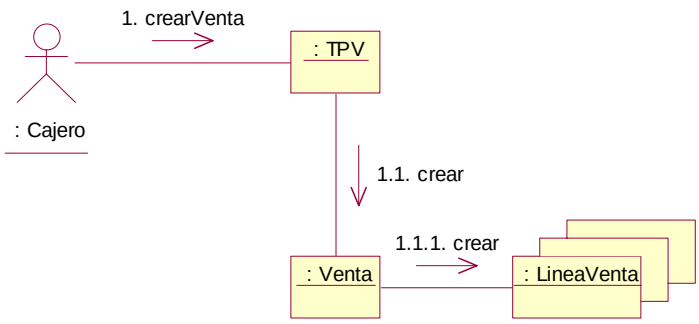
Ejercicios Prácticos y Teóricos

Diagramas de interacción y de UML a código Java

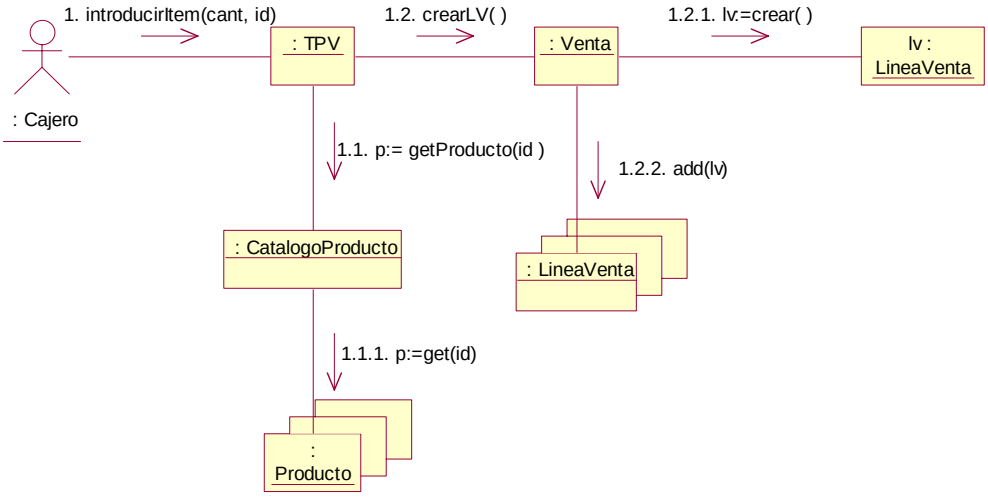
Ejercicio 1. Gestión de un Terminal de Punto de Venta (TPV)

Dados los siguientes diagramas de interacción, correspondientes a la gestión de un TPV, obtener el modelo del dominio mediante un diagrama de clases con toda la información posible (atributos y métodos en las clases, así como las relaciones entre dichas clases).

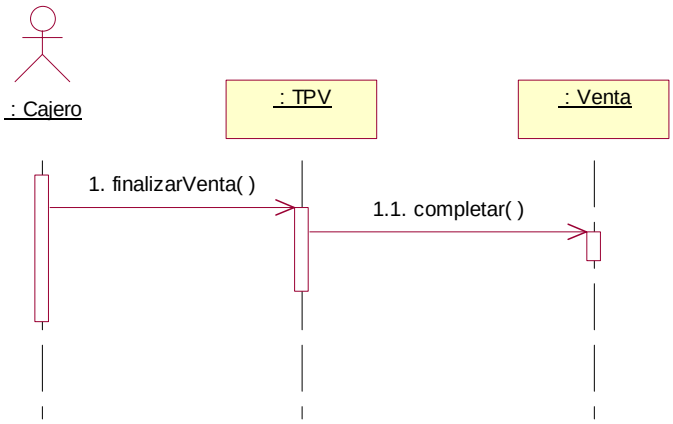
1. CrearVenta



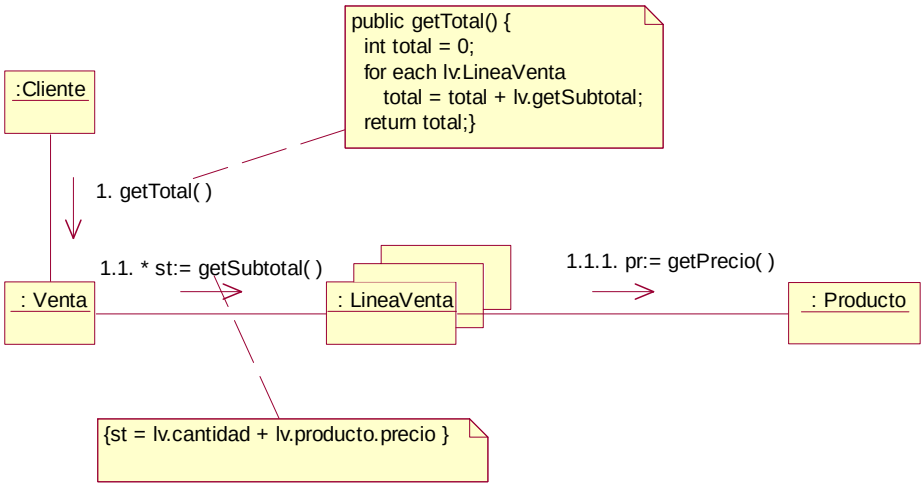
2. *IntroducirItem*



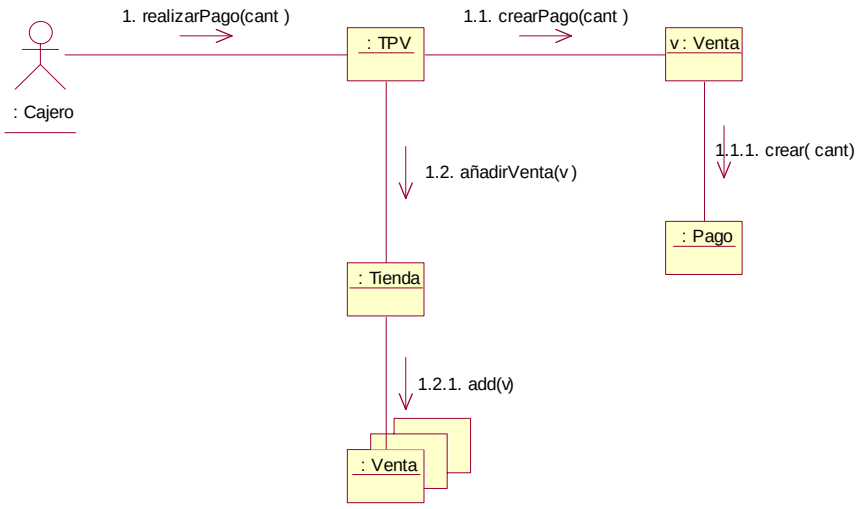
3. *FinalizarVenta*



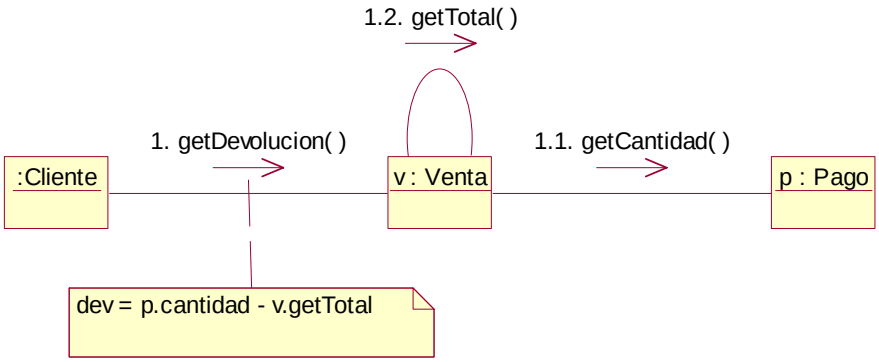
4. *CalcularTotal*



5. *RealizarPago*



6. *CalcularDevolución*



Ejercicio 2. Gestión de un TPV

Dado el código Java de las clases, correspondientes a la gestión de un TPV, obtener el diagrama de clases con toda la información posible (atributos y métodos en las clases, así como las relaciones entre dichas clases). Además, se debe comprobar que coincide con el diagrama de clases anterior. Si no es así, entonces se tiene que corregir el código Java de estas clases de acuerdo con el diagrama de clases anterior.

```
public class Pago {
        private Dinero cantEntregada;

        public Pago (Dinero cantidad) { cantEntregada = cantidad; }
```

```
        public Dinero getCantEntregada () { return cantEntregada; }
}

public class CatalogoProducto {
        private Map productos = new HashMap ();

        public CatalogoProductos () {
                ItemId id1 = new ItemID(100);
                ItemId id2 = new ItemID(200);
                Dinero precio1 = new Dinero (3);
                Dinero precio2 = new Dinero (5);
                Producto p;
                p:=  new Producto (id1, precio1, “producto 1”);
                productos.put(id1, p);) }
                p = new Producto (id2, precio2, “producto 2”);
                productos.put(id2, p);) }

        public Producto getProducto (ItemId id) {
                return (Producto) productos.get(id); }
}

public class TPV {
        private CatalogoProducto catalogo;
        private Venta venta;

        public TPV(CatalogoProducto cp) { catalogo = cp; }
        public void crearNuevaVenta () {venta = new Venta();}
        public void finalizarVenta () { venta.completar(); }
        public void introducirItem (ItemId id, int cant) {
                Producto p = catalogo.getProducto (id);
                Venta.crearLineaVenta(p, cant); }
        public void realizarPago() { venta.crearPago(cant)}
}

public class Producto {
        private itemID id;
        private Dinero precio;
        private String descripcion;

        public Producto(ItemID id, Dinero precio, String desc) {
                this.id = id;
                this.precio = precio;
                this.descripcion = desc;}
        public ItemId getId() { return id; }
        public Dinero getPrecio() { return precio; }
        public String getDescripcion() { return descripcion; }
}

public class Venta {
        private List lineaVentas = new ArrayList();
        private Date fecha = new Date();
        private boolean esCompleta;
        private Pago pago;

        public Dinero getDevolucion() { return pago.getCantEntregada(). minus(getTotal() ); }
        public void completar() { esCompleta = true; }
        public void crearLineaVenta(Producto p, int cant) {
                lineaVentas.add(new LineaVenta(p,cant)); }

        public Dinero getTotal() {
```

```
                Dinero total = new Dinero();
                Iterator i = lineaVentas.iterator();
                while (i.hasNext()) {
                        LineaVenta lv = (LineaVenta) i.next();
                        total.add(lv.getSubtotal()); }
                return total;
        }
        public void crearPago (Dinero cantEntregada) { pago = new Pago(cantEntregada); }
}

public class LineaVenta {
        private int cantidad;
        private Producto producto;

        public LineaVenta(Producto p, int cant) {
                this.producto = p;
                this.cantidad = cant; }
        public Dinero getSubtotal () { return producto.getPrecio().times(cantidad); }
}

public class Tienda {
        private CatalogoProducto catalogo;
        private TPV tpv;

        public TPV getTPV {return TPV; }
}
```

Ejercicios teóricos

Pregunta 1.

Construir los diagramas de interacción para modelar cada uno de los siguientes comportamientos dinámicos:

a. Un objeto de clase *ClaseA* recibe como punto de entrada un mensaje *mensaje1()* y si la condición *cond* se satisface, envía un *mensaje2()* a un objeto de clase *ClaseB*, y en caso contrario, envía un mensaje *mensaje3()* a un objeto de clase *ClaseC*.

b. Un objeto de clase *ClaseA*, al recibir el mensaje *mensaje1()*, crea una instancia de clase *ClaseB*, y se lo envía a un objeto de clase *ClaseC*, el cual envía el mensaje *mensaje3()* a la instancia recién creada. Este objeto (la instancia de la clase *ClaseB*) retorna un valor *valorRetorno*. Al recibir dicho valor el objeto de clase *ClaseC* envía al objeto de clase *ClaseB* un mensaje *destroy()* con el que se elimina este último objeto.

Pregunta 2.

¿Qué tipo de relaciones pueden existir entre una clase y una interfaz?

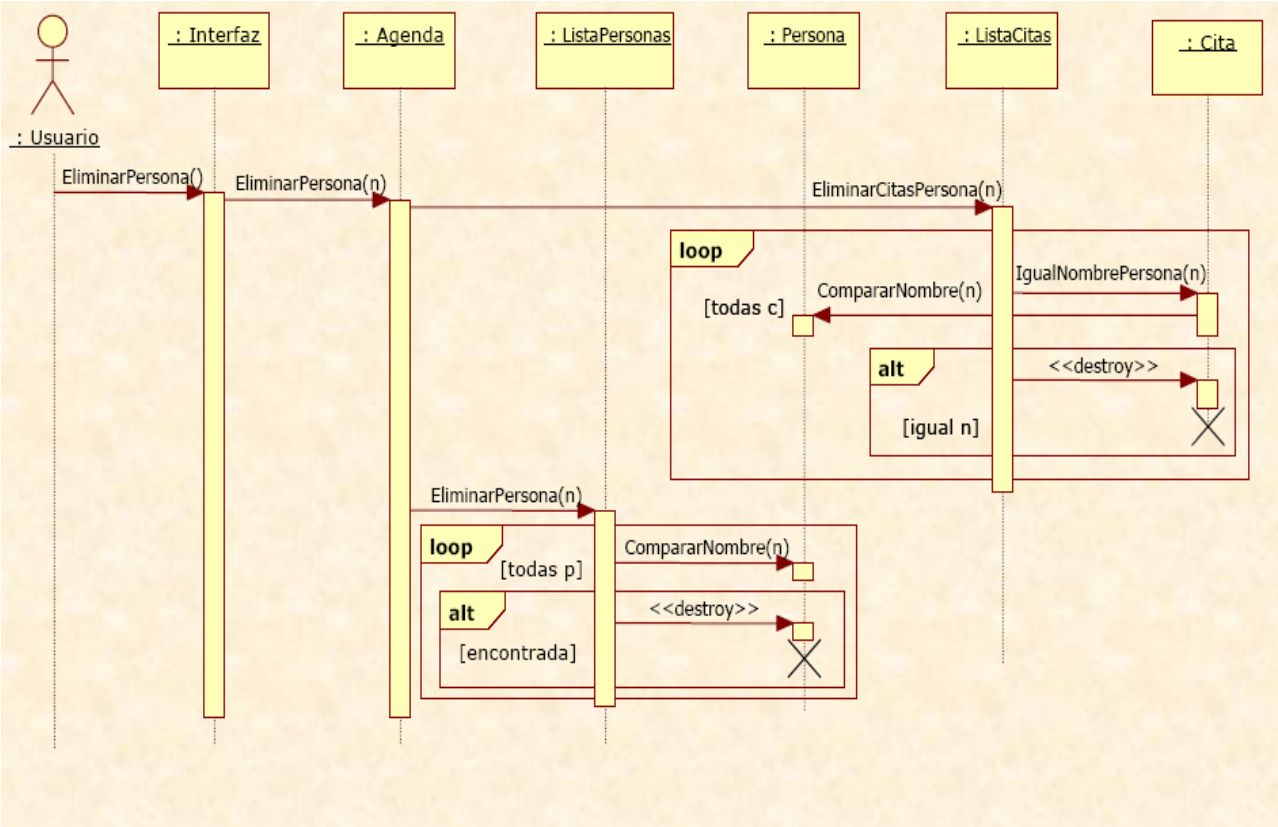
Se pide:

a. Explicar cada una de estas relaciones mediante ejemplos ilustrativos en UML, incluyendo los atributos y métodos mínimos que sean necesarios.

b. Realizar la implementación en Java de los ejemplos anteriores.

Pregunta 3.

Dado el siguiente diagrama de secuencia, correspondiente al escenario del caso de uso para eliminar una persona de una agenda, representar el diagrama de clases parcial en UML con toda la información posible (atributos y métodos en las clases, y relaciones entre dichas clases).



Pregunta 3.

A continuación se describe detalladamente un conjunto de clases, interfaces y métodos junto con su funcionamiento:

- *ICliente* es una interfaz que especifica un método llamado *service()*.
- *Clase* es una clase que incluye una referencia a un objeto *another* de tipo *OtraClase* y un método llamado *useful()* que invoca a *helper()* sobre el objeto *another*.
- *OtraClase* contiene un atributo *text* que se inicializa con el String "¿Necesitas ayuda?" y un método *helper()* que simplemente imprime el contenido de *text* en la salida estándar o *System.out*.
- *MiClase* es una clase que implementa *ICliente* y que hereda de *Clase*. La implementación de *service()* simplemente invoca a *useful()* sobre el objeto *this*.
- El método *main()* debe crear primero un objeto de *MiClase* y luego una instancia de *OtraClase* que se asigna al campo *another* del primero (del objeto de *MiClase*). Finalmente el método *main()* llama a *service()* sobre la instancia de *MiClase*.

Se pide:

a. Realizar la implementación en Java de la descripción anterior, considerando todos los atributos y todos los métodos públicos.

b. Dibujar el diagrama de clases (atributos y métodos) asociados a la descripción dada, respetando los nombres codificados en el apartado anterior.

c. Dibujar el diagrama de secuencia (objetos y mensajes) que se desencadena a partir de la ejecución de *main()* y que debe incluir todos los nombres implementados en el apartado a).