

ACTIVIDAD 6.7

Realiza en WhiteStarUML el diagrama de casos de uso para el sistema de venta y el de gestión de notas.

6.4. DIAGRAMAS DE INTERACCIÓN

El diagrama de interacción describe el intercambio de secuencias de mensajes entre las partes de un sistema. Proporciona una visión integral del comportamiento de un sistema, es decir, muestra el flujo de control a través de varios objetos.

Dentro de la categoría de diagramas de interacción tenemos dos diagramas que se centran en aspectos distintos: el diagrama de secuencia (centrado en la ordenación temporal de los mensajes) y el diagrama de comunicación o colaboración (centrado en la organización estructural de los objetos que envían y reciben mensajes).

6.4.1. Diagrama de secuencia

El **diagrama de secuencia de sistema** muestra gráficamente los eventos que fluyen de los actores al sistema. Para su elaboración se parte de los casos de uso elaborados durante la etapa de análisis. Un diagrama de secuencia tiene dos dimensiones: la dimensión vertical que representa el tiempo y la dimensión horizontal que representa los roles que participan en la interacción. La Figura 6.30 muestra el diagrama de secuencia de sistema para el caso de uso *BuscarProductos*.

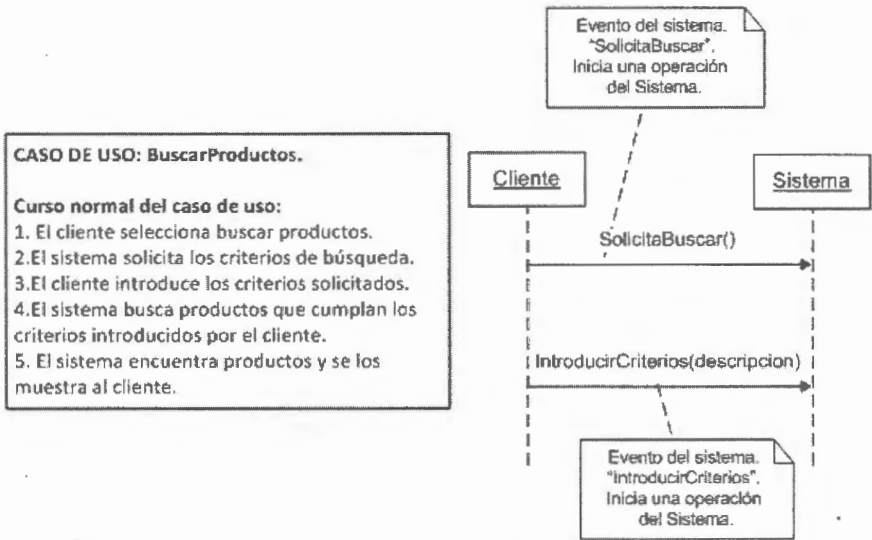
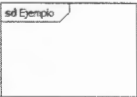
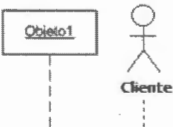







Figura 6.30. Diagrama de secuencia de sistema para el caso de uso *BuscarProductos*.

Cada rol, ya sea actor u objeto, se representa mediante un rectángulo distribuido horizontalmente en la zona superior del diagrama. A cada uno se le asocia una línea vertical llamada **línea de vida**, donde se describe la interacción a lo largo del tiempo. De cada línea vertical salen diferentes mensajes representados mediante flechas que muestran la interacción, encima de la flecha se muestra el mensaje. Es importante la secuencia de los mensajes en esta línea ya que indican el orden en que ocurren los eventos. A menudo el diagrama de secuencia va acompañado de la descripción del curso normal de eventos del caso de uso.

En la Figura 6.30 se muestran horizontalmente los roles que participan en la interacción: el cliente y el sistema, con sus líneas de vida (algunas herramientas gráficas, como WhiteStarUML, utilizan para representar el actor, el símbolo del monigote en lugar del rectángulo). Se muestran dos mensajes, que parten del cliente (están señalizados con una nota). El primer mensaje (también le podemos llamar evento), *SolicitaBuscar()*, inicia la operación de búsqueda de productos en el sistema. El segundo, *IntroducirCriterios(descripción)*, inicia la operación de búsqueda de productos según los criterios especificados, en este caso se ha considerado una búsqueda de productos por descripción.

Los principales elementos en un diagrama de secuencia son los siguientes:

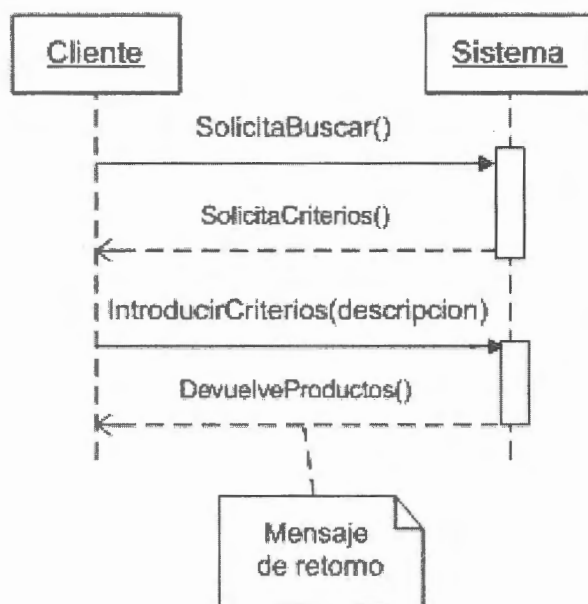
SÍMBOLO	FUNCIÓN	NOTACIÓN
<b>Marco</b>	Se utiliza para dar un borde visual al diagrama de secuencia. A la izquierda del marco se escribe la etiqueta <i>sd</i> seguida de un nombre	
<b>Línea de vida</b>	Representa a un participante durante la interacción. Normalmente contiene un rectángulo con el nombre del objeto y una línea punteada. Algunas veces un diagrama de secuencia tendrá una línea de vida con un símbolo de actor en la parte superior	
<b>Actor</b>	Representa el papel desempeñado por un usuario	
<b>Mensaje</b>	<b>Mensaje síncrono</b>	Mensaje() 
	<b>Mensaje asíncrono</b>	Mensaje() 
	<b>Mensaje de retorno</b>	Retorno 
<b>Activación</b>	Son opcionales. Representan el tiempo durante el que se ejecuta una función. Se suelen poner cuando está activo un método, ya sea porque está efectuando operaciones o porque se encuentra esperando la devolución de otro método	

Los mensajes representan la comunicación entre los participantes. Se dibujan como flechas dirigidas desde el participante que lo envía hasta el que lo ejecuta. Se etiquetan con un nombre acompañado o no de parámetros. Pueden ser de varios tipos:

- **Mensaje síncrono:** su comportamiento es el siguiente: cuando se envía un mensaje a un objeto, no se recibe el control hasta que el objeto receptor ha finalizado la ejecución.

- **Mensaje asíncrono:** representa flujo de control asíncrono. En este caso, quien envía un mensaje asíncrono continúa con su trabajo después de enviado, es decir, no necesita esperar la finalización del mismo en el objeto receptor. Este tipo de mensaje se representa mediante una flecha con la punta abierta o con media punta de flecha abierta. Se utiliza en sistemas multihilo donde se producen procesos concurrentes.
- **Mensaje de retorno:** representa un mensaje de confirmación. Su uso es opcional. Se representa mediante una flecha punteada.

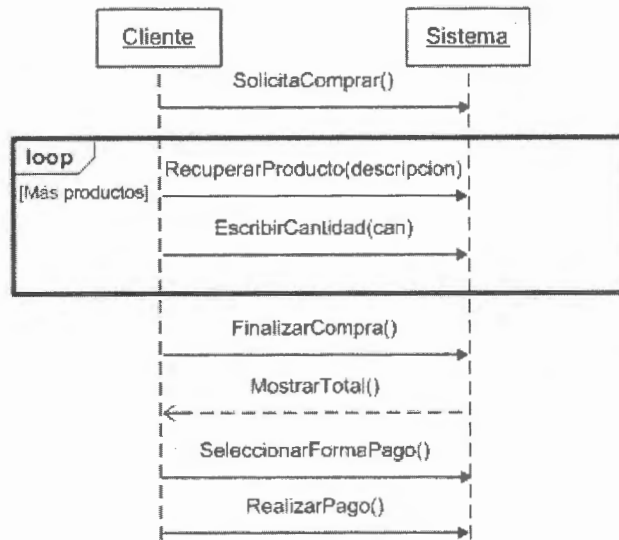
La Figura 6.31 muestra el diagrama de secuencia de sistema del caso de uso *BuscarProductos* con activaciones y mensajes de retorno. Tanto las activaciones como los mensajes de retorno son opcionales, conviene no abusar de ellos y utilizarlos solo para aumentar la claridad del diagrama.



**Figura 6.31.** Diagrama de secuencia con activaciones y mensajes de retorno.

En un diagrama de secuencia, los mensajes que se repiten se pueden incluir dentro de un marco. Por ejemplo, el curso normal del diagrama de casos de uso *ComprarProductos* se puede representar como se muestra en la Figura 6.32. Se encierra en una caja, etiquetada con **loop**, los eventos de recuperar el producto y escribir la cantidad del producto que el cliente desea comprar. Se escribe debajo la condición del bucle entre corchetes, para el ejemplo se ha etiquetado como *[Más productos]*, indicando que se repita mientras el cliente quiera comprar más productos.

Más adelante se mostrará como representar bucles y alternativas en los diagramas de secuencia utilizando una extensión denominada **fragmento combinado**; estos utilizarán una palabra clave o etiqueta (**loop**, **opt**, **alt**) para indicar el tipo.



**Figura 6.32.** Diagrama de secuencia de sistema del caso de uso *ComprarProductos*.

En los ejemplos vistos hasta ahora se ha utilizado el **diagrama de secuencia de sistema** para describir dentro de un caso de uso **los eventos que fluyen desde el cliente hacia el sistema**. Solo se incluía en la dimensión horizontal esos dos roles. En un sistema orientado a objetos, donde tenemos varias clases y varios objetos que se comunican unos con otros, las clases y los objetos se representan en la dimensión horizontal.

Existe una nomenclatura a la hora de nombrar los rectángulos que representan los objetos: **nombre:tipo**. Tanto el nombre como el tipo pueden omitirse, pero no ambos a la vez. El nombre representa el objeto dentro de la interacción, el tipo es el tipo de objeto (es decir, la clase). Ejemplos:

Una clase <i>Empleado</i>	<u>Empleado</u>
<i>emp</i> es un objeto de la clase <i>Empleado</i>	<u>emp:Empleado</u>
Un objeto cualquiera de la clase <i>Empleado</i>	<u>:Empleado</u>

**Ejemplo 13.** En el análisis del sistema de venta se han diseñado varias clases, una clase para datos de cliente, otra clase para datos de un producto, una clase para representar una venta y otra clase para representar una línea de venta. Un cliente puede tener muchas ventas, una venta puede tener muchas líneas de venta y cada línea de venta tiene un producto. Para representar que una venta puede tener muchas líneas de venta se define un atributo *ArrayList* en la clase *Venta*: *ArrayList<LineaVenta> lineas*. Estas clases se agrupan en el paquete *Datos*. El diagrama y sus relaciones se muestran en la Figura 6.33.

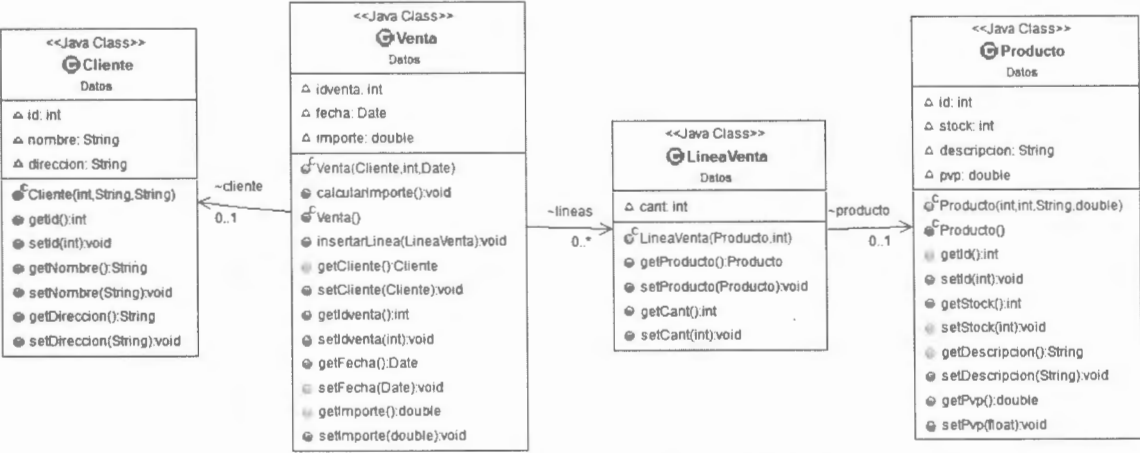


Figura 6.33. Diagrama de clases del paquete *Datos* del sistema de venta.

Todas ellas definen métodos get y set para acceder a los datos. Para la gestión de productos se ha diseñado otra clase llamada *GestorProductos* (incluida en el paquete *Gestion*) con las operaciones a realizar sobre los productos: recuperar, insertar, eliminar y modificar un producto, actualizar las existencias de un producto y obtener una lista con los identificadores de todos los productos, véase Figura 6.34.

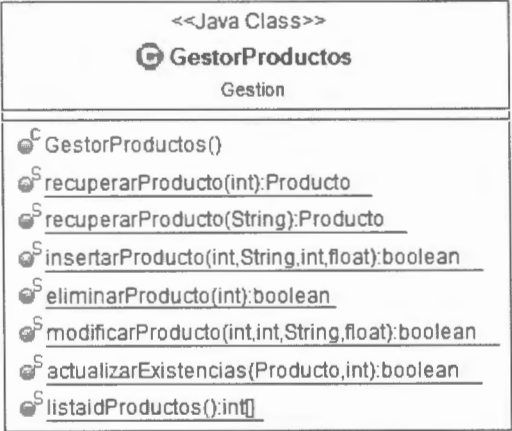


Figura 6.34. Clase *GestorProductos*.

El aspecto de esta clase es el siguiente:

```
package Gestion;
import Datos.Producto;
public class GestorProductos {
    public static Producto recuperarProducto(int id) {
        .....
        return producto;
    }

    public static Producto recuperarProducto(String descripcion) {
```

```

        .....
        return producto;
    }

    public static boolean insertarProducto (int id, String descripcion,
        int stock, float pvp) {
        .....
        return valor;
    }

    public static boolean eliminarProducto(int id) {
        .....
        return valor;
    }

    public static boolean modificarProducto(int id, int stock,
        String descripcion, float pvp) {
        .....
        return valor;
    }

    public static boolean actualizarExistencias(Producto p,
        int cantidad) {
        int cant = p.getStock() - cantidad;
        p.setStock(cant);
        return true;
    }

    public static int[] listaidProductos () {
        .....
        return valor;
    }
}

```

Se define también una clase *Pantalla* que constituye la interfaz gráfica desde la que el cliente se comunica con el sistema. El diagrama de secuencia correspondiente al caso de uso *BuscarProductos* es el mostrado en la Figura 6.35. Muestra lo siguiente:

- El cliente escribe la descripción del producto que desea buscar en la pantalla gráfica (objeto de la clase *Pantalla*)
- El objeto pantalla envía el mensaje *recuperarProducto(desc)* a la clase *GestorProductos* pidiéndola obtener los datos del producto cuya descripción se envía. Esta operación devuelve un objeto de la clase *Producto*, esto se indica en el rectángulo etiquetado como *p.Producto*. En este caso *GestorProductos* es una clase, no un objeto, entonces *recuperarProducto()* es un método estático.
- El obteto pantalla envía los mensajes *getId()*, *getPvp()* y *getStock()* al objeto *p.Producto* solicitando los datos del producto, el identificador, el pvp y el stock.

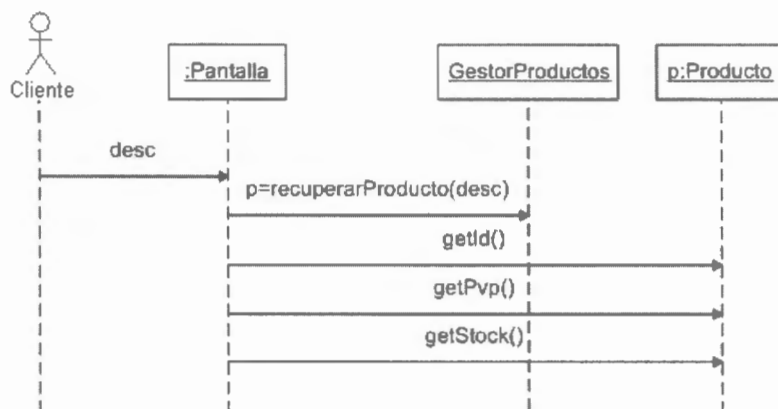


Figura 6.35. Diagrama de secuencia del Ejemplo 13.

## CREACIÓN Y DESTRUCCIÓN DE OBJETOS

En un diagrama de secuencia se puede mostrar la creación y destrucción de objetos. La creación se representa mediante un mensaje que termina en el objeto que va a ser creado. El mensaje puede llevar la identificación «create». La destrucción del objeto da lugar a la finalización de su línea de vida, se denota mediante una X grande en su línea de vida. El mensaje puede llevar la identificación «destroy». Las Figuras 6.36 y 6.37 muestran un ejemplo de creación y destrucción de objetos.

En la Figura 6.36 se muestra el diagrama de secuencia correspondiente al caso de uso de *InsertarProductos*. El administrativo escribe los datos del producto en la pantalla gráfica (objeto de la clase *Pantalla*). El objeto pantalla envía el mensaje *insertarProducto(id, desc, pvp, stock)* al gestor de productos (que se encarga de gestionar todas las operaciones para la gestión de productos) con los datos del producto a insertar. Desde el gestor de productos se creará el objeto *Producto* con los datos enviados por el objeto *Pantalla*.

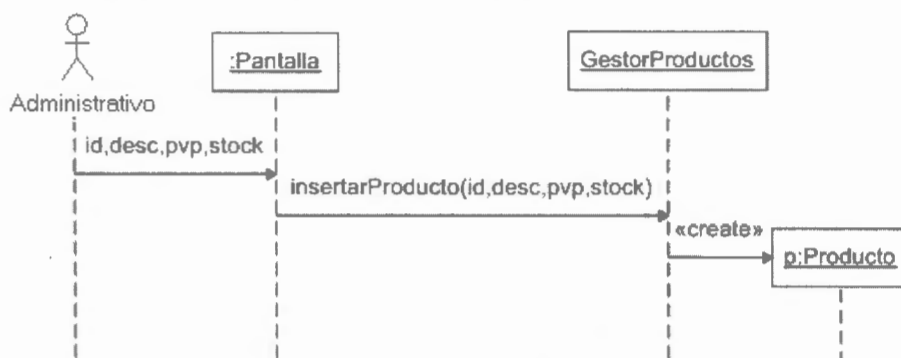


Figura 6.36. Creación de un objeto.

En la Figura 6.37 se muestra el diagrama de secuencia correspondiente al caso de uso de eliminar un producto. El administrativo escribe el identificador del producto a eliminar en la pantalla gráfica. El objeto pantalla envía el mensaje *eliminarProducto(id)* al gestor de productos con el identificador del producto a eliminar. Desde el gestor de productos se recupera el objeto *Producto* y se elimina incluyendo la X en su línea de vida.

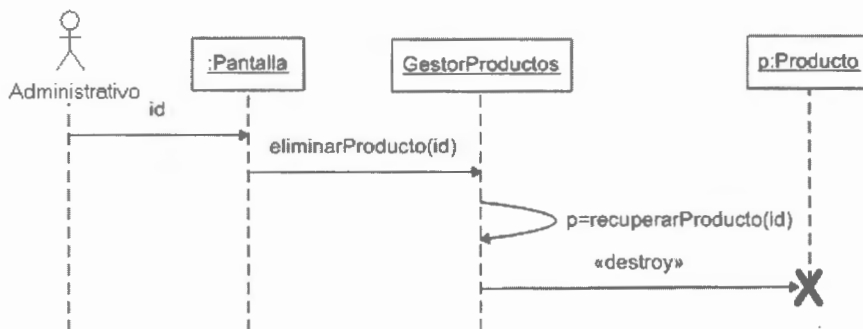


Figura 6.37. Destrucción de un objeto.

En la línea de vida de la clase *GestorProductos* se muestra un mensaje enviado a sí mismo (es decir, dentro de la clase *GestorProductos* se hace una llamada a la operación *recuperarProducto(id)*), a este mensaje se le llama **autodelegación** o **mensaje reflexivo**. La **autodelegación** es un mensaje que un objeto se envía a sí mismo regresando la flecha del mensaje de vuelta a la misma línea de vida.

Cuando se programa en Java normalmente no se destruye explícitamente un objeto, ya que el recolector de basura se encarga de esta acción. Sin embargo, hay veces que queremos resaltar lo que hacemos con el objeto.

## ALTERNATIVAS Y BUCLES

En los diagramas de secuencia podemos introducir extensiones para dar soporte a los bucles y a las alternativas. A estas extensiones se les llama **fragmentos combinados**, hay varios tipos, entre ellos están las alternativas y los bucles.

Ya vimos en el caso de uso *ComprarProductos* como se podían representar, encerrados en un marco o caja, los eventos que se repiten. En UML podemos modelar varios tipos de alternativas:

- Utilizando el operador **opt** seguido de una condición, si la condición se cumple, el contenido del marco se ejecuta. Véase Figura 6.38.
- Utilizando el operador **alt**, este va seguido de varias condiciones y al final la palabra clave *else*. El marco se divide en varias zonas, dependiendo de las condiciones que haya, cuyo contenido se ejecuta si se cumple la condición asociada. La parte *else* se ejecuta si no se cumplen las condiciones. Se utiliza para modelar estructuras *if...then...else*.

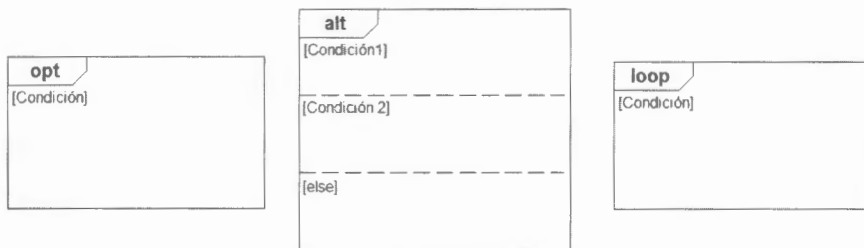


Figura 6.38. Marcos que representan alternativas y bucles.

Para representar un bucle se utiliza el operador **loop** seguido de una condición. Lo encerrado en el marco se ejecutará mientras se cumpla la condición. En la Figura 6.39 se muestra un diagrama de secuencia con un bucle y una alternativa. En este caso, el objeto pantalla manda el



mensaje *listaidProductos()* a la clase *GestorProductos* para obtener una lista con todos los identificadores de productos. En el bucle, para cada identificador de producto que se encuentre en la lista se obtiene un objeto *Producto*, y si el stock del producto es mayor que 100 se muestran sus datos.

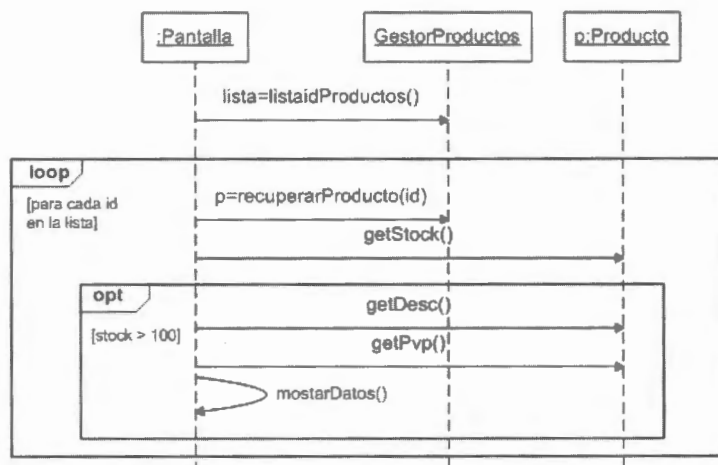


Figura 6.39. Diagrama de secuencia con bucle y alternativa.

**Ejemplo 14.** El siguiente ejemplo muestra el diagrama de secuencia considerando las clases y objetos que intervienen en el caso de uso *ComprarProductos*, véase Figura 6.41. Para la gestión de compras de productos se ha diseñado la clase llamada *GestorVentas* (incluida en el paquete *Gestion*) con las operaciones a realizar sobre las compras de productos: iniciar el proceso de venta, obtener el identificador de la venta, insertar los datos de la línea de venta en la venta, finalizar la venta y realizar el pago, véase Figura 6.40.

En Java la clase *GestorVentas* tiene el siguiente aspecto:

```

package Gestion;
import Datos.*;
public class GestorVentas {
    static Venta v;
    public static void iniciarVenta(Cliente c) {
    }

    private static int obtenerIdventa() {
        .....
        return valor;
    }

    public static void datosLinea(int id, int cantidad) {.....}

    public static void finalizarVenta()
    {
        v.calcularImporte();
        realizarPago();
    }

    private static void realizarPago() {.....}
}
  
```



Figura 6.40. Clase *GestorVentas*.

El diagrama de secuencia se muestra en la Figura 6.41. Muestra lo siguiente:

- Desde el objeto pantalla se envía el mensaje *iniciarVenta(Cliente)* a la clase *GestorVentas*.
- Desde esta clase se envía un mensaje a sí misma, *obtenerIdVenta()*, para obtener el identificador de la venta.
- Una vez obtenido el identificador se crea un objeto *Venta*.
- Se encierra en un marco etiquetado con **loop** el conjunto de operaciones que se van a repetir por cada producto que desee comprar el cliente.
- El objeto pantalla envía el mensaje *datosLinea(id, cantidad)* al gestor de ventas, con el identificador de producto y la cantidad deseada.
- Desde la clase *GestorVentas* se envía el mensaje *recuperarProducto(id)* a la clase *GestorProductos* solicitando los datos del producto cuyo identificador se envía. La operación devuelve un objeto de la clase *Producto*.
- Desde la clase *GestorVentas* se envía el mensaje *getStock()* al objeto producto recuperado para obtener el stock actual del producto.
- Se encierra en un marco etiquetado con **opt** el conjunto de operaciones que se realizan si hay stock en el producto solicitado.
- Si hay stock el gestor de ventas envía el mensaje *actualizarExistencias(id, cant)* al gestor de productos para que actualice el stock. Después crea un objeto *LineaVenta* con los datos del producto y la cantidad. Por último envía el mensaje *insertarLinea(lin)* al objeto venta creado inicialmente para añadir una nueva línea a la venta.
- Al finalizar el proceso repetitivo, cuando el cliente no desea comprar más productos, desde el objeto pantalla se envía el mensaje *finalizarVenta()* al gestor de ventas.
- Entonces el gestor de ventas envía el mensaje *calcularImporte()* al objeto venta creado.
- Por último, desde la clase *GestorVentas* se envía el mensaje de autodelegación *realizarPago()*.

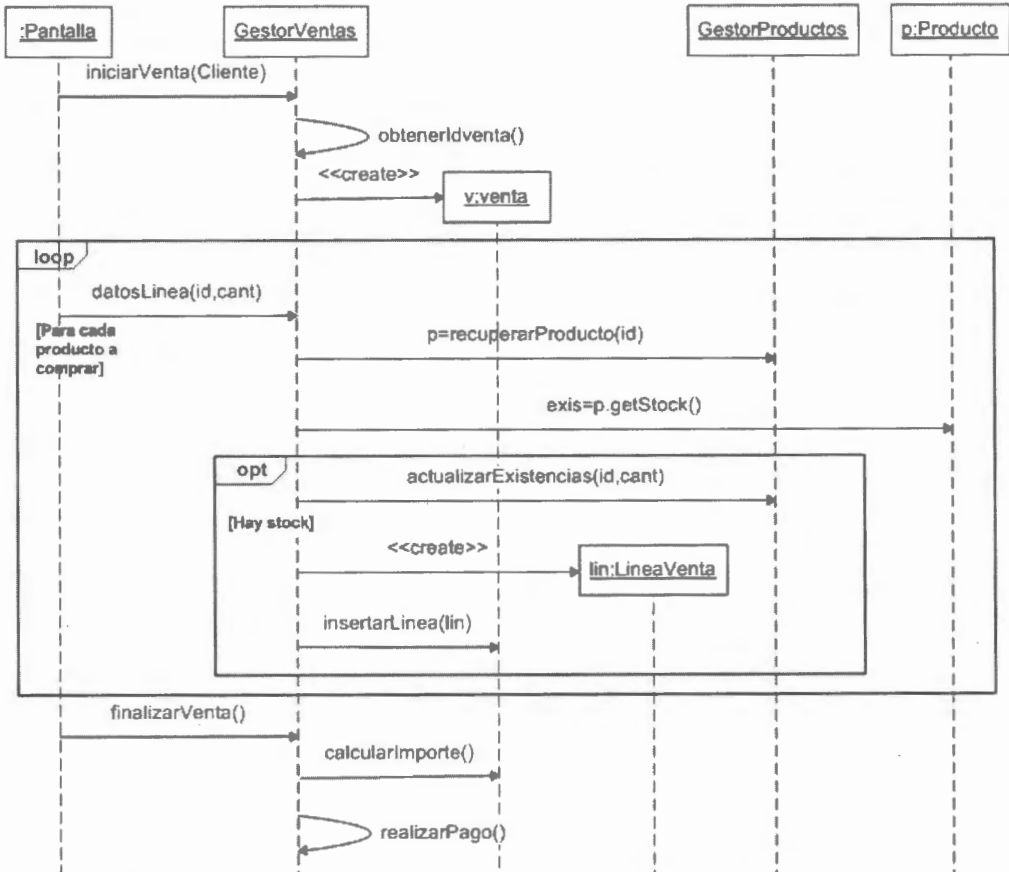


Figura 6.41. Diagrama de secuencia con clases y objetos, caso de uso *ComprarProductos*.

Podemos imaginar que desde el objeto: *Pantalla* se realizan las siguientes llamadas a las operaciones de la clase *GestorVentas*:

```

GestorVentas.iniciarVenta(Cliente);
while (el cliente quiera comprar productos) {
    //introducir datos del producto a comprar
    GestorVentas.datosLinea(id de producto, cantidad);
}
GestorVentas.finalizarVenta();
    
```

### 6.4.2. Diagrama de comunicación o colaboración

El diagrama de colaboración muestra los objetos junto con los mensajes que se envían entre ellos. Representa la misma información que el diagrama de secuencia, de hecho muchas de las herramientas UML permiten convertir el diagrama de secuencia en un diagrama de colaboración y viceversa.

A diferencia del diagrama de secuencia, centrado en el orden a través del tiempo en que ocurren los mensajes; el diagrama de colaboración se centra en el contexto y la organización general de los objetos que envían y reciben mensajes. El diagrama de objetos muestra los objetos y sus relaciones, el de colaboración es una extensión a este, además de la relación entre los objetos muestra los mensajes que se envían entre sí.

Un diagrama de colaboración es como un grafo con los siguientes elementos:

SÍMBOLO	FUNCIÓN	NOTACIÓN
<b>Objetos o roles</b>	Se representan mediante un rectángulo que contiene el nombre y la clase del objeto en el siguiente formato: <i>objeto:Clase</i> . Se puede omitir el nombre del objeto o la clase pero no ambos a la vez	<div style="border: 1px solid black; padding: 5px; display: inline-block;"><u>objeto:Clase</u></div>
<b>Enlaces</b>	Son los arcos del grafo que conectan los objetos. En un mismo enlace se pueden mostrar muchos mensajes, pero cada uno de ellos con un número de secuencia único	_____
<b>Mensajes</b>	Se representan mediante una flecha dirigida con un nombre y un número de secuencia	1: Mensaje() →
<b>Número de secuencia</b>	Indica el orden de un mensaje dentro de la interacción. Comienza en 1 y se incrementa en una unidad por cada nuevo mensaje en el flujo de control. El mensaje que inicia el diagrama no lleva número de secuencia	
<b>Iteración</b>	Se representa colocando un * después del número de secuencia y una condición encerrada entre corchetes. Ejemplo: 2*[para cada identificador de la lista]:prepara()	
<b>Alternativa</b>	Las alternativas se indican con condiciones entre corchetes, por ejemplo: 2 [condición]: operación(). Los caminos alternativos tendrán el mismo número de secuencia, seguido del número de subsecuencia	
<b>Anidamiento</b>	Se puede mostrar el anidamiento de mensajes con números de secuencia y subsecuencia. Por ejemplo 2.1, significa que el mensaje con número de secuencia 2 no acaba de ejecutarse hasta que no se han ejecutado todos los 2. X	

Para la numeración de los mensajes se pueden usar varios esquemas:

- Numeración simple: empieza en 1, se va incrementando en 1 y no hay ningún nivel de anidamiento; véanse Figuras 6.42 y 6.44.
- Numeración decimal: se muestran varios niveles de subíndices para indicar anidamiento de operaciones. Por ejemplo, 1 es el primer mensaje; 1.1 es el primer mensaje anidado en el mensaje 1, 1.2 es el segundo mensaje anidado en el mensaje 1; y así sucesivamente; véanse Figuras 6.43 y 6.45.

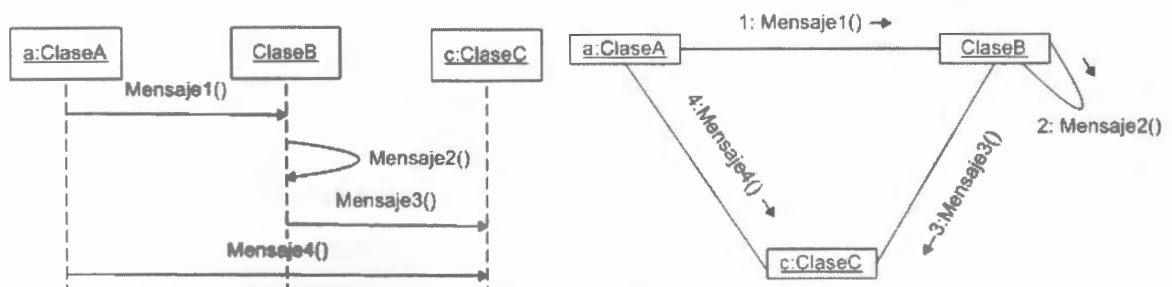


Figura 6.42. Diagrama de secuencia y diagrama de colaboración con numeración simple.

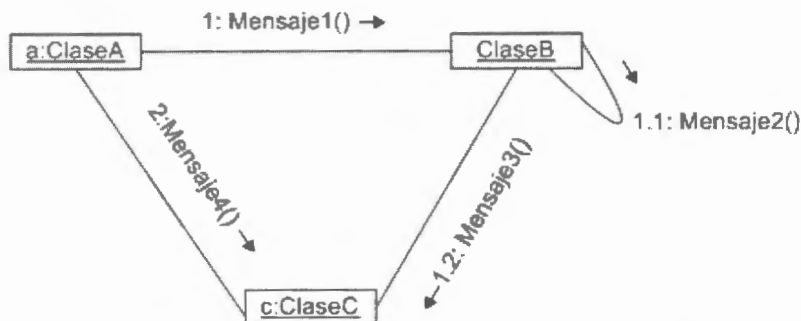


Figura 6.43. Diagrama de colaboración con numeración decimal.

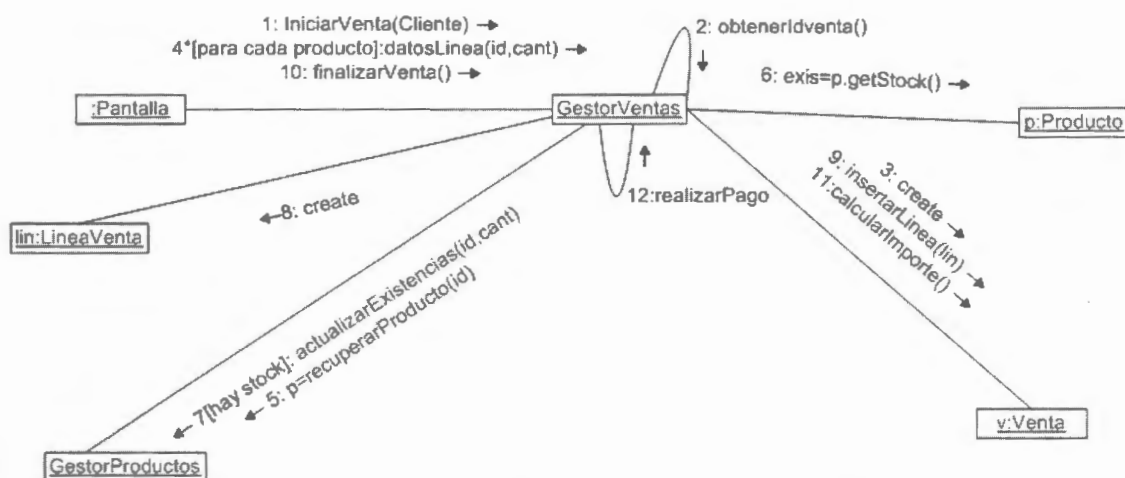


Figura 6.44. Diagrama de colaboración del caso de uso *ComprarProductos* con numeración simple.

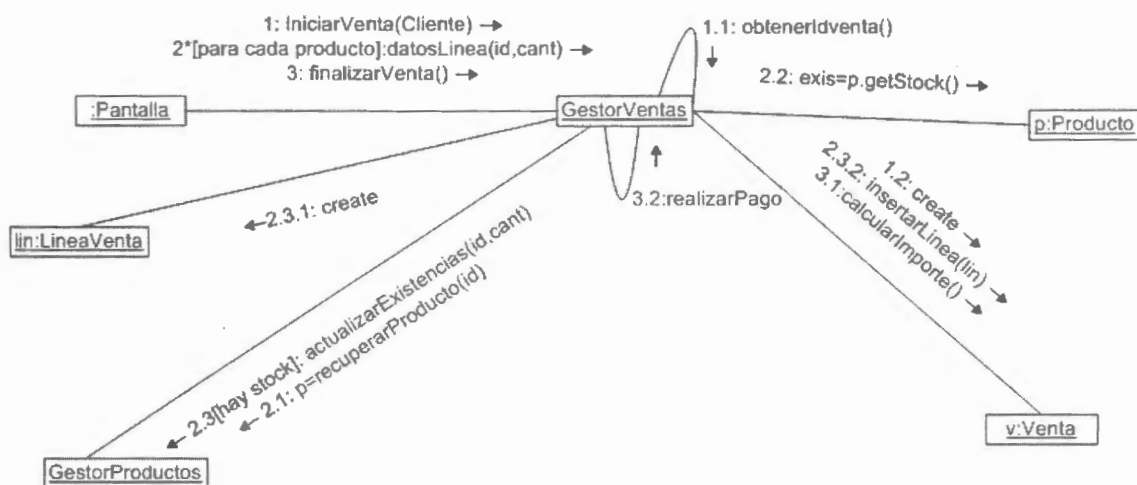


Figura 6.45. Diagrama de colaboración del caso de uso *ComprarProductos* con numeración decimal.

En las Figuras 6.44 y 6.45 se muestran diferentes esquemas para la numeración de los mensajes, quizá en la figura con numeración decimal resulte más difícil apreciar la secuencia general de los mensajes que en la figura con numeración simple.

diagrama de secuencia y escribir los mensajes entre emisor y receptor genera las operaciones para las clases. Para indicar la creación de un objeto se selecciona la propiedad *ActionKind* del mensaje y se elige la opción *CREATE*. Genera después del diagrama de colaboración. Desde las propiedades del mensaje en el diagrama de colaboración se puede documentar el bucle (*Detail->Iteration*) y la alternativa (*Detail->Branch*), véase Figura 6.66.

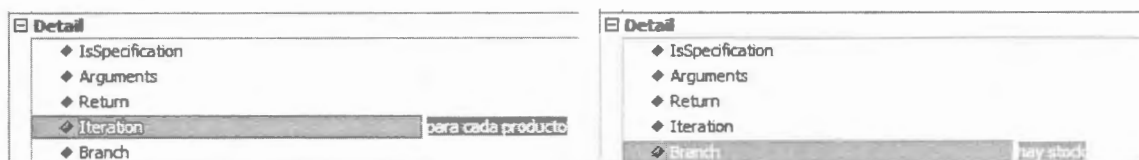


Figura 6.66. Bucle y alternativa en las propiedades del mensaje







## ARGOUMML


Al abrir ArgoUML en el modelo aparece un diagrama de clase y un diagrama de casos de uso. Creamos el diagrama de casos de uso y el diagrama de clases para la gestión de empleados. Para crear el diagrama pulsamos con el botón derecho del ratón sobre el modelo, a continuación elegimos *Crear diagrama -> Diagrama de secuencia*. Véase Figura 6.67. Desde las propiedades escribimos el nombre.




Figura 6.67. Crear diagrama de secuencia en ArgoUML.

En la parte superior de la pantalla de edición se muestra la barra de herramientas con varios botones:

-  *Rol clasificador nuevo*. Crea un nuevo rol en el diagrama con su línea de vida.
-  *Acción de llamada*. Crea un mensaje de llamada entre un emisor y un receptor, la creación del mensaje lleva incluido un mensaje de retorno.
-  *Crear acción de envío*. Crea un mensaje de envío entre emisor y receptor, el mensaje no incluye mensaje de retorno.
-  *Crear acción de retorno*. Crea un mensaje de retorno entre emisor y receptor.
-  *Crear acción*. Se utiliza para crear un objeto.
-  *Crear acción de destrucción*. Se utiliza para destruir un objeto.

 **Barrido.** Se pulsa y se arrastra por la pantalla, se realiza un barrido de todos los elementos.

 Se utiliza para añadir comentarios.

Partiendo del diagrama de casos de uso y del diagrama de clases, realizamos el diagrama de secuencia. Arrastramos el operador y las clases *VentanaEmple*, *OperacionesEmple* y *Empleado* a la zona de edición, y damos nombre a los elementos. Pulsamos en el botón  **Acción de llamada** para añadir los mensajes entre emisor y receptor. Pulsamos con el botón derecho del ratón sobre el mensaje, elegimos *Operation* y la operación que nos interese para que se muestre en el mensaje. Este tipo de mensaje crea dos mensajes uno de llamada y el de retorno, véase Figura 6.68.

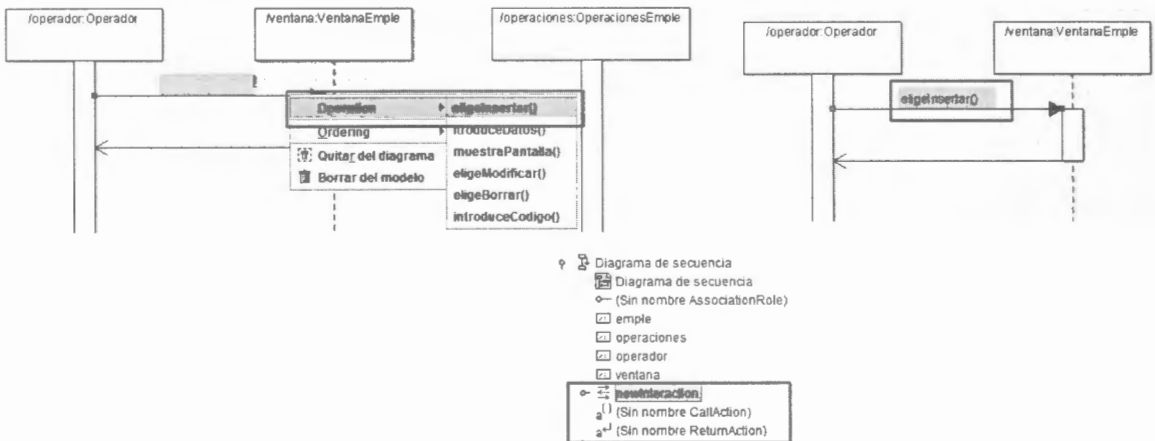



Figura 6.68. Añadir mensajes en ArgoUML.

Conviene dar un nombre a los mensajes que se van generando. Para añadir un mensaje de llamada a sí mismo pulsamos en el botón  **Acción de llamada**, y en la línea de vida donde queremos aplicarlo, dibujamos una especie de rectángulo haciendo clic en las esquinas y terminando en la línea de vida, véase Figura 6.69.

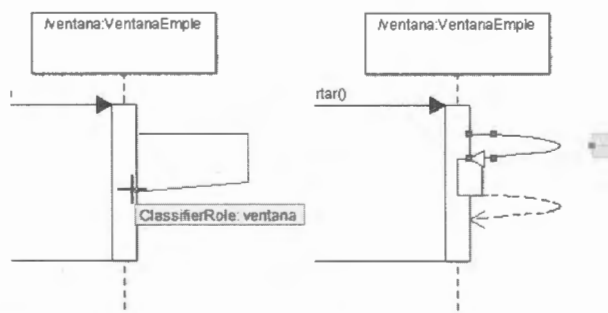



Figura 6.69. Mensaje de llamada así mismo en ArgoUML.

A la hora de crear el empleado podemos utilizar el botón  **Crear acción** que nos permite crear un objeto. El diagrama de secuencia con ArgoUML se muestra en la Figura 6.70.

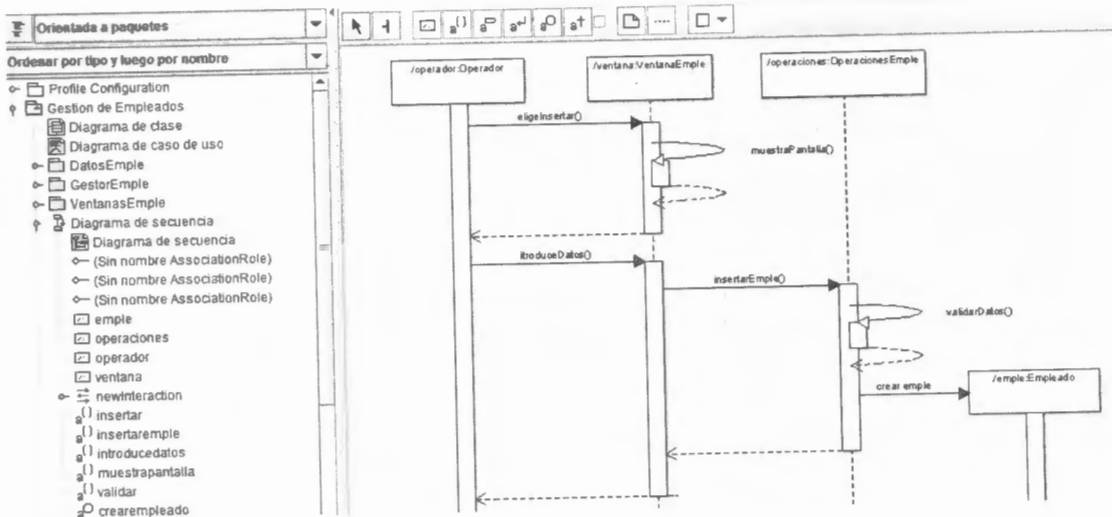


Figura 6.70. Diagrama de secuencia en ArgoUML.

La versión de ArgoUML utilizada soporta los diagramas UML 1.4 estándar, por ello no hay opciones para representar los bucles y las alternativas, ya que estas se definieron en el estándar UML 2.

El diagrama de colaboración se crea manualmente. Se puede crear dentro de un paquete, dentro del modelo o dentro de una colaboración (*unattachedCollaboration*). Por ejemplo, para crearlo dentro del modelo pulsamos sobre el modelo con el botón derecho del ratón y elegimos *Crear diagrama->Diagrama de colaboración*, véase Figura 6.71.

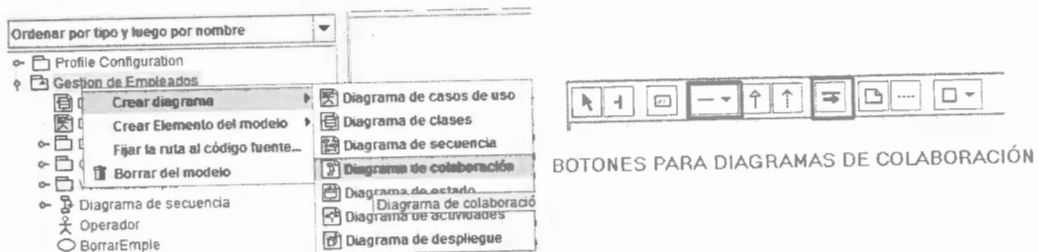




Figura 6.71. Crear diagrama de colaboración en ArgoUML.

Para crear el diagrama seleccionamos los elementos del explorador de la izquierda, el *Operador* y las clases *OperacionesEmple*, *VentanaEmple* y *Empleado*; y los arrastramos al área de edición del diagrama. Les asignamos un nombre.

Para crear la asociación entre dos elementos, pulsamos en el botón *Role de asociación nuevo*  y arrastramos desde el emisor al receptor. A continuación y teniendo la asociación seleccionada, pulsamos en el botón *Añadir mensaje* , vemos que aparece el número de secuencia. Hacemos doble clic en él para crear la acción de llamada. Podemos escribir el nombre de la operación (si no existe en la clase, entonces se creará), véase Figura 6.72.

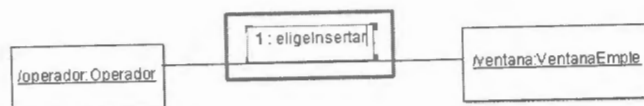
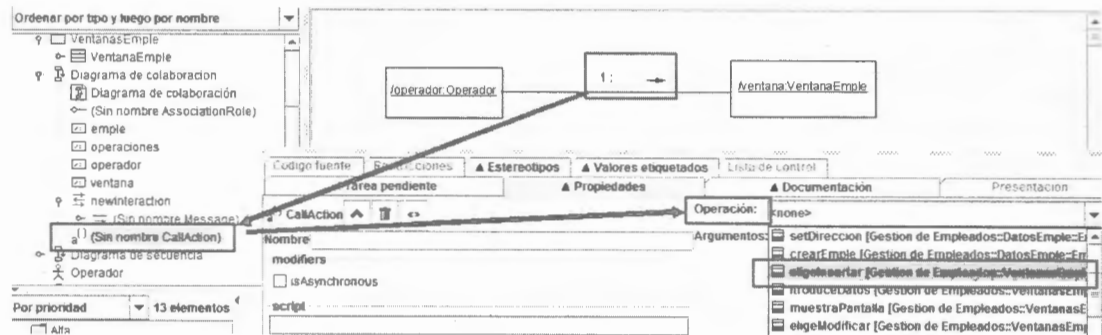


Figura 6.72. Añadir mensaje en el diagrama de colaboración en ArgoUML.

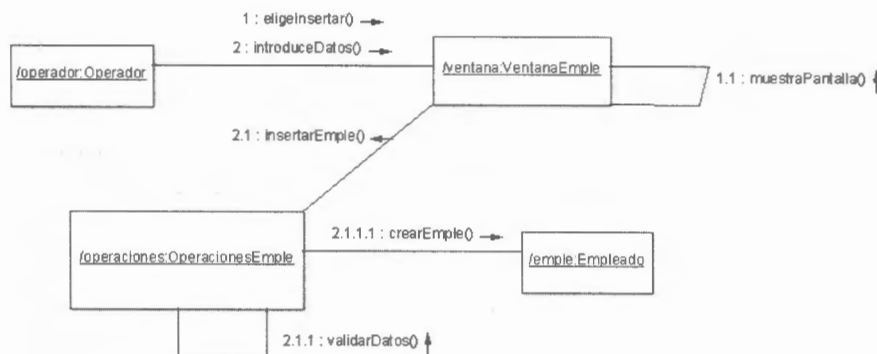


O bien, desde el explorador seleccionamos la acción de llamada y elegimos dentro de la pestaña de propiedades la operación *eligeInsertar()* de la lista *Operación*, véase Figura 6.73.



**Figura 6.73.** Selección de la acción de llamada en el diagrama de colaboración en ArgoUML.

Repetimos los pasos para crear el resto de asociaciones y mensajes entre los elementos del diagrama, en el orden en que ocurren para que los números de secuencia aparezcan ordenados. El resultado final se muestra en la Figura 6.74. Para crear varios mensajes en una misma asociación, se selecciona la asociación y después se pulsa el botón *Añadir mensaje*. Los mensajes a sí mismo se realizan como en el diagrama de secuencia.



**Figura 6.74.** Diagrama de colaboración en ArgoUML.

## COMPRUEBA TU APRENDIZAJE

1. Construye el diagrama de casos de uso para el siguiente enunciado: en una oficina se lleva a cabo la gestión de proyectos. La única persona que controla los proyectos es el administrador, cuyas funciones son las siguientes:

- Puede agregar, eliminar y actualizar un proyecto, pero para eliminar y actualizar es necesario encontrar el proyecto en cuestión.
- A la hora de actualizar un proyecto se pueden dar dos situaciones: cambiar la información sobre las tareas del proyecto o cambiar los recursos asociados al proyecto.
- Para informar a todos los miembros del equipo sobre los avances en el proyecto se envía un documento por e-mail.

Utiliza las relaciones «include», «extend» y de generalización que consideres.

2. Escribe un diagrama de casos de uso para modelar la interacción de un cliente y un empleado de un banco con un cajero automático. No utilices relaciones «extend» e «include». Las especificaciones son las siguientes:
  - El cajero automático lo puede utilizar el cliente del banco y el empleado de la sucursal.
  - El cliente debe identificarse en el cajero antes de realizar cualquier operación.
  - El cliente puede cambiar el pin, sacar dinero, consultar el saldo y consultar los últimos movimientos.
  - El empleado utiliza el cajero únicamente para reponer dinero.
3. Realiza una segunda aproximación del diagrama del ejercicio 2 considerando la relación «extend».
4. Realiza una tercera aproximación del diagrama del ejercicio 2 considerando la relación «include».
5. Realiza la descripción del caso de uso de sacar dinero del ejercicio 2. La situación que vive el cliente al retirar el dinero es la siguiente: el cliente se acerca al cajero automático de su banco, introduce la tarjeta, escribe el pin, solicita retirar dinero y escribe la cantidad a retirar. El cajero le entrega el dinero solicitado siempre y cuando la operación se pueda realizar. Por último, el cliente retira la tarjeta y se va. No incluyas alternativas.
6. Completa la descripción del caso de uso anterior incluyendo las siguientes alternativas: PIN incorrecto para la tarjeta, PIN introducido de forma incorrecta durante 3 veces consecutivas y la cantidad solicitada supera el saldo.
7. Realiza un diagrama de actividad para el caso de uso de sacar dinero del cajero automático.
8. Disponemos de las siguientes clases Java incluidas en el paquete con nombre *Datos*:

<pre> package Datos;  public class Puerta {     private double alto;     private double ancho;      public Puerta(double alto,                   double ancho) {         this.alto = alto;         this.ancho = ancho;     }     public double getAlto() {         return alto;     }     public void setAlto(double alto) {         this.alto = alto;     }     public double getAncho() {         return ancho;     }     public void setAncho(double ancho) {         this.ancho = ancho;     } } </pre>	<pre> package Datos;  public class Ventana {     private double alto;     private double ancho;      public Ventana(double alto,                   double ancho) {         this.alto = alto;         this.ancho = ancho;     }     public double getAlto() {         return alto;     }     public void setAlto(double alto) {         this.alto = alto;     }     public double getAncho() {         return ancho;     }     public void setAncho(double ancho) {         this.ancho = ancho;     } } </pre>
<pre> package Datos;  public class Habitacion {     Puerta puerta;     Ventana ventana;     double metros;      public Habitacion(double metros) {         this.metros = metros;     }     public void setPuerta(Puerta puerta) {         this.puerta = puerta;     }     public Ventana getVentana() {         return ventana;     }     public void         setVentana(Ventana ventana) {         this.ventana = ventana;     }     public double getMetros() {         return metros;     }     public void setMetros(double metros) {         this.metros = metros;     } } </pre>	<pre> package Datos;  public class Casa {     int numhabit;//num de habitaciones     Habitacion [] habitaciones;     int n; //para llenar el array      public Casa(int numhabit) {         this.numhabit=numhabit;         habitaciones =             new Habitacion[numhabit];         n=0;     }      public void addHabitacion         (Habitacion h){         habitaciones[n] = h;         n++;     }      public Habitacion[] getHabitaciones()     {         return habitaciones;     } } </pre>

Se pide realizar el diagrama de secuencia para la operación *main()* de la clase *ConstruirCasal*:

```

package Proceso;
import Datos.*;

public class ConstruirCasal {
    public static void main(String[] args) {
        Casa casa = new Casa(1); //casa con una habitación
        Habitacion h = new Habitacion(15); //15 metros
    }
}

```

```

Puerta p = new Puerta(2.10, 1); // alto 2.10, ancho 1

h.setPuerta(p);
Ventana v = new Ventana(1.60, 1.20); // alto 1.60 ancho 1,20
h.setVentana(v);

casa.addHabitacion(h); //añadir habitación a la casa

Habitacion hab[] = casa.getHabitaciones();
System.out.println("Número de Habitaciones:" + hab.length);
}
}

```

## 9. Construye el diagrama de secuencia para la operación *main()* de la clase *ConstruirCasa2*:

```

package Proceso;
import Datos.*;

public class ConstruirCasa2 {
    public static void main(String[] args) {
        Casa casa = new Casa(1); //casa con una habitación
        boolean conventana = true;
        Habitacion h = new Habitacion(15);
        Puerta p = new Puerta(2.10, 1); // alto 2.10, ancho 1
        h.setPuerta(p);

        if (conventana) {
            Ventana v = new Ventana(1.60, 1.20); // alto 1.60 ancho 1.20
            h.setVentana(v);
        }

        casa.addHabitacion(h); //añadir habitación a la casa

        Habitacion hab[] = casa.getHabitaciones();
        System.out.println("Número de Habitaciones:" + hab.length);
    }
}

```

## 10. Construye el diagrama de secuencia para la operación *main()* de la clase *ConstruirCasa3*:

```

package Proceso;
import Datos.*;

public class ConstruirCasa3 {
    public static void main(String[] args) {
        //Crear una casa con 4 habitaciones
        Casa casa = new Casa(4);
        boolean conventana = true;

        for (int i = 0; i < 4; i++) {
            Habitacion h = new Habitacion(15);
            Puerta p = new Puerta(2.10, 1); // alto 2.10, ancho 1
            h.setPuerta(p);

            if (conventana) {
                Ventana v = new Ventana(1.60, 1.20);
                h.setVentana(v);
            }
            casa.addHabitacion(h);
        }
    }
}

```

```

    Habitación hab[] = casa.getHabitaciones();
    System.out.println("Número de habitaciones:" + hab.length);
}
}

```

11. Crea el diagrama de colaboración de los diagramas de secuencia creados anteriormente. Utiliza para ello la herramienta WhiteStarUML.
12. La Figura 6.126 muestra un diagrama de secuencia de una aplicación java en la que un usuario se identifica ante el sistema. Construye a partir de este diagrama las clases y métodos que intervienen en la aplicación:

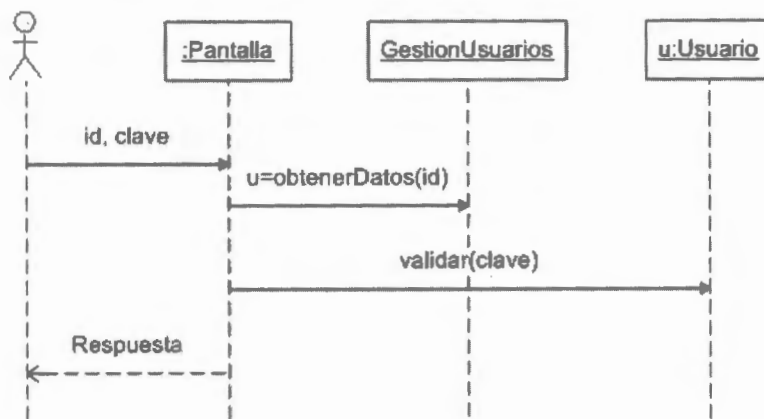


Figura 6.126. Diagrama de secuencia del Ejercicio 12.

13. Realiza el diagrama de casos de uso para una empresa de Radiotaxis que ha solicitado el desarrollo de un sistema que le apoye en sus procesos clave. Hay tres tipos de usuarios: administrativos, choferes, y el gerente. Los requerimientos son los siguientes:

Los administrativos de la empresa de Radiotaxis podrán:

- Dar de alta nuevos clientes.
- Dar de alta reservas de viajes indicando el cliente, el chofer solicitado, la dirección de origen, de destino y la fecha y hora de salida. Será necesario consultar los datos del chofer solicitado para comprobar si está disponible. Si al dar de alta una reserva, el cliente no existe en el sistema se podrá dar de alta directamente. Desde aquí también se podrá confirmar la reserva que se está dando de alta.
- Confirmar y cancelar las reservas ya dadas de alta.

Los choferes podrán consultar las reservas que tienen asignadas para el día de la fecha.

El gerente podrá dar de alta nuevos choferes al sistema y liquidar las comisiones de los choferes mensualmente, para ello se necesita consultar la información del chofer.

14. Realiza la descripción del caso de uso de dar de alta reservas de viajes. Considera los pasos que lleva a cabo el administrativo para dar de alta las reservas. Ten en cuenta varias alternativas: si el cliente no existe se puede dar de alta, chofer no disponible o inexistente, datos incorrectos, etc.