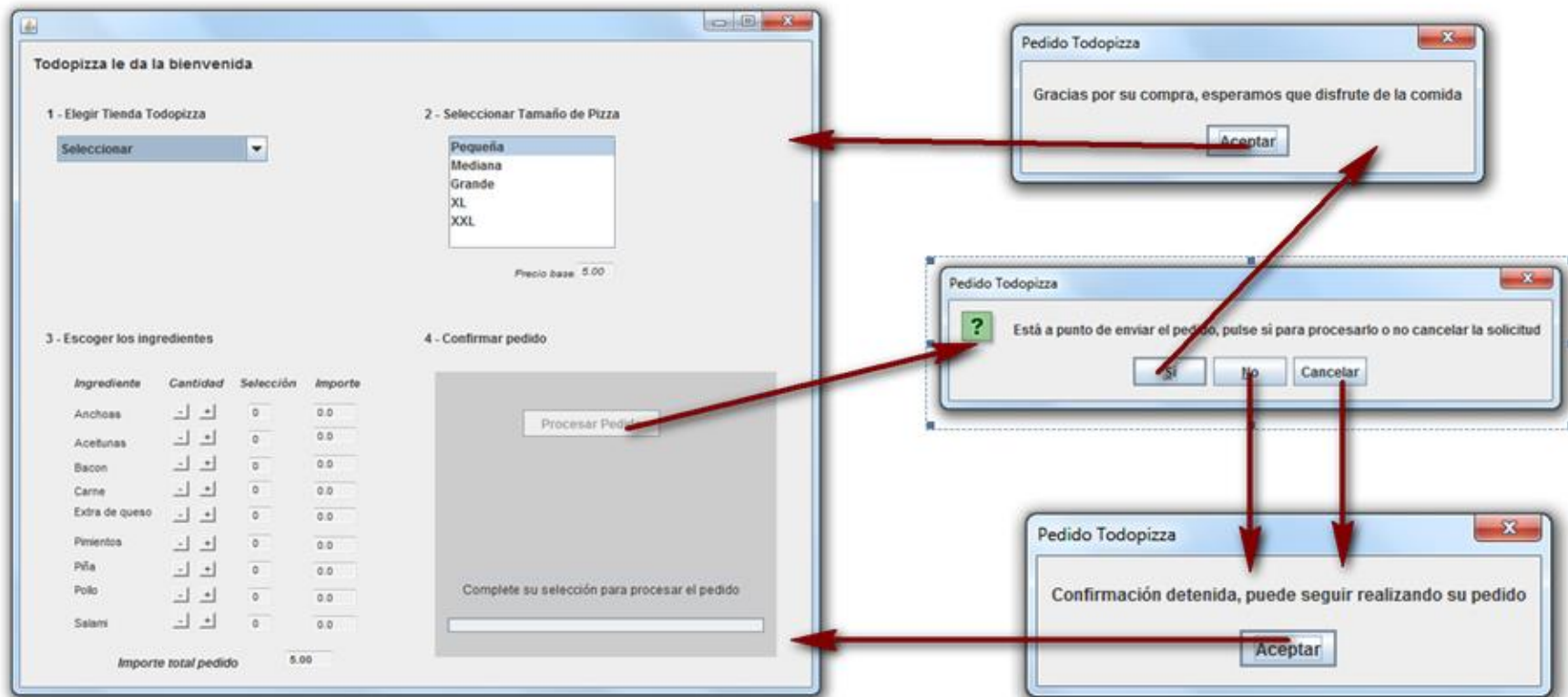
Mapa de formularios

Nivel 1	Nivel 2	Nivel 3			
	<p>Éste es el primer formulario que aparece cuando se abre la práctica. Se trata de un JFrame con todos los controles necesarios para realizar un pedido a Todopizza.</p> <p>Dispone de un solo botón para confirmar el pedido, de manera que sea fácil su utilización, además , de los típicos de control de los objetos tipo ventana (cerrar, minimizar y restaurar).</p> <p>Incorpora una zona informativa especial donde el usuario es notificado en tiempo real sobre la marcha del pedido y la parte en la que se encuentra el mismo.</p>				
Diálogo de Procesar Pedido		Al pulsar sobre el botón <i>Procesar Pedido</i> aparece este diálogo modal pidiendo la confirmación para procesarlo.			
	Diálogo Pedido Completado		Al pulsar en el botón sí del diálogo de procesar pedido, se inicia la preparación del mismo. Una vez se ha terminado, aparece este diálogo. Cuando se cierra el diálogo, se reinicia el formulario principal para que se pueda realizar otro pedido.		
	Diálogo Pedido Cancelado		Al pulsar en el botón no del diálogo de procesar pedido, se devuelve el control al formulario principal.		

El mapa queda de este modo:

Las flechas representan a qué objetos se accede pulsando los botones el IGU.

Tras aceptar el mensaje de confirmación de que el pedido está ya preparado, el formulario principal se reinicia, de manera que se puede volver a pedir de nuevo una pizza sin tener que reiniciar todo a mano.



La IGU en detalle.

Formulario principal.

Todopizza le da la bienvenida

1 - Elegir Tienda Todopizza

Seleccionar ▼

2 - Seleccionar Tamaño de Pizza

Pequeña
Mediana
Grande
XL
XXL

Precio base 5.00

3 - Escoger los ingredientes

Ingrediente	Cantidad	Selección	Importe
Anchoas	- +	0	0.0
Aceitunas	- +	0	0.0
Bacon	- +	0	0.0
Carne	- +	0	0.0
Extra de queso	- +	0	0.0
Pimientos	- +	0	0.0
Piña	- +	0	0.0
Pollo	- +	0	0.0
Salami	- +	0	0.0

Importe total pedido 5.00

4 - Confirmar pedido

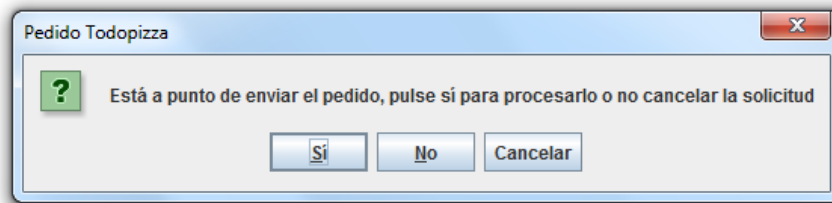
Procesar Pedido

Complete su selección para procesar el pedido

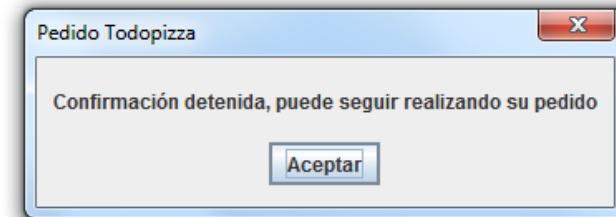
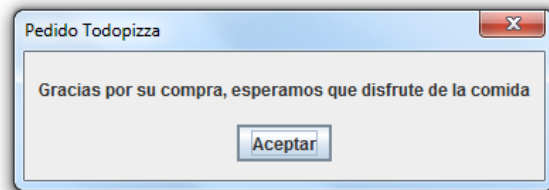
Para ayudar al usuario a completarlo, se divide en cuatro secciones las cuales están numeradas en el orden en que se han de ir completando. El formulario final añade algunas etiquetas, y por lo demás desarrolla la maqueta que se realizó al principio.

Hay dos tipos de diálogos, uno de confirmación y otros dos de información.

El diálogo de confirmación de pedido permite al usuario volver atrás y seguir modificando su pedido aunque hubiera pulsado el botón de procesar el pedido.



Los dos diálogos informativos informan al usuario de si el pedido se realizó o si el usuario canceló el mismo.



Una de las funcionalidades importantes es que el formulario está adscrito como observador al pedido, de manera que recibe todas las notificaciones de cambio de estado del mismo.

De esta manera, el formulario informa en tiempo real del estado del mismo al cliente y muestra una barra de progreso también en tiempo real.

Para mostrar el funcionamiento del mismo, he realizado cinco pedidos tal como se pide en el enunciado. En vez de mostrar cinco veces los procesos del pedido copio en cada proceso un pedido distinto, ya que la única modificación del formulario en tanto que es procesado se refiere al estado del pedido, no a la información del pedido, que durante todo el tiempo ya se muestra en pantalla.

En total, el proceso tiene más de cinco pasos, por lo que se pueden apreciar más de cinco pedidos incluso.

A continuación copio el proceso completo de pedido:

1. Está seleccionada la franquicia de Guadalajara, la pizza pequeña y una unidad de cada ingrediente, de manera que se puede comprobar con facilidad la corrección de la información respecto del enunciado. **El pedido está listo** para enviar y la señal es que **el botón de procesar se activa**.

The screenshot shows a web application window titled 'Todopizza le da la bienvenida'. It contains four main sections for creating a pizza order:

- 1 - Elegir Tienda Todopizza:** A dropdown menu with 'Guadalajara' selected.
- 2 - Seleccionar Tamaño de Pizza:** A list box with options: 'Pequeña' (selected), 'Mediana', 'Grande', 'XL', and 'XXL'. Below it, 'Precio base' is 5.00.
- 3 - Escoger los ingredientes:** A table with columns 'Ingrediente', 'Cantidad' (with '-' and '+' buttons), 'Selección' (with a '1' button), and 'Importe'.
- 4 - Confirmar pedido:** A large grey box containing a 'Procesar Pedido' button and the text 'Complete su selección para procesar el pedido' with an empty input field below it.

At the bottom left, 'Importe total pedido' is 10.5.

Ingrediente	Cantidad	Selección	Importe
Anchoas	- +	1	0.75
Aceitunas	- +	1	0.5
Bacon	- +	1	0.75
Carne	- +	1	1.0
Extra de queso	- +	1	0.5
Pimientos	- +	1	0.25
Piña	- +	1	0.25
Pollo	- +	1	1.0
Salami	- +	1	0.5

2. Un pedido distinto, pero el siguiente paso en el proceso. El programa le pide al usuario que su **confirmación para realizar el pedido**.

The screenshot shows a Java Swing window titled "Todopizza le da la bienvenida". It contains three main sections for ordering a pizza:

- 1 - Elegir Tienda Todopizza:** A dropdown menu showing "Madrid".
- 2 - Seleccionar Tamaño de Pizza:** A list box showing "Pequeña", "Mediana", "Grande", "XL", and "XXL".
- 3 - Escoger los ingredientes:** A table with columns "Ingrediente", "Cantidad", and "Selección". It lists ingredients like Anchoas, Aceitunas, Bacon, Carne, Extra de queso, Pimientos, Piña, Pollo, and Salami, each with a quantity of 1 and a price of 0.75.

At the bottom, there is a label "Importe total pedido" with a value of 6.75. A "Procesar Pedido" button is located in the bottom right corner.

A modal dialog box titled "Pedido Todopizza" is overlaid on the main window. It contains a green question mark icon and the text: "Está a punto de enviar el pedido, pulse sí para procesarlo o no cancelar la solicitud". Below the text are three buttons: "Sí", "No", and "Cancelar".

Ingrediente	Cantidad	Selección	Importe
Anchoas	- +	1	0.75
Aceitunas	- +	0	0.0
Bacon	- +	0	0.0
Carne	- +	0	0.0
Extra de queso	- +	1	0.5
Pimientos	- +	0	0.0
Piña	- +	0	0.0
Pollo	- +	0	0.0
Salami	- +	1	0.5

3. El usuario acepta la confirmación de pedido y el sistema comienza a informar de que el pedido está en curso y de la primera fase del mismo, que es **preparando ingredientes**. Además, el botón de procesar pedido se deshabilita para evitar que se pulse de nuevo mientras se procesa el pedido.

Todopizza le da la bienvenida

1 - Elegir Tienda Todopizza

Tarragona ▼

2 - Seleccionar Tamaño de Pizza

Pequeña
Mediana
Grande
XL
XXL

Precio base 10.0

3 - Escoger los ingredientes

Ingrediente	Cantidad	Selección	Importe
Anchoas	- +	2	1.5
Aceitunas	- +	2	1.0
Bacon	- +	2	1.5
Carne	- +	1	1.0
Extra de queso	- +	1	0.5
Pimientos	- +	1	0.25
Piña	- +	2	0.5
Pollo	- +	4	4.0
Salami	- +	1	0.5

Importe total pedido 20.75

4 - Confirmar pedido

Procesar Pedido

preparando los ingredientes

Progress bar: []

4. Horneando pizza...

Todopizza le da la bienvenida

1 - Elegir Tienda Todopizza

Madrid

2 - Seleccionar Tamaño de Pizza

Pequeña
Mediana
Grande
XL
XXL

Precio base 25.0

3 - Escoger los ingredientes

Ingrediente	Cantidad	Selección	Importe
Anchoas	- +	0	0.0
Aceitunas	- +	3	1.5
Bacon	- +	2	1.5
Carne	- +	2	2.0
Extra de queso	- +	3	1.5
Pimientos	- +	3	0.75
Piña	- +	0	0.0
Pollo	- +	4	4.0
Salami	- +	6	3.0

Importe total pedido 39.25

4 - Confirmar pedido

Procesar Pedido

horneando pizza

5. Cortando pizza en porciones...

Todopizza le da la bienvenida

1 - Elegir Tienda Todopizza

Bilbao

2 - Seleccionar Tamaño de Pizza

Pequeña

Mediana

Grande

XL

XXL

Precio base 15.0

3 - Escoger los ingredientes

Ingrediente	Cantidad	Selección	Importe
Anchoas	- +	0	0.0
Aceitunas	- +	0	0.0
Bacon	- +	1	0.75
Carne	- +	0	0.0
Extra de queso	- +	0	0.0
Pimientos	- +	0	0.0
Piña	- +	0	0.0
Pollo	- +	0	0.0
Salami	- +	0	0.0

Importe total pedido 15.75

4 - Confirmar pedido

Procesar Pedido

cortando pizza en porciones

6. Empaquetando pizza...

Todopizza le da la bienvenida

1 - Elegir Tienda Todopizza

Guadalajara

2 - Seleccionar Tamaño de Pizza

Pequeña
Mediana
Grande
XL
XXL

Precio base 10.0

3 - Escoger los ingredientes

Ingrediente	Cantidad	Selección	Importe
Anchoas	- +	0	0.0
Aceitunas	- +	1	0.5
Bacon	- +	0	0.0
Carne	- +	0	0.0
Extra de queso	- +	0	0.0
Pimientos	- +	0	0.0
Piña	- +	0	0.0
Pollo	- +	0	0.0
Salami	- +	0	0.0

Importe total pedido 10.5

4 - Confirmar pedido

Procesar Pedido

empaquetando la pizza

7. Por último, la el sistema informa de que la pizza **pizza está preparada** y agradece con un diálogo la compra. Al pulsar el botón de aceptar, el formulario se reinicia automáticamente, listo para un nuevo pedido.

The screenshot shows the 'Todopizza' application window. The title bar says 'Todopizza le da la bienvenida'. The interface is divided into three main sections:

- 1 - Elegir Tienda Todopizza:** A dropdown menu showing 'Barcelona'.
- 2 - Seleccionar Tamaño de Pizza:** A list box with options: 'Pequeña', 'Mediana', 'Grande', 'XL' (selected), and 'XXL'. Below it, 'Precio base' is 20.0.
- 3 - Escoger los ingredientes:** A table with ingredients and their quantities/prices.

A modal dialog box titled 'Pedido Todopizza' is overlaid on the interface, displaying the message: 'Gracias por su compra, esperamos que disfrute de la comida'. It has an 'Aceptar' button.

Below the ingredients table, the 'Importe total pedido' is 22.25.

On the right side of the interface, there is a large grey area with the text 'pizza preparada' and a blue progress bar at the bottom.

Ingrediente	Cantidad	Precio
Anchoas	- +	
Aceitunas	- +	
Bacon	- +	0 0.0
Carne	- +	0 0.0
Extra de queso	- +	0 0.0
Pimientos	- +	0 0.0
Piña	- +	1 0.25
Pollo	- +	1 1.0
Salami	- +	0 0.0