

UDE - Licenciatura en Informática
Punta del Este

TALLER DE VERANO
2026

Faustino Santos - 5.419.343-1

Felipe de León - 5.511.520-4

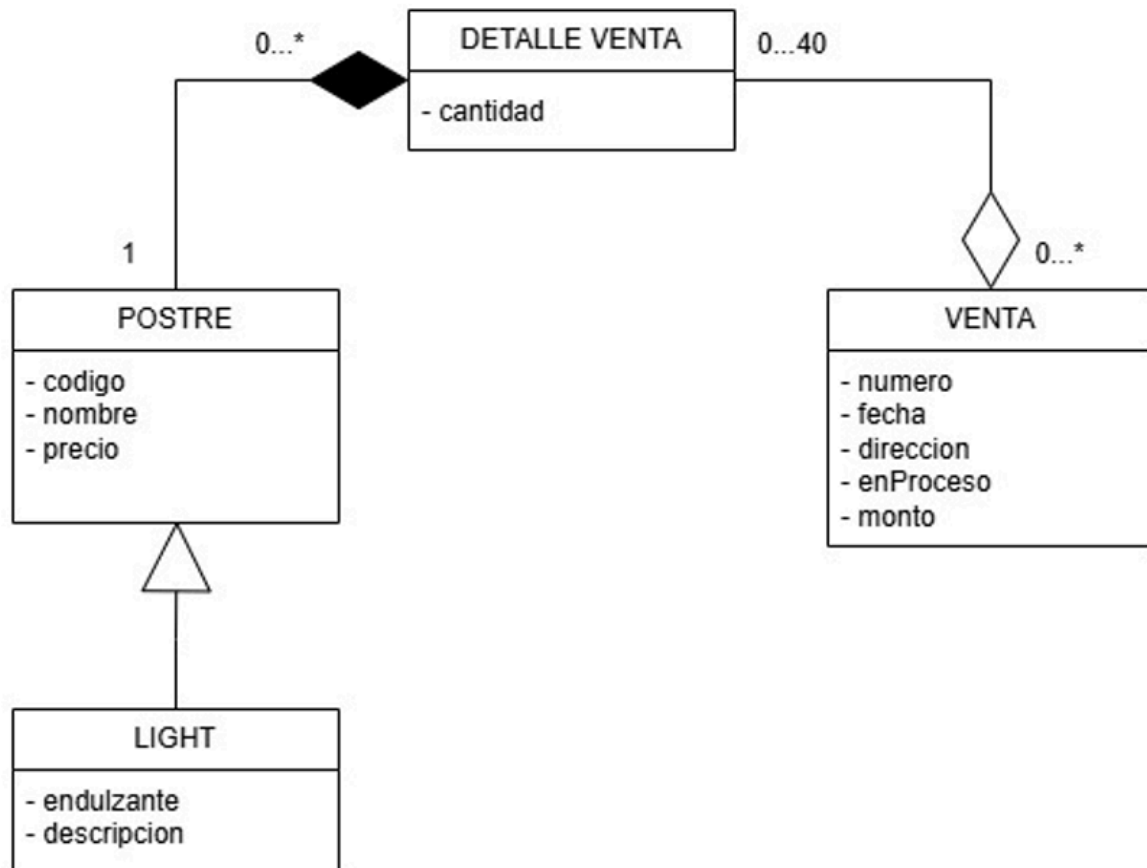
Nicolás Podchibiakin - 5.051.944-1

Tutor: Leonardo Domínguez

Parte A.....	1
Diagrama de clases conceptual.....	1
Parte B.....	2
Elección del diseño.....	2
Desglose de requerimientos.....	3
Diagrama de clases de implementación.....	10
Elección de estructuras de datos.....	11

Parte A

Diagrama de clases conceptual



Parte B

Elección del diseño

Definición de tipos abstractos de datos a utilizar

Postre = código x nombre x precio

Light = Postre + (endulzante x descripción)

DetalleVenta = Postre x cantidad

Venta = número x fecha x dirección x estado x monto x Secuencia (DetalleVenta)

Ventas = Secuencia (Venta)

Postres = Diccionario (Postre)

Análisis de elección de tipos abstractos de datos a utilizar

Postre = código x nombre x precio

Se representará a cada postre común mediante un producto cartesiano de los datos código (string), nombre (string), precio (double).

Light = Postre + (endulzante x descripción)

Cada postre light se heredará los datos de un Postre adicionando al mismo sus datos específicos: dulce (string), descripción (string).

DetalleVenta = Postre x cantidad

Cada detalle de venta será representado con un producto cartesiano de un postre y una cantidad (int).

Venta = id x fecha x dirección x enProceso x monto

Una venta será representada con un producto cartesiano de los datos id (), fecha (), dirección(), enProceso () y monto ()

Ventas = Secuencia (Venta)

Se optó por utilizar una secuencia para representar la colección de ventas registradas en el sistema. Dicho tipo abstracto, permitirá realizar inserciones al final de la colección de forma eficiente para mantener el orden de los elementos.

Postres = Diccionario (Postre)

Para representar la colección de postres, se utilizará un diccionario. De esta manera los postres podrán ser encontrados de forma eficiente por su identificador alfanumérico.

Desglose de requerimientos

1. Alta de nuevo postre

E: Datos de nuevo postre

S: Registro de nuevo postre en diccionario de Postres

M:

```
SI Diccionario(Postres).Member(VO_Postre.getCodigo()) ENTONCES
    Desplegar error: "El código ingresado ya está registrado para
    un postre"
SINO
    SI VO_Postre.getTipo() == "Común"
        nuevoPostre = Crear Postre(VO_Postre.getCodigo(),
        VO_Postre.getNombre(), VO_Postre.getPrecio())
    SINO
        nuevoPostre = Crear Light(VO_Postre.getCodigo(),
        VO_Postre.getNombre(), VO_Postre.getPrecio(),
        VO_Postre.getDescripcion(), VO_Postre.getEndulzante())
    FIN SI
    Diccionario(Postres).Insert(nuevoPostre)
FIN SI
```

2. Listado general de postres

E: -

S: Se retorna un listado de los postres registrados

M:

```
nuevaListaPostres = Diccionario(Postres).ListarPostres()
Devolver nuevaListaPostres
```

3. Listado detallado de un postre

E: Código de postre

S: Retorna los datos detallados de un postre

M:

```
SI !Diccionario(Postres).Member(código) ENTONCES
    error: "El código ingresado no está registrado
    para ningún postre".
SINO
    postreBuscado = Diccionario(Postres).Find(código)

    SI postreBuscado.getTipo() == "Común" ENTONCES
        postreVO = Crear VO_Postre(postreBuscado.getCodigo(),
        postreBuscado.getNombre(), postreBuscado.getPrecio(),
        postreBuscado.getTipo())
    SINO
        postreVO = Crear VO_Light(postreBuscado.getCodigo(),
        postreBuscado.getNombre(), postreBuscado.getPrecio(),
```

```
postreBuscado.getTipo(), postreBuscado.getDescripcion(),  
postreBuscado.getEndulzante())
```

FIN SI

FIN SI

Devolver postreVO

4. Comienzo de venta

E: Fecha y dirección de entrega de una nueva venta

S: Se registra una nueva venta en Secuencia(Ventas)

M:

```
ultimaVenta = Secuencia(Ventas).getUltimaVenta()
```

```
SI ultimaVenta != null && ultimaVenta.getFecha() <
```

```
VO_VentaBasico.getFecha() ENTONCES
```

```
    error: "La fecha de la nueva venta no puede ser menor que la  
    de la última venta registrada"
```

SINO

```
    enProceso = true
```

```
    monto = 0
```

```
    nuevaVenta = Crear Venta(VO_VentaBasico.getFecha(),
```

```
VO_VentaBasico.getDireccion(), enProceso, monto)
```

```
    nuevaVenta.setNumero(ultimaVenta ? (ultimaVenta.getNumero() +  
1): 1)
```

```
    Secuencia(Ventas).InsBack(nuevaVenta)
```

FIN SI

5. Agregar postre a la venta

E: Código de postre, cantidad de unidades y número de venta

S: Agregar dicha cantidad de unidades del postre a esa venta

M:

```
SI VO_DetalleVenta.getCantidad() <= 0 ENTONCES
```

```
    error: "La cantidad ingresada debe ser mayor a 0"
```

```
SINO SI VO_DetalleVenta.getCantidad() > 40 ENTONCES
```

```
    error: "Un detalle de venta no puede superar las 40 unidades"
```

SINO SI

```
!Diccionario(Postres).Member(VO_DetalleVenta.getCodigoPostre())
```

ENTONCES

```
    error: "El código ingresado no está registrado  
    para ningún postre"
```

```
SINO SI !Secuencia(Ventas).Member(VO_DetalleVenta.getNumeroVenta())
```

ENTONCES

```
    error: "El número ingresado no está registrado  
    para ninguna venta"
```

SINO

```
ventaBuscada =
```

```
Secuencia(Ventas).Find(VO_DetalleVenta.getNumeroVenta())
```

```
SI !ventaBuscada.getEnProceso() ENTONCES
```

error: "La venta ya está finalizada, no le puede agregar más postres".

SINO SI ventaBuscada.getTotalUnidades()+
VO_DetalleVenta.getCantidad() > 40 **ENTONCES**

error: "La suma total de cantidades de una venta no puede superar las 40 unidades".

SINO

SI

ventaBuscada.ExisteDetalle(VO_DetalleVenta.getCodigoPostre()) **ENTONCES**

detalleBuscado =

ventaBuscada.GetDetalle(VO_DetalleVenta.getCodigoPostre())

detalleBuscado.setCantidad(detalleBuscado.getCantidad() + VO_DetalleVenta.getCantidad())

ventaBuscada.ModificarDetalleLista(detalleBuscado)

SINO

nuevoDetalle = Crear

DetalleVenta(VO_DetalleVenta.getCodigoPostre(),

VO_DetalleVenta.getCantidad())

ventaBuscada.InsertarDetalle(nuevoDetalle)

FIN SI

Secuencia(Ventas).Modify(ventaBuscada)

FIN SI

FIN SI

6. Eliminar postre de la venta

E: Código que identifica a un postre, la cantidad de unidades a eliminar, y el número que identifica a una venta en proceso

S: Restar dicha cantidad del postre a esa venta, o eliminar el detalle de venta si se resta la cantidad total disponible.

M:

SI VO_DetalleVenta.getCantidad() <= 0 **ENTONCES**

error: "La cantidad ingresada debe ser mayor a 0"

SINO SI

!Diccionario(Postres).Member(VO_DetalleVenta.getCodigoPostre())

ENTONCES

error: "El código ingresado no está registrado para ningún postre"

SINO SI !Secuencia(Ventas).Member(VO_DetalleVenta.getNumeroVenta())

ENTONCES

error: "El número ingresado no está registrado para ninguna venta"

SINO

ventaBuscada =

Secuencia(Ventas).Find(VO_DetalleVenta.getNumeroVenta())

SI !ventaBuscada.getEnProceso() **ENTONCES**

```

        error: "La venta ya está finalizada, no le puede agregar
más postres".
    SINO
        SI
            !ventaBuscada.ExisteDetalle(VO_DetalleVenta.getCodigoPos
tre()) ENTONCES
                error: "No existe un detalle de venta
correspondiente a los datos ingresados"
            SINO
                detalleBuscado =
ventaBuscada.GetDetalle(VO_DetalleVenta.getCodigoPostre())
                SI detalleBuscado.getCantidad() <=
VO_DetalleVenta.getCantidad()
                    ventaBuscada.BorrarDetalle(VO_DetalleVenta.ge
tCodigoPostre())
                SINO
                    detalleBuscado.setCantidad(detalleBuscado.get
Cantidad() - VO_DetalleVenta.getCantidad())
                    ventaBuscada.ModificarDetalle(detalleBuscado)
            FIN SI
        Secuencia(Ventas).Modify(ventaBuscada)
    FIN SI
FIN SI

```

7. Finalizar una venta

E: Número que identifica a una venta en proceso y una indicación de si se confirma o se cancela

S: Venta confirmada o cancelada.

M:

montoTotal = 0

SI !Secuencia(Ventas).Member(VO_FinalizarVenta.getNumero())

ENTONCES

error: "No existe venta registrada con el número
ingresado"

SINO

ventaBuscada =

Secuencia(Ventas).Find(VO_FinalizarVenta.getNumero())

SI ventaBuscada.DetallesEmpty() ||

!VO_FinalizarVenta.getConfirma() **ENTONCES**

Secuencia(Ventas).Borrar(ventaBuscada.getNumero())

SINO SI VO_FinalizarVenta.getConfirma() **ENTONCES**

ventaBuscada.setEnProceso(false)

montoTotal = ventaBuscada.getMonto()

FIN SI

FIN SI

montoVO = Crear VO_Monto(montoTotal)

devolver montoVO

8. Listado de ventas (Todas, en proceso o finalizadas)

E: Indicación T (Total), P (Proceso) o F (Finalizadas)

S: Se despliega en pantalla un listado de las ventas correspondientes a la indicación ingresada

M:

```
indicacion = VO_IndicacionListado.getIndicacion()
nuevaListaVentas = Crear Lista<VO_VentaCompleto>
SI indicacion != "T" && indicacion != "P" && indicacion != "F")
    error: "Debe ingresar una indicación válida (T, P o F)"
SINO SI indicacion == "T" ENTONCES
    nuevaListaVentas = Secuencia(Ventas).ListarVentas()
SINO SI indicacion == "P" ENTONCES
    nuevaListaVentas = Secuencia(Ventas).ListarVentasEnProceso()
SINO SI indicacion == "F" ENTONCES
    nuevaListaVentas =
Secuencia(Ventas).ListarVentasFinalizadas()
FIN SI
Devolver nuevaListaVentas
```

9. Listado de postres de una venta

E: Número que identifica a una venta.

S: Listado de todos los postres agregados a dicha venta.

M:

```
nuevaListaPostres = Crear Lista<VO_PostreCantidad>
SI !Secuencia(Ventas).Member(VO_NumeroVenta.getNumeroVenta())
ENTONCES
    error: "El número ingresado no está registrado
    para ninguna venta"
SINO
    ventaBuscada =
Secuencia(Ventas).Find(VO_NumeroVenta.getNumeroVenta())
    PARA CADA detalle EN ventaBuscada.getDetalles() HACER
        postreBuscado =
Diccionario(Postres).Find(detalle.getCodigo())
        postreCantidadVO = Crear
VO_PostreCantidad(postreBuscado.getCodigo(),
postreBuscado.getNombre(), postreBuscado.getPrecio(),
postreBuscado.getTipo(), detalle.getCantidad())
        nuevaListaPostres.Insertar(postreCantidadVO)
    FIN
FIN SI
Devolver nuevaListaPostres
```

10. Recaudación de un postre en una fecha

E: Código de un postre y una fecha.

S: Monto total de ventas junto con la cantidad total de unidades vendidas de dicho postre en esa fecha.

M:

```
cantidad = 0
```

```
montoTotal = 0
```

```
SI !Diccionario(Postres).Member(VO_PostreFecha.getCodigo())
```

```
ENTONCES
```

```
    error: "El código ingresado no está registrado  
    para ningún postre"
```

```
SINO SI VO_PostreFecha.getFecha() > fechaActual ENTONCES
```

```
    error: "La fecha ingresada no puede ser mayor a la fecha de  
    hoy"
```

```
SINO
```

```
    MIENTRAS !Secuencia(Ventas).EsVacia() &&  
    siguienteVenta.getFecha() <= VO_PostreFecha.getFecha() HACER  
        siguienteVenta = Secuencia(Ventas).Siguiente()
```

```
        SI siguienteVenta.getFecha() ==  
        VO_PostreFecha.getFecha() && !siguienteVenta.getEnProceso() ENTONCES
```

```
            SI
```

```
                siguienteVenta.ExisteDetalle(VO_PostreFecha.getCodigo())
```

```
            ENTONCES
```

```
                detalleBuscado =
```

```
                siguienteVenta.GetDetalle(VO_PostreFecha.getCodigo())
```

```
                postreBuscado =
```

```
                Diccionario(Postres).Find(VO_PostreFecha.getCodigo())
```

```
                cantidad += detalleBuscado.getCantidad()
```

```
                montoTotal += detalleBuscado.getCantidad() *
```

```
                postreBuscado.getPrecio()
```

```
            FIN SI
```

```
        FIN SI
```

```
    FIN
```

```
FIN SI
```

```
datosDevolver = Crear VO_CantidadMonto(cantidad, montoTotal)
```

```
Devolver datosDevolver
```

11. Respaldo de datos

E:

S: Se genera un archivo .bin con los datos del sistema (postres y ventas)

M:

```
SI !Persistencia.Existe(datos.bin) ENTONCES  
    Persistencia.Crear(datos.bin)  
FIN SI
```

```
Persistencia.Abrir(datos.bin)
```

```
PARA CADA postre EN Diccionario(Postres) HACER  
    Persistencia.Guardar(postre, datos.bin)
```

```
FIN
```

```
PARA CADA venta EN Secuencia(Ventas) HACER  
    Persistencia.Guardar(venta, datos.bin)
```

```
FIN
```

```
Persistencia.Cerrar(datos.bin)
```

12. Recuperación de datos

E:

S: Se recuperan los datos del sistema (postres y ventas) a partir de un archivo .bin previamente almacenado

M:

```
SI !Persistencia.Existe(datos.bin) ENTONCES  
    error: "No existen datos para recuperar"  
FIN SI
```

```
Persistencia.Abrir(datos.bin)
```

```
registro = Persistencia.LeerRegistro(datos.bin)
```

```
MIENTRAS registro != null HACER
```

```
    SI registro.type == Postre ENTONCES  
        Diccionario(Postres).Insert(registro)
```

```
    FIN SI
```

```
    SI registro.type == Venta ENTONCES  
        Secuencia(Ventas).InsBack(registro)
```

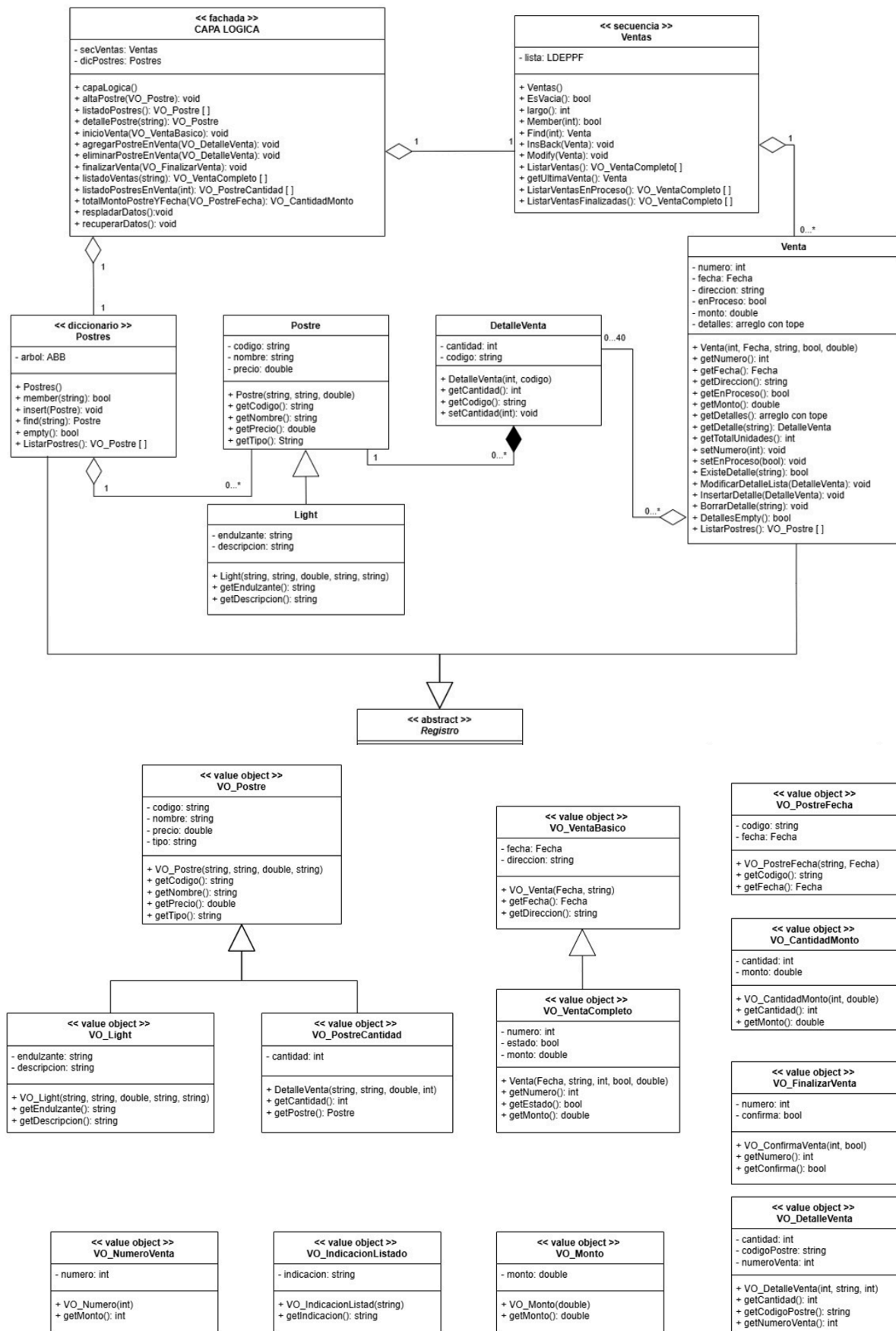
```
    FIN SI
```

```
    registro = Persistencia.LeerRegistro(datos.bin)
```

```
FIN
```

```
Persistencia.Cerrar(datos.bin)
```

Diagrama de clases de implementación



Elección de estructuras de datos

Postres: Árbol Binario de Búsqueda

Se elige un Árbol Binario de Búsqueda para representar el diccionario de postres, ya que el problema no establece una cota máxima de elementos, por lo que se requiere una estructura dinámica que permita inserciones eficientes.

Además, el requerimiento 2 plantea listar todos los postres registrados en base a su código alfanumérico, por lo que resulta conveniente utilizar una estructura que mantenga los elementos ordenados por clave. El Árbol Binario de Búsqueda permite realizar inserciones y búsquedas en tiempo $O(\log n)$ promedio y obtener los elementos ordenados mediante un recorrido en orden.

Ventas: Lista doblemente enlazada con puntero al inicio y al final

Se decide utilizar una lista enlazada debido a que el problema no establece una cota máxima de ventas, por lo que se requiere una estructura dinámica que permita almacenar una cantidad variable de elementos.

Dado que los números de venta son secuenciales y las mismas se registran en orden creciente, las nuevas ventas se insertarán siempre al final de la colección. Por este motivo, mantener una referencia al final de la lista permite realizar inserciones en tiempo $O(1)$ sin necesidad de recorrer la estructura.

Se opta por una lista doblemente enlazada ya que permite recorrer la colección en ambos sentidos. Considerando que varias operaciones requieren localizar ventas recientes o en proceso (requerimientos 5, 6 y 7), disponer de acceso directo al final puede reducir el recorrido promedio necesario para encontrarlas. Si bien la búsqueda continúa siendo $O(n)$, se considera un compromiso adecuado entre simplicidad y eficiencia para el contexto del problema.

Detalles: Arreglo con tope

Se decide utilizar un arreglo con tope para almacenar los detalles de una venta, ya que el problema establece que no se pueden superar las 40 unidades totales de postres por venta, lo que permite definir una cota máxima para la cantidad de elementos. Teniendo en el peor de los casos el arreglo completo con 1 unidad de postre diferente por cada espacio del arreglo.

Las operaciones de búsqueda e inserción presentan complejidad $O(n)$, pero se consideran adecuadas debido a que $n \leq 40$.