Breve manual de diseño de la práctica del Kiosco de Pedidos de una Hamburguesería

Tabla de contenidos

Versión 1.0.49

18:58 - 09/12/24

1 Introducción

El siguiente documento pretende guiar al estudiante por un diseño adecuado para implementar la práctica del Kiosco de Pedidos de una Hamburguesería (KPH).

El diagrama de actividad de la Figura 1 muestra a grandes rasgos la lógica asociada a la aplicación KPH. Obsérvese que este diagrama no contempla la lógica de bajo nivel asociada a cada operación. Este diagrama tampoco refleja la estructura de clases del programa.

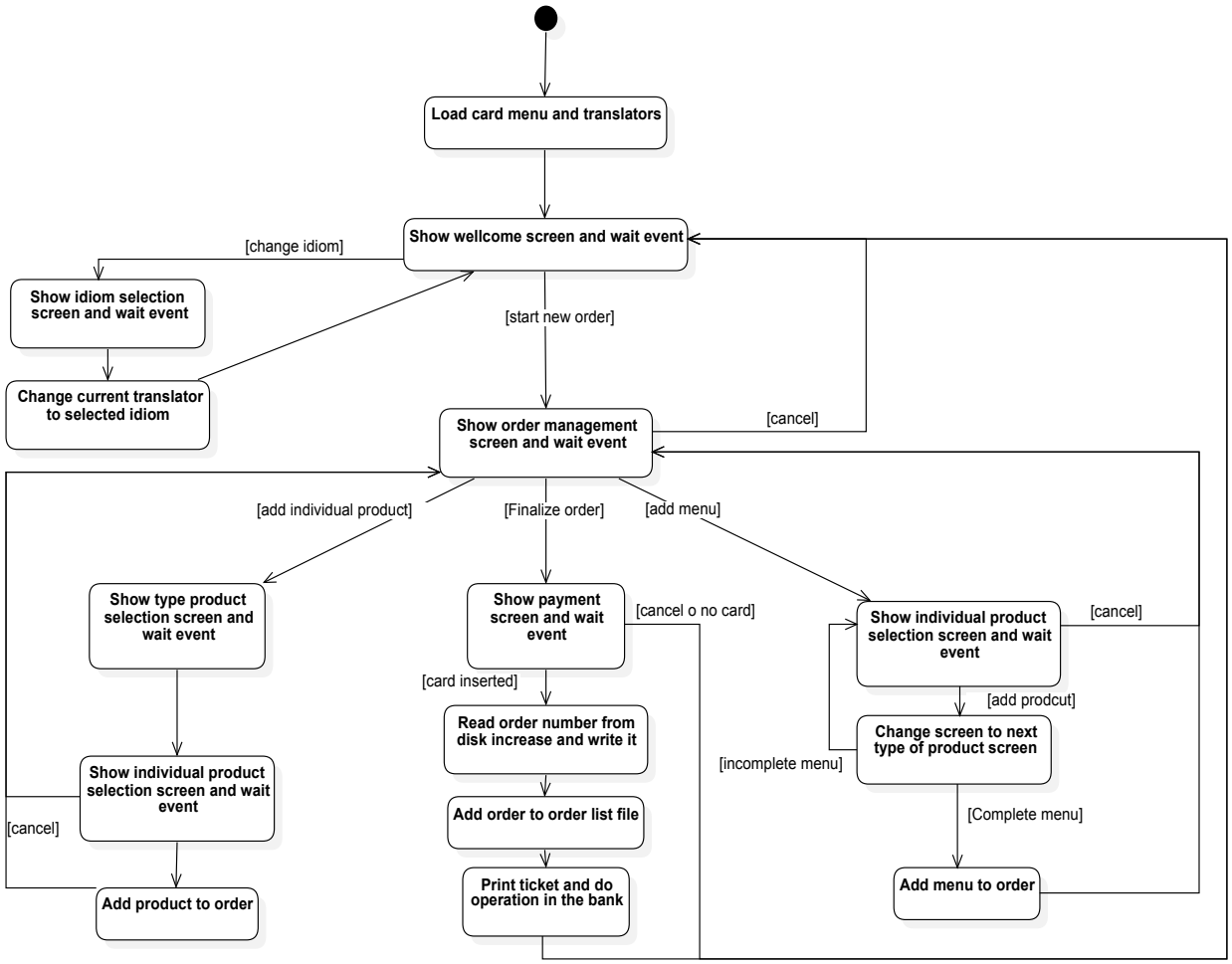


Figura 1.- Diagrama de actividad con la lógica de alto nivel asociada a la práctica.

2 Clases propuestas

La Figura 2 presenta las clases principales que gestionan la operativa. A continuación se resume la responsabilidad de cada clase de este diagrama.

- `BurgerSelfOrderKiosk`.- Encargada de iniciar el programa.
- `KioskManager`.- Encargada de iniciar el algoritmo de gestion de pedidos (ver Figura 1).
- `KioskScreen`.- Representa el modelo de una pantalla. Cada modelo de pantalla estará encargado de realizar las diferentes operaciones que realiza el programa en una pantalla. Todas las clases que derivarán de

`KioskScreen` comparten el método `show`. Este método recibe el contexto y devuelve la siguiente pantalla que deberá mostrarse de acuerdo con el resultado de las acciones del usuario sobre esta pantalla.

- `MenuCard`.- Encargada de almacenar y gestionar la carta del restaurante. Esta clase tiene un método estático que permite deserializar la carta del restaurante que se proporciona en el fichero catalog.xml que está dentro del zip PRODUCTOS.
- `SimpleKiosk`.- Encargada de simplificar la interacción con la clase `SimpleOrderKioskSimulator`. Por ejemplo, para traducir automáticamente las cadenas que se le pasen o la de borrar el contenido de las pantallas.
- `Context`.- Encargada de agrupar todo el contexto necesario para realizar las diferentes operaciones: el traductor, la carta, el pedido y el kiosko.
- `Translator`.- Encargada de traducir las cadenas del código desde el idioma en el que estén escritas a un idioma determinado. Así, habrá un objeto Translator para traducir al inglés, otro para el catalán...
- `TranslatorManager`.- Encargada de gestionar los diferentes `Translator` existentes. Cuando el `TranslatorManager` se inicia carga los diferentes `Translator`. La propiedad `currentTranslator` apunta al `Translator` activo. Si `currentTranslator` es `null` las cadenas no se traducen, en otro caso se usa el `Translator` apuntado para traducir la cadena.

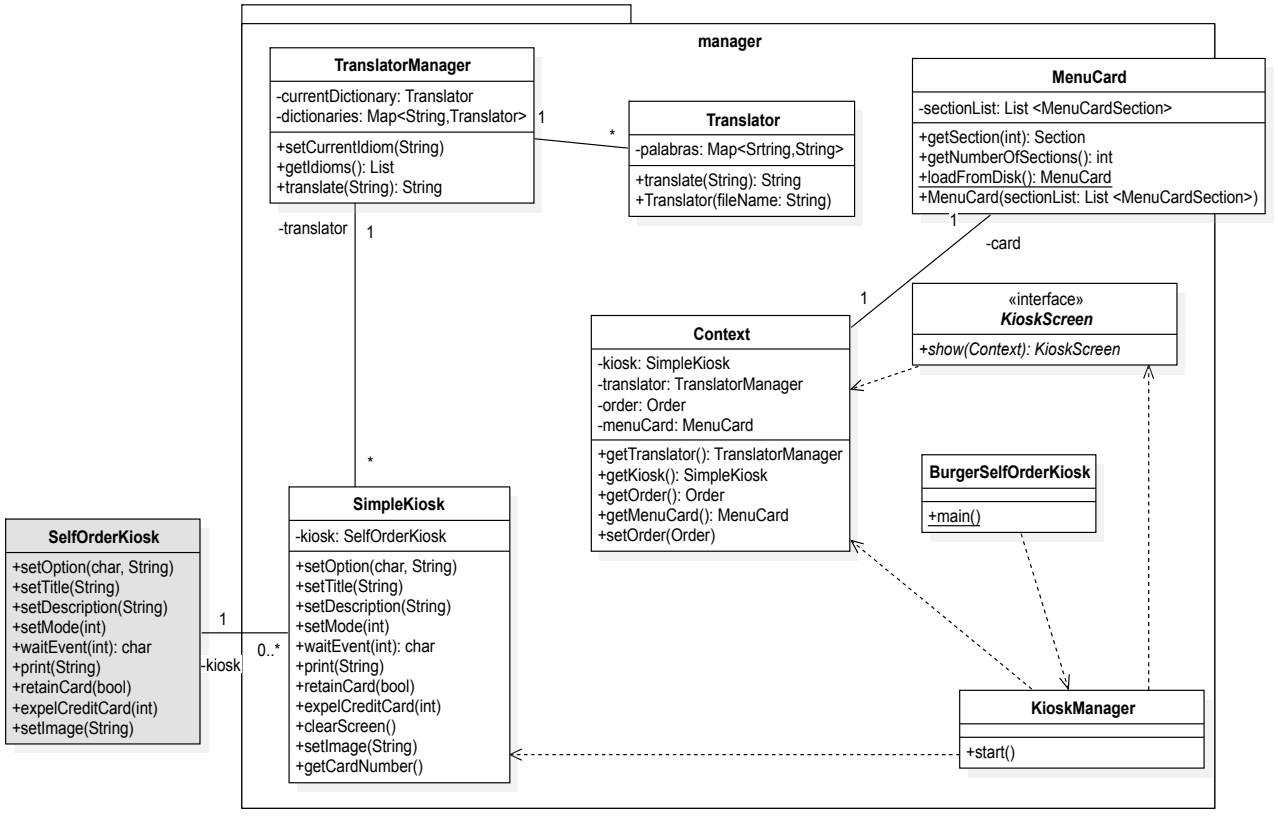


Figura 2.- Diagrama de clases principal.

La Figura 3 presenta las clases que derivan de `KioskScreen` y que se encargan de modelar las diferentes operaciones. A continuación se resume la responsabilidad de cada clase de esta figura.

- `WellcomeScreen`.- Gestiona la pantalla que ve el usuario al inicio de su interacción. Dicha pantalla mostrará dos botones: cambiar idioma o iniciar pedido.

- `OrderScreen`.- Modela la pantalla de gestión del pedido. Los botones de esa pantalla permitirán: añadir un producto individual al pedido, añadir un menú, revisar el pedido, finalizar el pedido o cancelar el pedido.
- `PurcheaseScreen`.- Modela la pantalla de pago, realizando la operativa de presentar un resumen del pedido, solicitar la tarjeta de crédito al usuario y realizar el pago usando la clase de comunicación con el banco.
- `CarouselScreen`.- Modela la parte común de las diferentes pantallas que necesitan un carrusel de elementos. Estas son: elegir entre los diferentes secciones de la carta, entre los diferentes productos de un tipo o la revisión de los productos comprados en un pedido.
- `CardSectionSelectionScreen`.- Modela la pantalla de selección de las diferentes secciones de la carta: bebidas, complementos, platos principales...
- `ProductSelectionScreen`.- Modela la pantalla de selección de productos de una sección de la carta: hamburguesa de pollo, hamburguesa de ternera, burrito…
- `MenuSelectionScreen`.- Modela la pantalla de selección de productos de un menú. Esta pantalla, en realidad incluye varias pantallas de tipo carrusel, ya que presentará tantas pantallas de selección como secciones tenga la carta.
- `OrderRevisionScreenScreen`.- Modela la pantalla de revisión de los productos de en un pedido para poder eliminarlos. En el caso de los menús, se eliminará el menú completo, no pudiéndose eliminar partes de un menú.

Finalmente, la Figura 4 presenta las clases relacionadas con la información de la carta del restaurante.

- `Product`.- Modela la interfaz común a los dos tipos productos: productos individuales y menús.
- `InidivudalProduct`.- Modela los productos individuales de la carta del restaurante.
- `MenuCardSection`.- Cada objeto de esta clase contiene los `InidivudalProduct` de una sección del menú. En principio habrá secciones para bebidas, principales y complementos.
- `Menu`.- Modela los menús que se compondrán de un elemento de cada una de las secciones. Esta clase, al iniciarse por primera vez carga de un fichero de configuración el valor de porcentaje de descuento que se aplica a los `InidivudalProduct` que lo compongan.

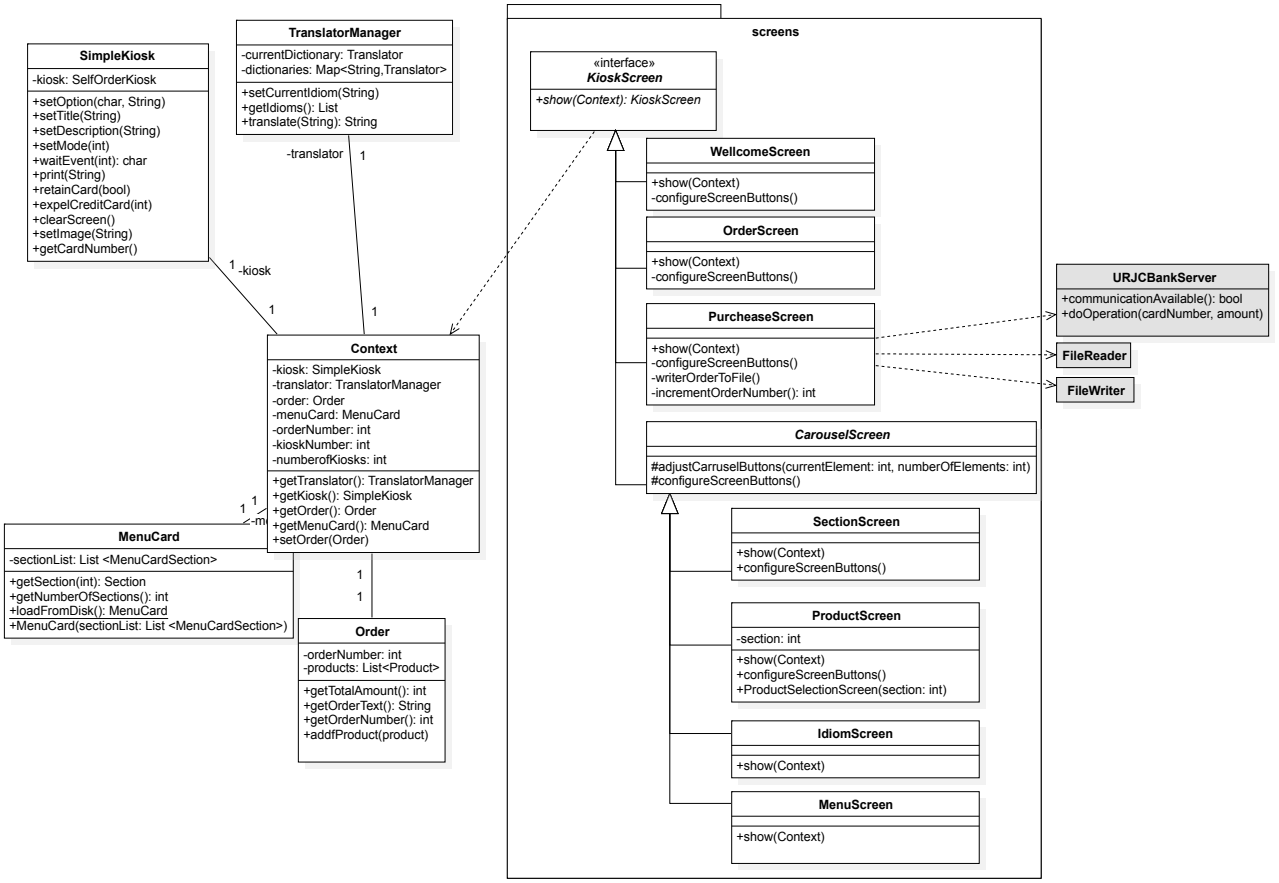


Figura 3.- Diagrama de clases que refleja los modelos utilizados por las diferentes pantallas con las que interaccionará el usuario para realizar un pedido.

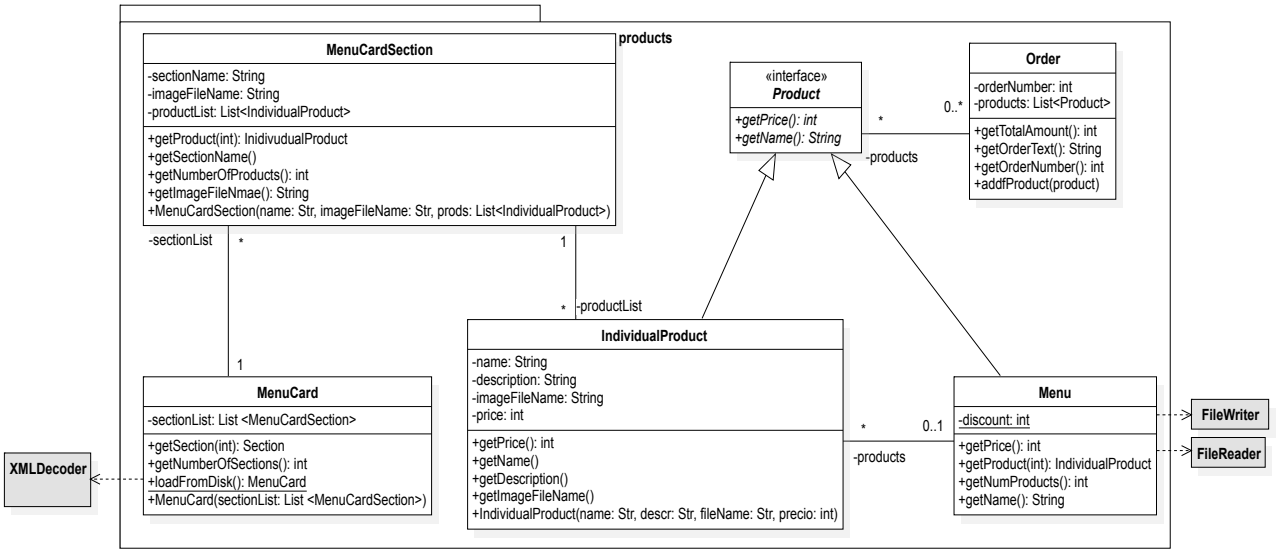


Figura 4.- Diagrama de clases para los productos de la carta y los objetos en donde aparecen.

3 Operativa general del kiosco

Partiendo de estas clases, el Diagrama de Actividad de la Figura 1 quedaría segmentado en las clases y métodos que se ven el diagrama de la Figura 5.

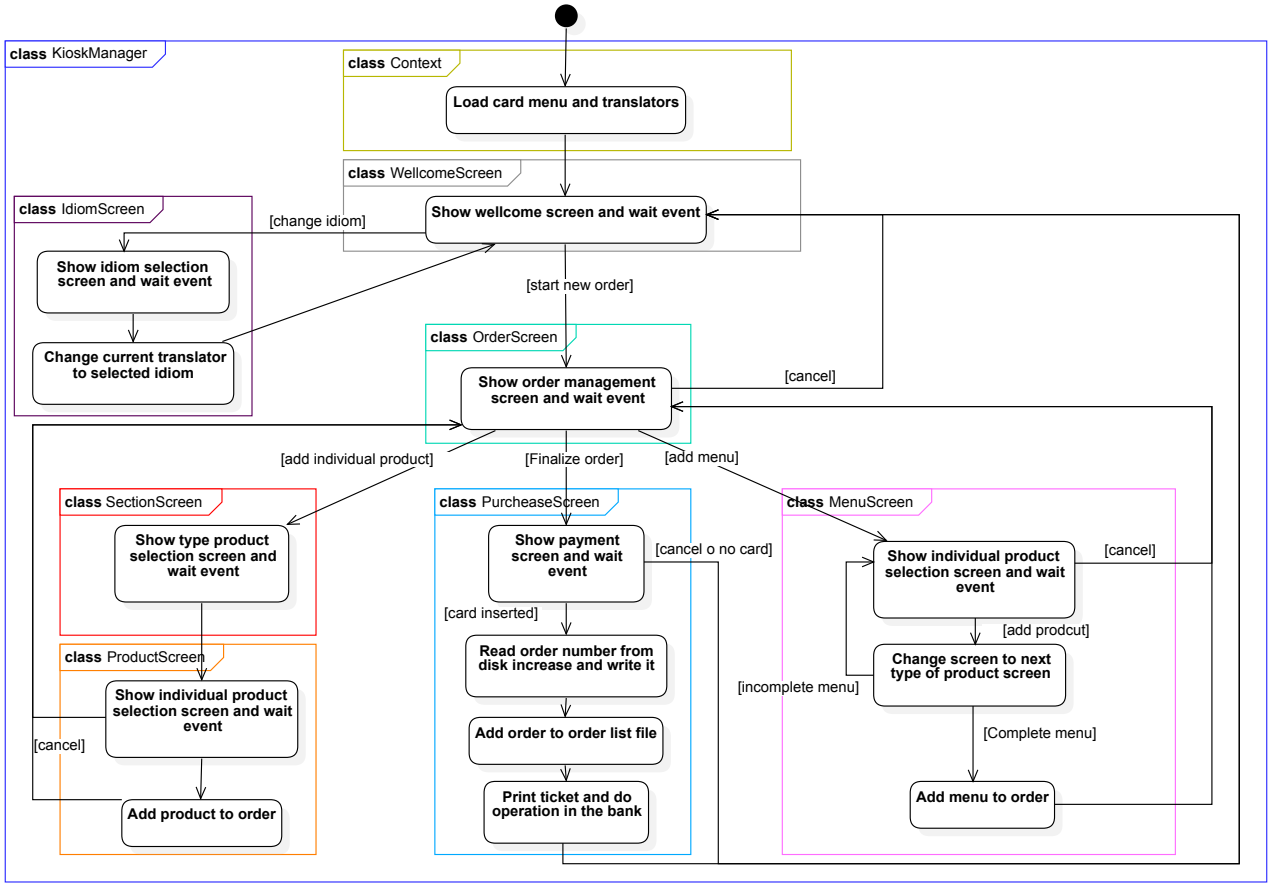


Figura 5.- Diagrama de actividad con la lógica de alto nivel asociada a la práctica repartida en las clases que lo implementarían.

3.1 Inicio de la aplicación y bucle principal

El diagrama de secuencia de la Figura 6 ilustra el inicio de la aplicación.

En el diagrama se ve que al inicio de la aplicación se crea el `KioskManager` que a su vez crea un objeto `Context`. Éste carga los diferentes diccionarios para la traducción de los mensajes que parten de la aplicación y también carga la carta. Luego se crea el objeto que gestiona la pantalla de bienvenida y la sitúa como pantalla actual.

Finalmente, entra en un bucle que muestra la pantalla actual. Cuando finaliza la ejecución de la pantalla actual, dicha pantalla devolverá la siguiente pantalla que deberá calificarse como pantalla actual.

Así, el orden de las pantallas viene determinado por las propias pantallas. Si se quisiese añadir una pantalla hay que añadirlo solo en la pantalla que le de paso.

La Figura 6 se complementa con el diagrama de secuencia de la Figura 7, que refleja la secuencia de pantallas que se produciría en el caso de un cliente que solo comprase un producto individual.

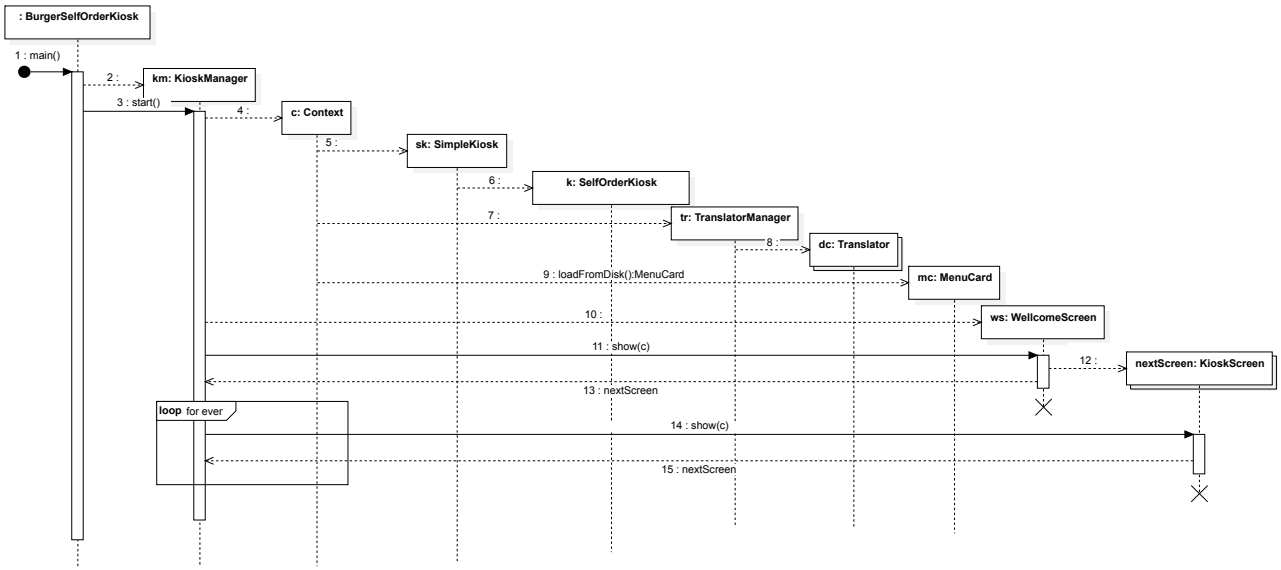


Figura 6.- Diagrama de secuencia que refleja el inicio del programa.

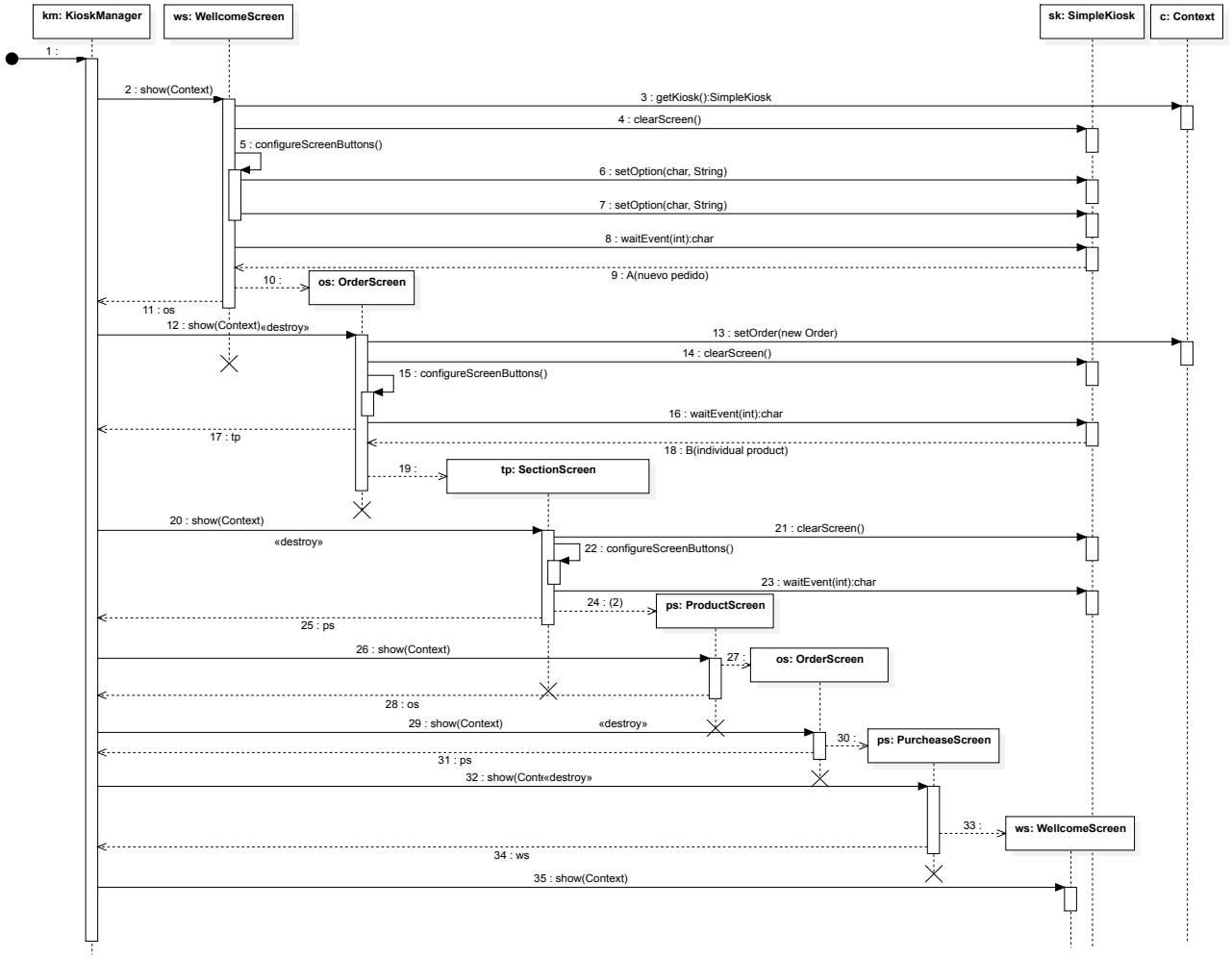


Figura 7.- Diagrama de secuencia que refleja la secuencia de pantallas que aparecen cuando un cliente compra solo un producto individual.

3.2 Operativa del kiosco simple y del traductor

El diagrama de la Figura 8 muestra el funcionamiento del sistema de traducción junto con la simplificación introducida por el cliente simple.

Por un lado, el `SimpleKiosk` se encarga de añadir métodos de utilidad (como `clearScreen`) que no tiene `SelfOrderKioskSimulator` y que son útiles para los usuarios de esta clase. Además, independiza el programa de la interfaz de usuario de Sienens, de manera que si en el futuro se cambia por otro el resto del programa no cambie.

Por otro lado, `SimpleKiosk` hace de intermediario (Proxi) para todos los métodos de `SelfOrderKioskSimulator`. Aunque, mejora algunos de ellos. Por ejemplo, `SimpleKiosk` intenta traducir todas las cadenas que pasan por sus manos camino de `SelfOrderKioskSimulator`. Obsérvese que, previamente se ha inicializado el `TranslatorManager`, el cual habrá cargado todos los traductores de los diferentes idiomas, y ya estará seleccionado un `currentTranslator`.

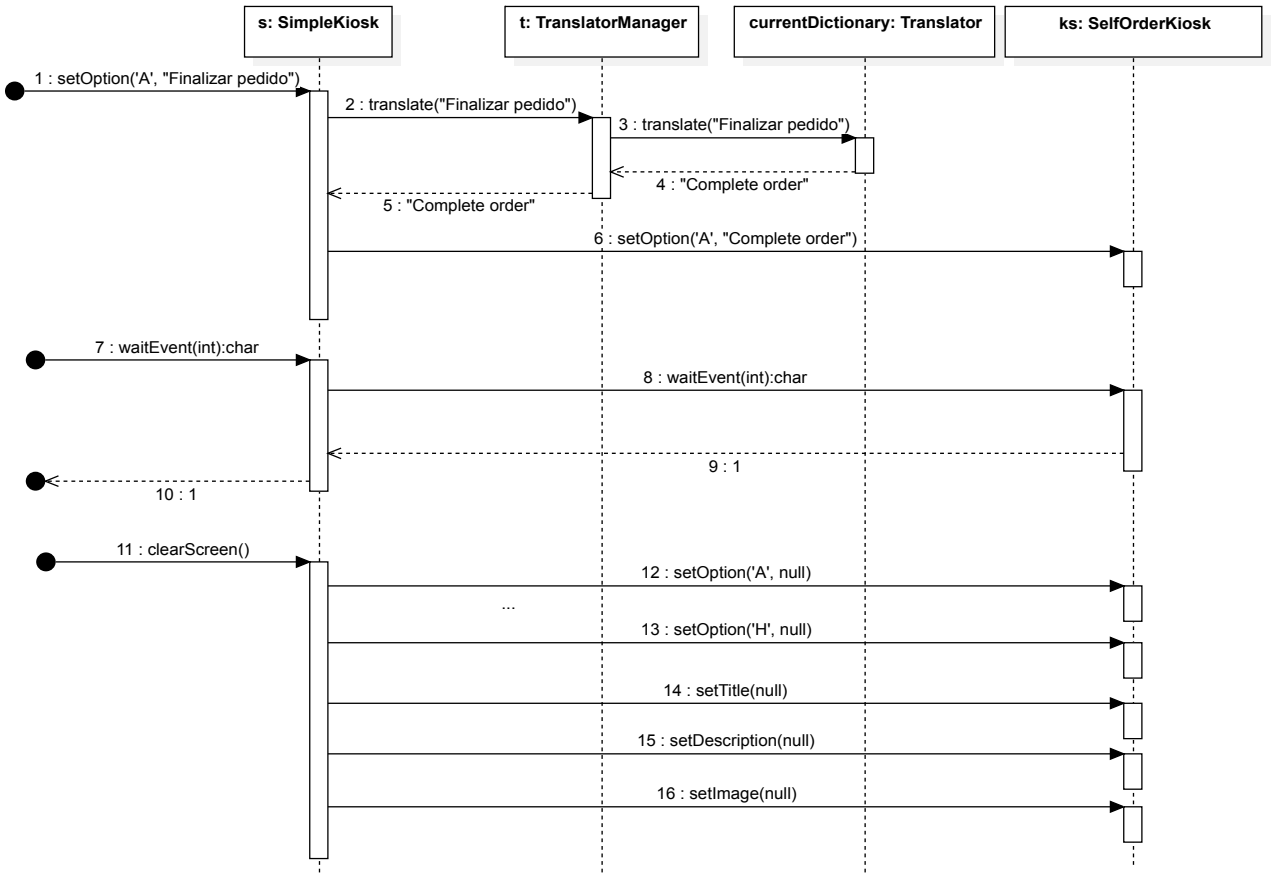


Figura 8.- Diagrama de secuencia que refleja el uso de SimpleKiosk y la traducción de una frase que insertada en un botón del kiosco.

Para que se pueda realizar la traducción a otro idioma de las frases que tiene el código, por cada idioma deberá existir un fichero y un objeto `Translator`. Cada uno de estos ficheros constará de todas las frases que pueden aparecer en el programa en español y en el otro idioma. Por ejemplo, la Figura 9 presenta el fichero de traducción de cadenas a inglés.

"Cancelar" " -	"Cancel"
…	…
"El precio del pedido es # euros"	"The order price is # euros"
"Finalizar pedido"	"Complete order"

Figura 9.- Ejemplo de traducción de frases del programa de español a inglés.

3.3 Operativa de carrusel

Los diagramas de las Figuras 11 y 10 ilustran las operaciones que ocurren dentro de las pantalla del carrusel cuando se les pide que muestren una lista de elementos.

En general, el carrusel es una pantalla que se debe usar cuando el número de elementos a mostrar excede el número de botones del kiosco.

Respecto a la implementación se puede decir que todas las pantalla de tipo carrusel derivan de la clase `CarouselScreen`. En esta clase padre se podrán poner, además de los métodos que se indican, todos los métodos comunes que se considere oportunos para que las clases que hereden los usen.

La Figura 10 es un diagrama de actividad que muestra el uso de los diferentes botones dentro del carrusel cuando se está seleccionando un producto individual.

La Figura 11 es es un diagrama de secuencia que muestra la selección de un producto individual usando un a pantalla de tipo carrusel. Ese producto está dentro de una sección de la carta del restaurante, y se supone que ya se ha elegido dicha sección.

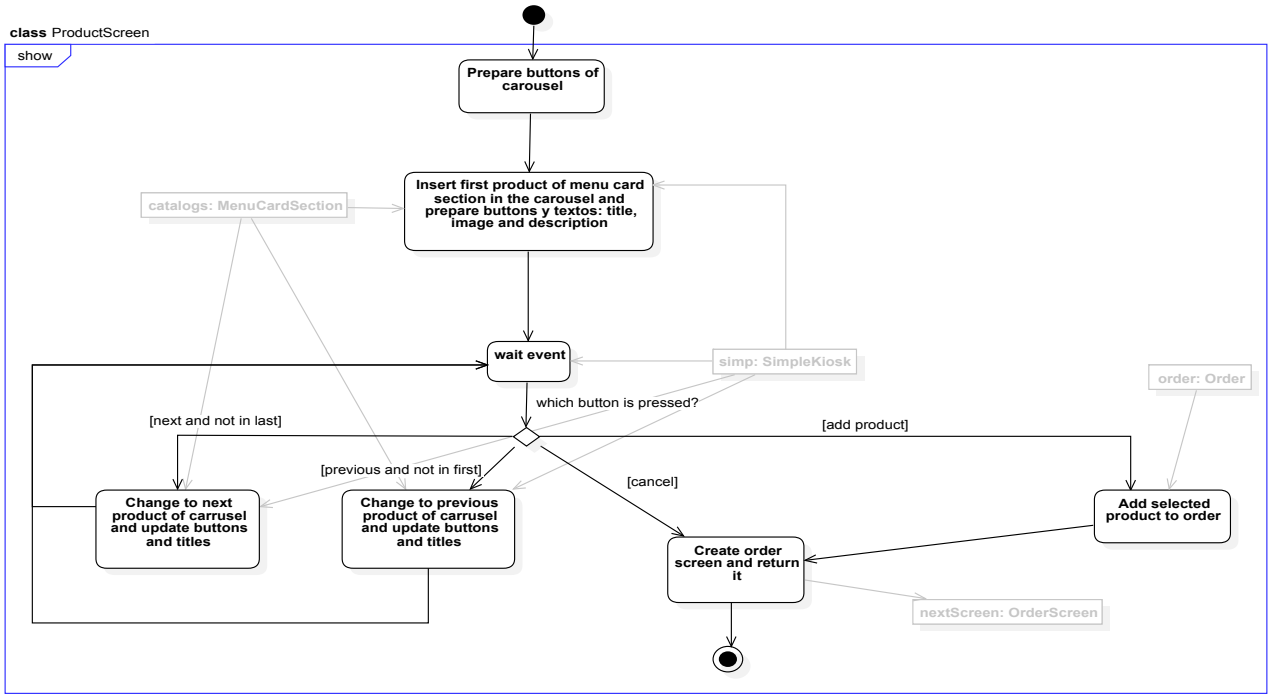


Figura 10.- Gestión de los eventos dentro de la pantalla de tipo carrusel de selección de un producto individual.

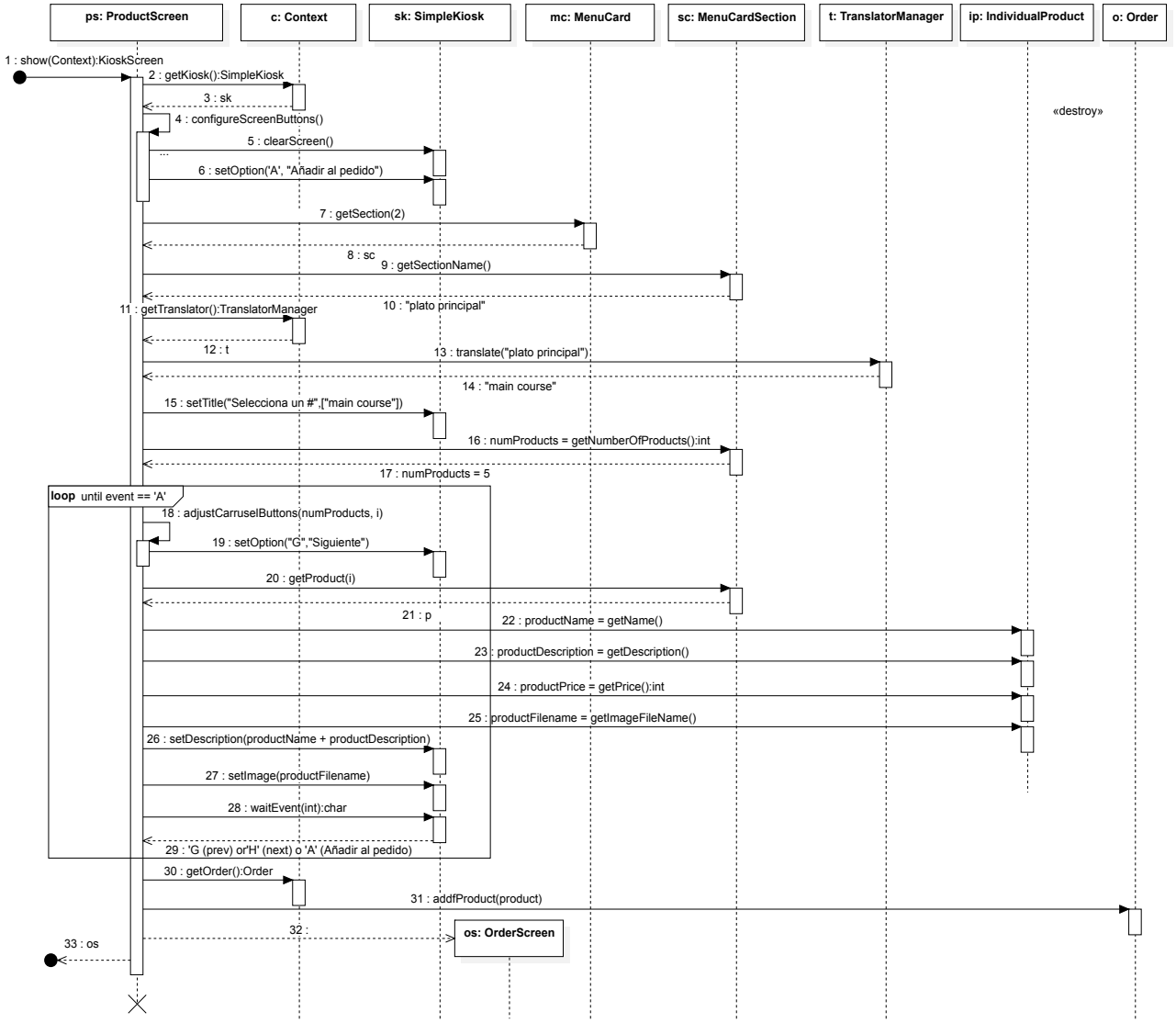


Figura 11.- Pantalla de tipo carrusel para la selección de producto individual.

3.4 Operativa de la pantalla de pago

La Figura 12 muestra la operativa que se desarrolla cuando se confirma el pago de un pedido.

La Figura 13 muestra el detalle de los métodos de incrementar el número del pedido y la escritura del fichero del pedido.

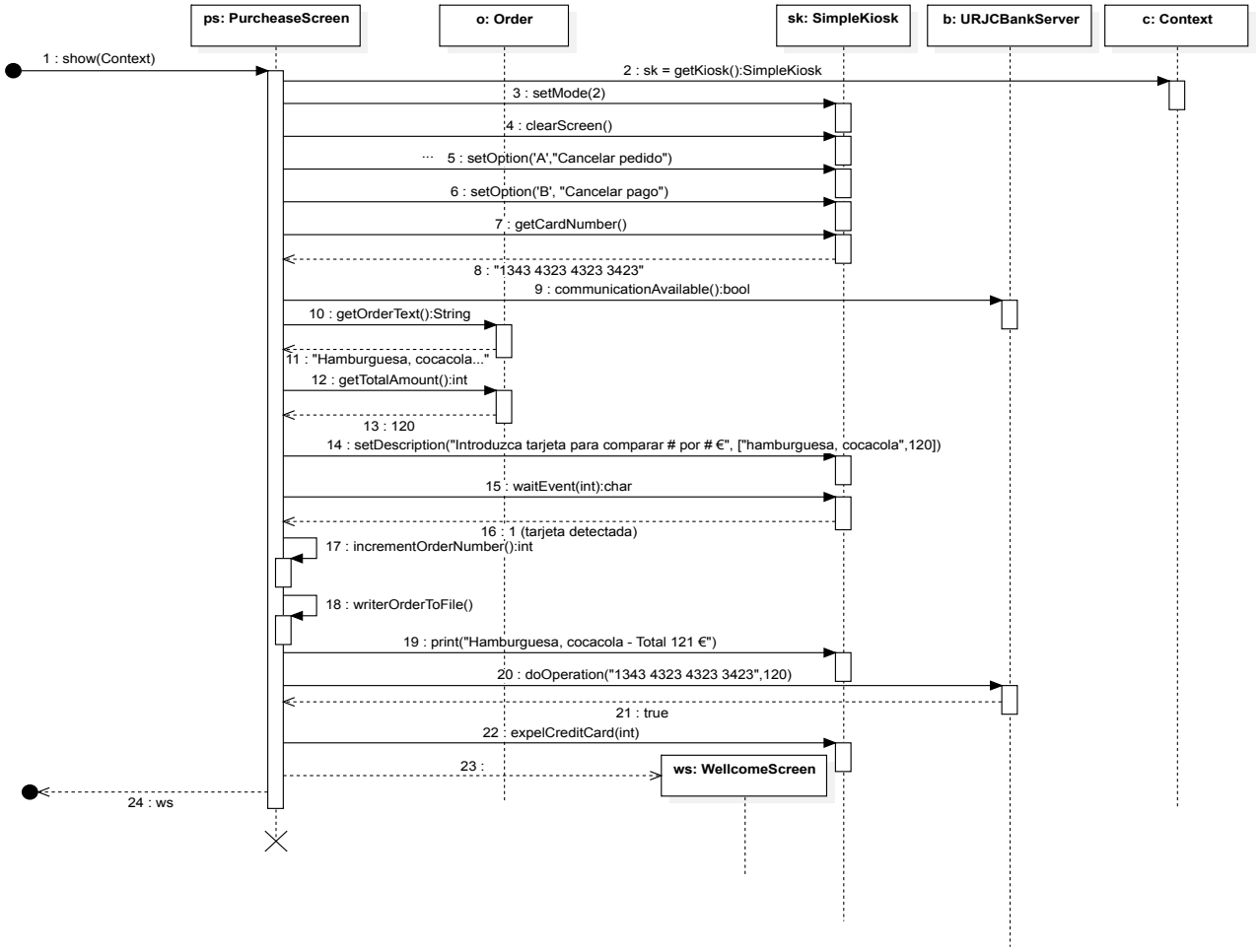


Figura 12.- Pantalla que se muestra al pulsar en fin del pedido.

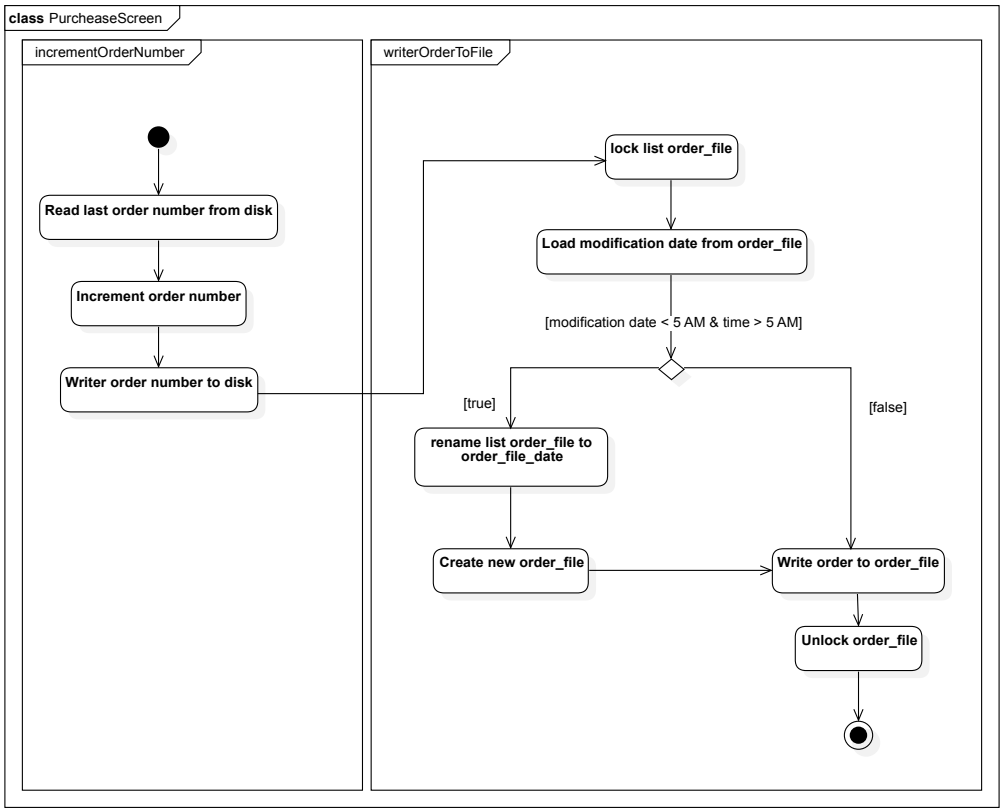


Figura 13.- Algoritmo de incremento del número de pedido y escritura del pedido al fichero de pedidos.

4 Conclusiones

Este documento presenta un diseño orientativo de la práctica del Kiosco de Pedidos de una Hamburguesería. En particular, se han presentado 11 diagramas con numerosas explicaciones que cubren los siguientes requisitos del documento de ERS. La tabla de la figura muestra junto al título de cada requisito, el número de aquellos diagramas que tienen alguna relación.

Se puede observar que el requisito de pedir confirmación a la pulsación de cancelar (R6) no se considera en este diseño. Tampoco se considera la opción de anular elementos de un pedido (R21). Por ello, estos dos requisitos (R6 y R21) no hay que implementarlos.

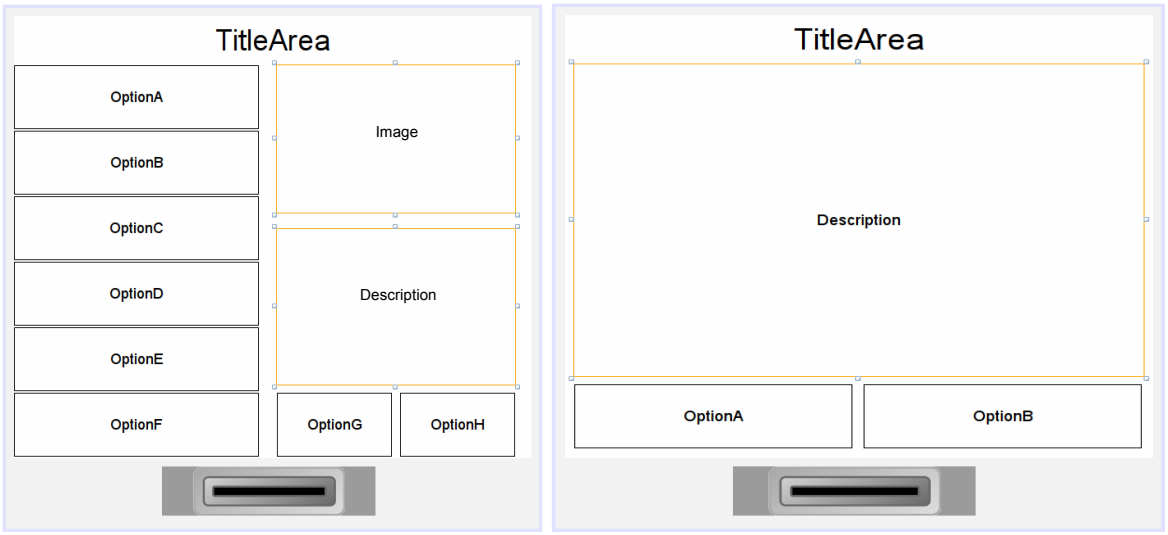
A pesar de todas estas descripciones y diseños, para obtener una aplicación funcionando, hay muchos aspectos necesarios que quedan fuera del alcance de este documento. Dichos aspectos los debe resolver el estudiante en la implementación. Estos aspectos pueden incluir la creación de nuevos métodos e incluso clases, aunque se deben seguir las líneas maestras marcadas en este documento.

Requisito	Diagramas que lo cubren
R1 – Leer productos	1, 2, 4, 5, 6
R2 – Elegir idioma	1, 5
R3 – Usar idioma	2, 9, 8
R4 – Botón nuevo pedido	1, 5, 7
R5 – Botones de cancelar pedido	1, 5
R6 – Pantallas de confirmar cancelación	
R7 – Pantalla de gestión de pedido	1, 5, 7
R8 – Al menos tres tipos de productos	1, 4, 5
R9 – Pantalla de producto individual	1, 5, 7
R10 - Carrusel	10, 11
R11 – Botón de fin de pedido	1, 12
R12 – Fotografía de producto	4, 8, 11
R13 – Descripción de producto	3, 4, 11
R14 – Botón cancelar productos	1, 5, 10
R15 – Botón cancelar tipo producto	1, 5
R16 – Descuento modificable	3, 4
R17 – Resumen del pedido	1, 4, 5, 12
R18 – Pedir tarjeta para pagar	1, 5, 12
R19 – Conectar con el banco	1, 3, 5, 12
R20 – Imprimir tique	1, 3, 12
R21 – Eliminar producto del pedido	
R22 – Siguiente y anterior en carrusel	10, 11
R23 – Leer número último pedido	1, 3, 5, 12, 13
R24 – Mostrar número en tique	4, 12
R25 – Escribir en fichero para cocina	1, 3, 5, 12, 13

Figura 14.- Tabla que muestra la relación de los requisitos con las diferentes figuras.

Anexo A API del simulador del kiosko

The dispenser has the Graphical User Interface (GUI) with two modes presented in Figura 15.



(a) (b)

Figura 15.- The two modes of the cinema ticket dispenser: (a) option screen and (b) message screen.

This dispenser can be controlled with the `SelfOrderKioskSimulator` class. In the following paragraphs the methods of `SelfOrderKioskSimulator` class are described.

```
public SelfOrderKiosk()
```

`SelfOrderKioskSimulator` class can build using the following constructor.

```
public void setOption(char optionNumber, java.lang.String message)
```

Set the message associated to the selected option button.

The screen shows buttons. Usually, buttons from A to F are use for options, while buttons G and H are used for navigating (next or previous) between the showed poducts in the image an description areas.

Parameters:

`optionNumber` - from 'A' to 'H'.
`message` - to be set in the screen button.

```
public void setTitle(java.lang.String message)
```

Set the message associated with the title to guide the user at the performed operation.

Parameters:

`message` - to be set in the title.

```
public void setDescriptio(java.lang.String message)
```

Set the message associated with the description text.

Parameters:
`message` - to be set in the description area.

```
public void setMode(int mode)
```

Set the dispenser mode: option screen or message screen.

Parameters:
`mode` – 0 for option screen or 1 for message screen.

```
public void setImage(java.lang.String imageFileName)
```

Set the image associated with the title to guide the user in the performed operation.

Parameters:
`message` - to be set in the title.

```
public char waitEvent(int seconds)
```

Wait until an event occur in the dispenser. This operation blocks the ejecution flow until an event is detected or until the time is over.

Parameters:
`seconds` - Time waiting for an event
Returns:
0: No event detected in the dispenser (as integer).
1: Credit card detected (as integer).
'A' to 'F' : pressed option (as character).

```
public boolean print(java.util.List<java.lang.String> text)
```

Print a ticket with the text provided.

Parameters:
`text` - to be printed.
Returns:
true if the operation is performed.

```
public void retainCreditCard(boolean definitely)
```

Retain the credit card in the dispenser.

Parameters:

`definitely` - if true the card is retained for ever and can not be expelled. If false the card can be expelled calling expelCreditCard().

```
public boolean expelCreditCard(int seconds)
```

Expel the credit card during a while. If the credit card is not retired it is retained again.

Parameters:

`seconds` – Seconds expelling the credit card. This operation blocks the ejecution flow until the card is removed or until the time is over.

Returns:

true if the credit card is retired by the customer or false if it is not retired.

```
public long getCardNumber()
```

Returns:

the card number of the inserted credit card or 0 if there aren't a card.

Anexo B Urjc Bank package

```
public UrjcBankServer()
```

UrjcBankServer class can build using the following constructor.

```
public boolean doOperation(long cardNumber , int amount)
```

Try to perform the payment with the amount passed by parameter.

Parameters:

`cardNumber` - of the customer credit card.

`amount` - in cents.

Returns:

True if the operation is performed or false if there aren't enough money in the acount.

Throws:

`java.lang.RuntimeException` - if the communication with the server is interrupted.

```
public boolean comunicationAvaiable()
```

Returns:

true if the communication with the bank server is available or false in other case.