|  |  |
| --- | --- |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXdDlW8_x6LtJEiAzA24OoQe2PkRYUKEC7nyPYcRAINi9dk3VDLCn6rn0TVCUMJ6rSiKvKlCbAn-EncXc72v_sncc_3NHMQcUeGCH-Nnn4VYnKv3koeyjQ67DjXqoDHhjmVhiwl3XMur0SdTn4l-_WlnRQ?key=HpZNx3n9EEGtA7UQp1SpjQ** | METODOLOGIA DE PROGRAMACION II |

**Título del informe**: “La Panificadora”

**Materia**: Metodología de Programación II

**Integrantes**:

* Cazon Juan Pablo
* Capararo Gonzalo
* Orellana Maximiliano

**Profesor/a**: Claudia Cappelletti

**Fecha de presentación**: 19/11/2024

**Informe del Proyecto: Panificadora**

**-Índice**

Descripción del problema…………………………………………………..3

Diagrama de clases…………………………………………………………3

Diagrama de secuencia…………………………………………………….4

Especificación de clases……………………………………………………4

Descripción de la metodología……………………………………………..12

Repositorio para el desarrollo del software……………………………….13

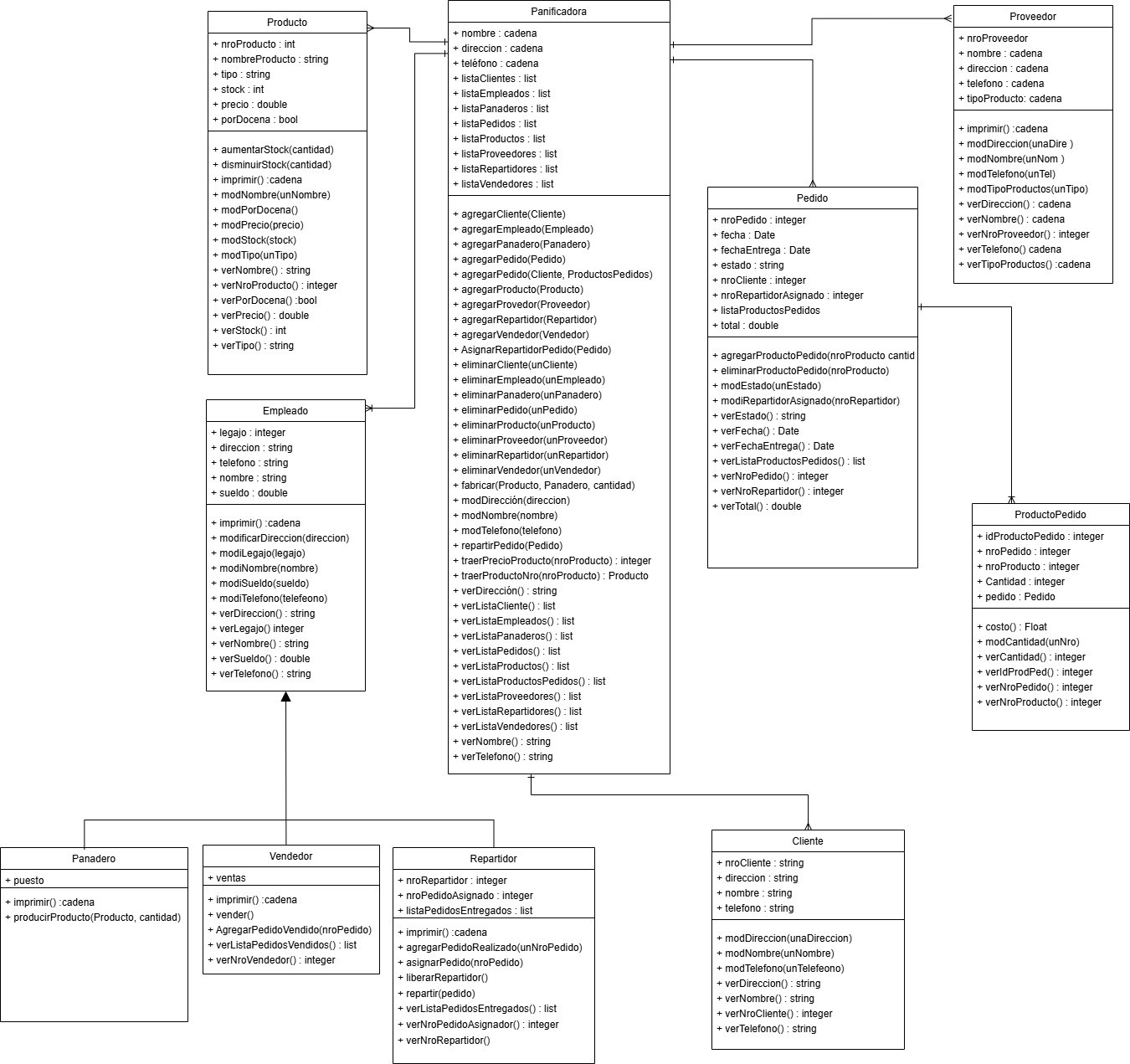
Uso de patrones……………………………………………………………...13

Descripción de algunos Frameworks para POO………………………….14

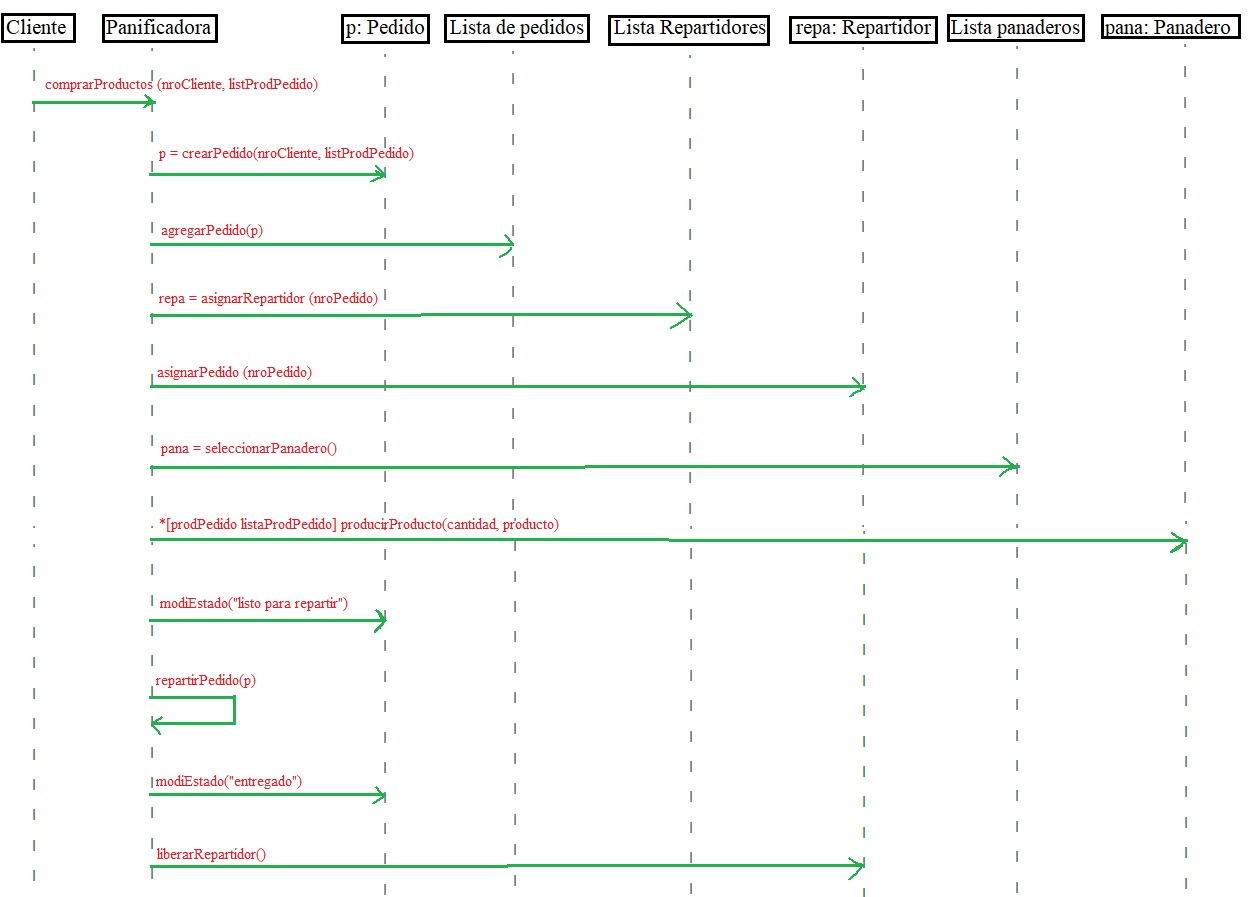
**-Descripción del Problema**

Una panificadora desea optimizar y digitalizar su proceso de gestión y ventas para mejorar la eficiencia operativa y el servicio al cliente. El objetivo del proyecto es desarrollar una aplicación que permita administrar de manera centralizada los productos, pedidos, empleados, clientes y proveedores. Esta digitalización busca reducir errores manuales, agilizar el control de inventarios, mejorar la asignación de pedidos, y facilitar la comunicación con los clientes y repartidores. La solución permitirá llevar un registro detallado y ordenado de todas las operaciones, proporcionando al negocio un mejor control y visibilidad sobre cada aspecto del proceso de ventas y distribución.

**-Diagrama de Clases**

****

**-Diagrama de secuencia**

****

**-Especificación de las clases**

#### Panificadora:

#### Atributos de Instancia

* nombre: Nombre de la panificadora.
* direccion: Dirección de la panificadora.
* telefono: Número de teléfono de la panificadora.
* listaClientes: Colección de clientes registrados.
* listaEmpleados: Colección de empleados de la panificadora.
* listaPanaderos: Colección de panaderos encargados de la producción.
* listaPedidos: Colección de pedidos realizados por los clientes.
* listaProductos: Colección de productos disponibles en la panificadora.
* listaProveedores: Colección de proveedores que suministran insumos.
* listaRepartidores: Colección de repartidores para la entrega de pedidos.
* listaVendedores: Colección de vendedores que gestionan las ventas.

Gestión **de Entidades**

* agregarCliente:: Agrega un cliente a listaClientes.
* agregarEmpleado:: Agrega un empleado a listaEmpleados.
* agregarPanadero:: Agrega un panadero a listaPanaderos.
* agregarPedido:: Agrega un pedido a listaPedidos.
* agregarProducto:: Agrega un producto a listaProductos.
* agregarProveedor:: Agrega un proveedor a listaProveedores.
* agregarRepartidor:: Agrega un repartidor a listaRepartidores.
* agregarVendedor:: Agrega un vendedor a listaVendedores.

Eliminación **de Entidades**

* Métodos como eliminarCliente:, eliminarEmpleado:, eliminarPanadero:, eliminarPedido:, eliminarProducto:, eliminarProveedor:, eliminarRepartidor:, y eliminarVendedor: permiten eliminar entidades de las listas respectivas y muestran un mensaje de confirmación.

Producción **y Gestión de Pedidos**

* comprarProductosNroCliente:listaProd:: Gestiona la compra de productos por parte de un cliente, crea un pedido, produce los productos y actualiza el estado del pedido a "listo para repartir" y luego "entregado".
* fabricarProducto:pana:cantidad:: Ordena la producción de un producto a un panadero especificado.
* asignarRepartidorPedido:: Asigna un repartidor libre a un pedido especificado.
* repartirPedido:: Verifica si el pedido está listo para repartir, asigna un repartidor y realiza la entrega si es posible.
* seleccionarPanadero:: Selecciona un panadero según el tipo de producto.

Consultas **y Modificaciones de Atributos**

* modDireccion:, modNombre:, modTelefono:: Modifican los atributos básicos de la panificadora.
* traerClienteNro:, traerProductoNro:, traerPrecioProducto:: Buscan y retornan un cliente, producto o el precio de un producto, respectivamente.
* Métodos como verDireccion, verListaClientes, verListaEmpleados, verListaPanaderos, etc., permiten acceder a las listas y atributos de la panificadora.

**Métodos de la Clase**

* **Inicialización**
  + crearPanificadoraNom:dire:tel:: Inicializa una instancia de Panificadora con nombre, dirección y teléfono, y crea colecciones vacías para las listas mencionadas.

### Cliente

* **Atributos de Instancia**:
  + nroCliente: Identificador único del cliente.
  + nombre: Nombre del cliente.
  + direccion: Dirección del cliente.
  + telefono: Teléfono del cliente.
* **Atributos de Clase**:
  + idCliente: Se usa para generar un identificador único para cada instancia de Cliente.

### Métodos de la Instacia Cliente

* **imprimir**: Devuelve una cadena de texto con los datos del cliente en un formato específico.
* **iniClienteNom:dire:tel:**: Inicializa una instancia de Cliente asignando valores a los atributos nombre, direccion, y telefono. También asigna un número único a nroCliente usando el método nextNroCliente.
* **Modificadores**:
  + modDireccion:: Cambia la dirección del cliente.
  + modNombre:: Cambia el nombre del cliente.
  + modTelefono:: Cambia el teléfono del cliente.
* **printOn:**: Imprime los datos del cliente en un Stream de texto. Esta es una forma alternativa de representar los datos del cliente.
* **Accesores**:
  + verDireccion: Retorna la dirección del cliente.
  + verNombre: Retorna el nombre del cliente.
  + verNroCliente: Retorna el número de cliente.
  + verTelefono: Retorna el teléfono del cliente.

### Métodos de Clase

1. **crearClienteNom:dire:tel:**: Crea y devuelve una instancia de Cliente inicializada con los valores de nombre, dirección y teléfono proporcionados.
2. **initialize**: Inicializa el atributo idCliente en 0.
3. **nextNroCliente**: Genera un número único para cada cliente, incrementando el valor de idCliente en uno cada vez que se llama, y lo asigna a nroCliente en una instancia nueva de Cliente.

### Empleado

* **Atributos de Instancia**:
  + legajo: Número de legajo del empleado.
  + nombre: Nombre del empleado.
  + direccion: Dirección del empleado.
  + telefono: Teléfono del empleado.
  + sueldo: Sueldo del empleado.

### Métodos de instancia Empleado

* **imprimir**: Retorna una cadena con los datos del empleado en un formato específico.
* **iniEmpleadoLegajo:nom:dire:tel:**: Inicializa una instancia de Empleado asignando valores a los atributos legajo, nombre, direccion, y telefono.
* **Modificadores**:
  + modDireccion:: Cambia la dirección del empleado.
  + modLegajo:: Cambia el legajo del empleado.
  + modNombre:: Cambia el nombre del empleado.
  + modSueldo:: Cambia el sueldo del empleado.
  + modTelefono:: Cambia el teléfono del empleado.
* **printOn:**: Imprime los datos del empleado en un Stream de texto. Es otra forma de representar los datos del empleado.
* **Accesores**:
  + verDireccion: Retorna la dirección del empleado.
  + verLegajo: Retorna el legajo del empleado.
  + verNombre: Retorna el nombre del empleado.
  + verSueldo: Retorna el sueldo del empleado.
  + verTelefono: Retorna el teléfono del empleado.

### Métodos de Clase

* **crearEmpleadoLegajo:nom:dire:tel:**: Crea y retorna una instancia de Empleado inicializada con los valores de legajo, nombre, dirección y teléfono proporcionados.

### Panadero (subclase de Empleado):

* **Variables de instancia**:
  + puesto: Una variable que guarda el puesto del panadero (por ejemplo, su rol dentro de la panadería).
* **Métodos de la instancia**:
  + **imprimir**: Retorna una cadena que incluye el puesto del panadero, llamando a un método imprimir de la clase superior (Empleado).
  + **iniPanaderoLegajo:nom:dire:tel:puesto:**: Inicializa una instancia de la clase Panadero. Este método también invoca el método iniEmpleadoLegajo de la clase superior (Empleado), estableciendo las propiedades del panadero, como el legajo, nombre, dirección, teléfono y el puesto. Además, establece el sueldo inicial del panadero en 800,000.
  + **printOn:**: Este método imprime información sobre el panadero en un flujo de salida (aStream), mostrando el puesto del panadero y llamando al método printOn de la clase superior para imprimir la información adicional del empleado.
  + **producirProducto:cant:**: Este método permite al panadero producir una cantidad de un producto específico. Si el producto se maneja por docena, indica que la unidad es "Docenas". Aumenta el stock del producto y muestra un mensaje en la consola con la cantidad producida y el stock disponible.
  + **verPuesto**: Retorna el puesto del panadero.
* **Métodos de clase**:
  + **crearPanaderoLegajo:nom:dire:tel:puesto:**: Este es un método de clase que crea y retorna una nueva instancia de Panadero, inicializándola con el legajo, nombre, dirección, teléfono y puesto proporcionados. Utiliza el método iniPanaderoLegajo para la inicialización.

### Pedido:

* **Variables de instancia**:
  + nroPedido: El número único del pedido.
  + fecha: La fecha en la que se realizó el pedido.
  + fechaEntrega: La fecha en que se debe entregar el pedido.
  + estado: El estado actual del pedido (por ejemplo, "en preparación").
  + nroCliente: El número del cliente que realizó el pedido.
  + nroRepartidorAsignado: El número del repartidor asignado para entregar el pedido.
  + listaProductosPedidos: Una colección de productos pedidos, representada por objetos de la clase ProductoPedido.
  + total: El costo total del pedido, calculado como la suma de los costos de los productos en listaProductosPedidos.
* **Métodos de la instancia**:
  + **agregarProductoPedido:cantidad:**: Este método agrega un producto al pedido. Si el producto ya está en la lista, se aumenta su cantidad; si no, se crea un nuevo objeto ProductoPedido y se agrega a la lista.
  + **agregarProductoPedidoProd:cantidad:**: Similar al método anterior, pero este agrega un producto directamente desde un objeto Producto y actualiza el total del pedido después de agregarlo.
  + **eliminarProductoPedido:**: Elimina un producto de la lista de productos pedidos en base al número de producto proporcionado. Si el producto no se encuentra, muestra un mensaje en la consola.
  + **imprimir**: Retorna una cadena con la información del pedido, incluyendo el número de pedido, fechas, número de cliente, número de repartidor asignado y el total.
  + **imprimirTrpt**: Imprime información detallada del pedido en el Transcript, incluyendo todos los datos del pedido.
  + **iniPedidoNroCli:fechaEnt:listPP:**: Inicializa un objeto Pedido con el número de cliente, la fecha de entrega y una lista de productos pedidos. Calcula el total y asigna los productos a este pedido.
  + **modEstado:**: Modifica el estado del pedido.
  + **modiRepartidorAsignado:**: Asigna un repartidor al pedido.
  + **verEstado**: Retorna el estado del pedido.
  + **verFechaEntrega**: Retorna la fecha de entrega del pedido.
  + **verListaProductosPedidos**: Retorna una cadena con los detalles de los productos en el pedido.
  + **verNroCliente**: Retorna el número del cliente que realizó el pedido.
  + **verNroPedido**: Retorna el número de pedido.
  + **verRepartidor**: Retorna el número del repartidor asignado.
  + **verTotal**: Calcula y retorna el total del pedido, sumando los costos de los productos.
* **Métodos de clase**:
  + **crearPedidoNroClien:listProdPed:**: Crea un pedido con el número de cliente y una lista de productos pedidos. Establece una fecha de entrega para el día siguiente.
  + **crearPedidoNroCliente:fechaEntrega:listProdPed:**: Similar al método anterior, pero permite especificar la fecha de entrega.
  + **initialize**: Inicializa la variable nroPedidoStatic en 1, que es usada para generar números de pedido únicos.
  + **nextId**: Genera y retorna un nuevo número de pedido único utilizando la variable nroPedidoStatic.

### Características generales de la clase Pedido:

* **Gestión de productos**: Permite agregar, eliminar y modificar productos en el pedido mediante métodos como agregarProductoPedido, agregarProductoPedidoProd, y eliminarProductoPedido.
* **Cálculo del total**: Calcula el costo total del pedido sumando los costos de los productos en listaProductosPedidos.
* **Estado y repartidor**: Permite modificar y ver el estado del pedido y asignar un repartidor.
* **Inicialización**: Los métodos de clase como crearPedidoNroClien:listProdPed: y crearPedidoNroCliente:fechaEntrega:listProdPed: permiten crear un pedido con los datos necesarios.
* **Métodos de impresión**: Tiene métodos para imprimir la información del pedido, tanto como una cadena de texto (imprimir) como en el Transcript (imprimirTrpt).
* **Producto**

**Atributos de instancia:**

* **nroProducto**: Número identificador único del producto.
* **nombreProducto**: Nombre del producto.
* **tipo**: Tipo de producto, que puede indicar la categoría o familia a la que pertenece (e.g., 'Factura').
* **stock**: Cantidad de unidades disponibles del producto en el inventario.
* **precio**: Precio por unidad del producto.
* **porDocena**: Booleano que indica si el producto se vende por docena (true) o por unidad (false).

**Atributos de clase:**

* **idProd**: Variable de clase utilizada para generar un identificador único incrementado automáticamente para cada producto.

**Métodos de instancia:**

* **aumentarStock: cantidad**  
  Incrementa el stock en la cantidad especificada.
* **disminuirStock: cantidad**  
  Reduce el stock en la cantidad especificada, sin validar si la cantidad resultante es negativa.
* **imprimir**  
  Retorna una cadena con los datos principales del producto, incluidos nroProducto, nombreProducto, tipo, stock, precio, y porDocena.
* **iniProductoNombre: unNom tip: unTipo prec: unPrecio**  
  Inicializa una instancia de Producto con un nombre (unNom), tipo (unTipo), y precio (unPrecio). Asigna un nuevo nroProducto único y ajusta porDocena si el tipo es 'Factura'.
* **modNombre: unNombre**  
  Cambia el nombreProducto del producto a unNombre.
* **modPorDocena**  
  Alterna el estado de porDocena entre true y false.
* **modPrecio: unPrecio**  
  Cambia el precio del producto a unPrecio.
* **modStock: unStock**  
  Modifica el stock del producto. Si unStock es menor a 0, muestra un mensaje de error en el Transcript.
* **modTipo: unTipo**  
  Cambia el tipo del producto a unTipo.
* **printOn: aStream**  
  Escribe los datos del producto en un flujo de texto aStream con un formato similar al método imprimir.
* **verNombre**  
  Retorna el nombreProducto.
* **verNroProducto**  
  Retorna el nroProducto.
* **verPorDocena**  
  Retorna el valor booleano de porDocena, indicando si el producto se vende por docena.
* **verPrecio**  
  Retorna el precio del producto.
* **verStock**  
  Retorna el stock.
* **verTipo**  
  Retorna el tipo del producto.

**Métodos de clase:**

* **crearProductoNombre: unNom tip: unTipo prec: unPrecio**  
  Crea e inicializa una nueva instancia de Producto con el nombre, tipo y precio especificados, retornando la instancia creada.
* **initialize**  
  Inicializa la variable de clase idProd en 0.
* **nextId**  
  Incrementa idProd en 1 y lo retorna, generando un identificador único para una nueva instancia.
* **ProductoPedido**

**Atributos de instancia:**

* **idProductoPedido**: Número de identificación único del producto en el pedido.
* **nroProducto**: Número identificador del producto asociado al pedido.
* **cantidad**: Cantidad del producto solicitado en el pedido.
* **nroPedido**: Número de identificación del pedido al cual pertenece este producto pedido.
* **pedido**: Referencia al objeto Pedido al que pertenece este ProductoPedido.
* **costoUnitario**: Precio unitario del producto en el pedido.

**Atributos de clase:**

* **nroProductoPedido**: Variable de clase utilizada para asignar un identificador único y secuencial a cada instancia de ProductoPedido.

**Métodos de instancia:**

* **imprimir**  
  Retorna una cadena de texto con la información de ProductoPedido, incluyendo idProductoPedido, nroProducto, cantidad, y el costo total.
* **iniProductoPedidoNroProd: unNroProduct cantidad: cant costoUnitario: cost**  
  Inicializa una instancia de ProductoPedido con un número de producto (unNroProduct), cantidad (cant), y costo unitario (cost).
* **iniProductoPedidoUnProd: unProducto cantidad: cant**  
  Inicializa una instancia de ProductoPedido tomando un objeto Producto (unProducto), con una cantidad (cant), y asigna el precio unitario a partir del producto.
* **iniProductoPedidoUnProd: unProducto Pedido: unPedido cantidad: cant**  
  Inicializa una instancia de ProductoPedido con un objeto Producto (unProducto), un objeto Pedido (unPedido), y la cantidad (cant).
* **modCantidad: unNro**  
  Modifica la cantidad de producto pedida a unNro.
* **modCostoUnitario: unNro**  
  Modifica el costo unitario del producto en el pedido a unNro.
* **modNroPedido: unNroPedido**  
  Cambia el número de pedido (nroPedido) a unNroPedido.
* **setPedido: unPedido**  
  Asocia este ProductoPedido con el objeto Pedido unPedido, actualizando nroPedido con el número de este pedido.
* **sumarCantidad: unaCant**  
  Incrementa la cantidad en el valor unaCant.
* **verCantidad**  
  Retorna el valor de cantidad.
* **verCosto**  
  Calcula y retorna el costo total del producto en el pedido, multiplicando cantidad por costoUnitario.
* **verCostoUnitario**  
  Retorna el valor de costoUnitario.
* **verIdPP**  
  Retorna el idProductoPedido.
* **verNrodelPedido**  
  Retorna el nroPedido.
* **verNroProducto**  
  Retorna el nroProducto.

**Métodos de clase:**

* **crearProductoPedido: unProducto cantidad: cant**  
  Crea una instancia de ProductoPedido usando un objeto Producto (unProducto) y una cantidad (cant), inicializándola con iniProductoPedidoUnProd:cantidad:.
* **crearProductoPedido: unProducto pedido: unPedido cantidad: cant**  
  Crea una instancia de ProductoPedido con un Producto (unProducto), un Pedido (unPedido), y una cantidad (cant), inicializándola con iniProductoPedidoUnProd:Pedido:cantidad:.
* **crearProductoPedidoNroProd: nroProd cantidad: cant costoUnitario: costo**  
  Crea una instancia de ProductoPedido inicializada con un número de producto (nroProd), cantidad (cant), y costo unitario (costo).
* **initialize**  
  Inicializa la variable de clase nroProductoPedido en 0 para comenzar el conteo de identificadores únicos.
* **nextNroPP**  
  Incrementa nroProductoPedido en 1 y lo retorna, proporcionando un identificador único para cada nueva instancia de ProductoPedido.
* **Proveedor**

**Atributos de instancia:**

* **nroProveedor**: Número identificador único del proveedor.
* **direccion**: Dirección del proveedor.
* **telefono**: Número de teléfono del proveedor.
* **tipoProductos**: Tipo de productos que provee el proveedor.
* **nombre**: Nombre del proveedor.

**Atributos de clase:**

* **idProveedor**: Variable de clase utilizada para asignar un identificador único y secuencial a cada instancia de Proveedor.

**Métodos de instancia:**

* **imprimir**  
  Retorna una cadena de texto con los datos de Proveedor, incluyendo nroProveedor, nombre, direccion, telefono, y tipoProductos.
* **iniProveedorNombre: unNom dir: unaDir tel: unTel tip: unTipo**  
  Inicializa una instancia de Proveedor con los valores para nombre (unNom), direccion (unaDir), telefono (unTel), y tipoProductos (unTipo).
* **modDireccion: unaDire**  
  Modifica la dirección del proveedor con unaDire.
* **modNombre: unNom**  
  Cambia el nombre del proveedor a unNom.
* **modTelefono: unTel**  
  Actualiza el número de teléfono del proveedor con unTel.
* **modTipoProductos: unTipo**  
  Modifica el tipo de productos que el proveedor ofrece, con unTipo.
* **verDireccion**  
  Retorna la dirección del proveedor.
* **verNombre**  
  Retorna el nombre del proveedor.
* **verNroProveedor**  
  Retorna el número identificador del proveedor.
* **verTelefono**  
  Retorna el número de teléfono del proveedor.
* **verTipoProductos**  
  Retorna el tipo de productos que ofrece el proveedor.

**Métodos de clase:**

* **crearProveedorNombre: unNom dir: unaDir tel: unTel tip: unTipo**  
  Crea una instancia de Proveedor inicializada con nombre, direccion, telefono, y tipoProductos.
* **initialize**  
  Inicializa la variable de clase idProveedor en 1, comenzando el conteo de identificadores para nuevos proveedores.
* **nextId**  
  Retorna un identificador único para una nueva instancia de Proveedor, incrementando idProveedor en cada llamada.
* **Repartidor** (subclase de Empleado)

**Atributos de instancia:**

* **nroRepartidor**: Número de identificación único del repartidor.
* **listaPedidosEntregados**: Lista de pedidos que el repartidor ha entregado.
* **nroPedidoAsignado**: Número del pedido asignado al repartidor.

**Atributos de clase:**

* **UnNuevoNroRepartidor**: Variable de clase utilizada para generar un número único y secuencial para cada nuevo Repartidor.

**Métodos de instancia:**

* **agregarPedidoRealizado: unNroPedido**  
  Agrega el número de un pedido entregado a la lista de listaPedidosEntregados.
* **asignarPedido: nroPedido**  
  Asigna un nuevo número de pedido para que el repartidor lo entregue.
* **imprimir**  
  Retorna una cadena con el número del repartidor, invocando además el método imprimir de la clase Empleado.
* **iniRepartidorLegajo: unLegajo nom: unNombre dire: unaDire tel: unTel**  
  Inicializa una instancia de Repartidor configurando su legajo, nombre, dirección y teléfono, además de asignar un nroRepartidor único, un sueldo predeterminado y una lista vacía de listaPedidosEntregados.
* **liberarRepartidor**  
  Agrega el pedido actual a listaPedidosEntregados y cambia nroPedidoAsignado a 0, dejando al repartidor libre para un nuevo pedido.
* **printOn: aStream**  
  Imprime la información del repartidor en el flujo aStream, incluyendo su número.
* **repartir: pedido**  
  Asigna el pedido al repartidor y luego libera al repartidor una vez que el pedido se entrega, mostrando la operación en el Transcript.
* **verListaPedidosEntregados**  
  Retorna la lista de pedidos entregados por el repartidor.
* **verNroPedidoAsignado**  
  Retorna el número del pedido actualmente asignado al repartidor.
* **verNroRepartidor**  
  Retorna el número de identificación del repartidor.

**Métodos de clase:**

* **crearRepartidorLegajo: unLegajo nom: unNom dire: unaDire tel: unTel**  
  Crea e inicializa una nueva instancia de Repartidor con el legajo, nombre, dirección y teléfono dados.
* **initialize**  
  Inicializa la variable de clase UnNuevoNroRepartidor en 0.
* **nuevoNroRepartidor**  
  Retorna un número único para un nuevo Repartidor, incrementando la variable de clase UnNuevoNroRepartidor en cada llamada.
* **Vendedor** (subclase de Empleado)

**Atributos de instancia:**

* **nroVendedor**: Número de identificación único del vendedor.
* **pedidosVendidos**: Lista de pedidos vendidos por el vendedor.

**Atributos de clase:**

* **UnNuevoNroVendedor**: Variable de clase utilizada para generar un número secuencial único para cada nuevo Vendedor.
* **idVendedor**: Variable de clase utilizada para asignar identificadores únicos a las instancias de Vendedor.

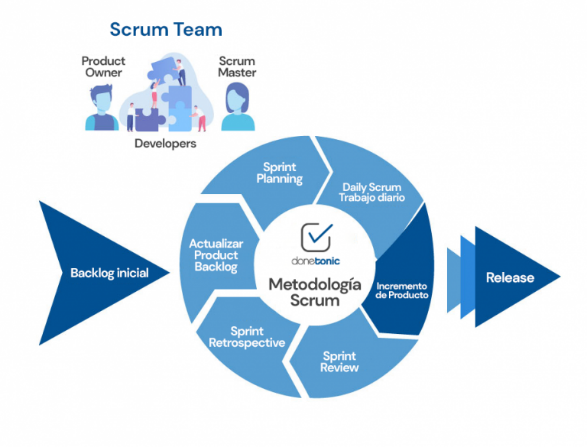
**Métodos de instancia:**

* **agregarPedidoVendido: unNroPedido**  
  Agrega el número de un pedido a la lista de pedidosVendidos.
* **imprimir**  
  Retorna una cadena con el número del vendedor, invocando además el método imprimir de la clase Empleado.
* **iniVendedorLegajo: unLegajo nom: unNombre dire: unaDire tel: unTel**  
  Inicializa una instancia de Vendedor con el legajo, nombre, dirección y teléfono dados. Asigna un número único a nroVendedor y establece un sueldo base de 600000.
* **printOn: aStream**  
  Imprime la información del vendedor en el flujo aStream, incluyendo su número.
* **verNroVendedor**  
  Retorna el número de identificación del vendedor.
* **verPedidosVendidos**  
  Retorna la lista de pedidos vendidos por el vendedor.

**Métodos de clase:**

* **crearVendedorLegajo: unLegajo nom: unNom dire: unaDire tel: unTel**  
  Crea e inicializa una nueva instancia de Vendedor con los datos proporcionados.
* **initialize**  
  Inicializa la variable de clase idVendedor en 0.
* **nextNroVendedor**  
  Retorna un número único para un nuevo Vendedor, incrementando idVendedor en cada llamada.
* **nuevoNumeroVendedor**  
  Genera un nuevo número para nroVendedor, incrementando UnNuevoNroVendedor en cada llamada, y asegura que esté inicializado.

**-Descripción de la metodología**

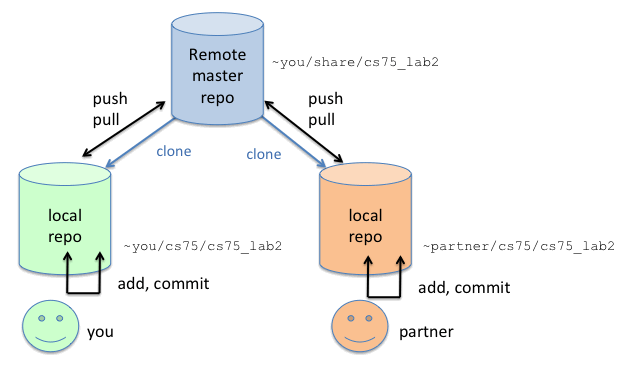
****

Para el desarrollo de este proyecto utilizamos varias prácticas de la metodología ágil **Scrum**. Entre ellas, implementamos **reuniones diarias de sincronización**, donde los integrantes del grupo se reunían para revisar el progreso y establecer objetivos. Estas reuniones, conocidas como "daily stand-ups" en Scrum, permitieron al equipo coordinarse y alinear sus tareas diarias.

Además, estructuramos el trabajo en **sprints** o ciclos cortos de desarrollo con objetivos específicos, lo cual facilitó la entrega de incrementos funcionales del proyecto en plazos definidos. Al finalizar cada sprint, realizábamos una **revisión de sprint** para evaluar el avance y recibir retroalimentación, identificando mejoras para el siguiente ciclo.

También implementamos la práctica de **retrospectivas** al cierre de cada sprint. En estas retrospectivas, reflexionábamos sobre los puntos positivos y las áreas de mejora en el trabajo en equipo, con el objetivo de optimizar la colaboración y la eficiencia del grupo en futuros sprints. Estas características de Scrum nos permitieron mantener un flujo constante de comunicación, ajustar nuestro enfoque según las necesidades del proyecto y asegurar que todos los integrantes estuvieran comprometidos con los objetivos establecidos.

**-Repositorio para el desarrollo del software**

****

Para el desarrollo de software colaborativo, el uso de repositorios Git es altamente recomendado, ya que permite el control de versiones de manera eficiente y facilita el trabajo en equipo. En nuestro caso, empleamos un repositorio compartido en Git, en el cual cada integrante del equipo tenía su propia rama de desarrollo. Esto ayudaba a que cada uno pudiera realizar cambios sin afectar el trabajo de los demás hasta que las modificaciones estuvieran listas para integrarse.

Cada integrante realizaba merges hacia la rama principal (master) solo cuando sus cambios habían sido revisados y aprobados, lo cual es una buena práctica en entornos colaborativos para evitar conflictos y asegurar la calidad del código. Este enfoque fomenta la colaboración continua y el versionado seguro del proyecto, permitiendo un desarrollo organizado y controlado.

**-Uso de patrones**

**Singleton para la clase Panificadora**  
 En este caso, el patrón Singleton asegura que exista solo una instancia de la clase Panificadora durante toda la ejecución del programa. Esto es útil porque la panificadora es única y centraliza la gestión de clientes, empleados, y pedidos. Utilizando Singleton, cualquier intento de crear una nueva instancia de Panificadora devolvería la misma instancia existente, facilitando un punto de acceso global y manteniendo la integridad de la información centralizada.

**Factory para la creación de Pedidos**

Un patrón Factory sería ideal para la creación de objetos Pedido. Como cada pedido puede tener diferentes características (como lista de productos, cantidades, y detalles de cliente), una Factory permite instanciar pedidos de manera controlada y con la configuración adecuada según el tipo de pedido (por ejemplo, pedidos en tienda, pedidos a domicilio). De esta manera, el patrón Factory ayuda a manejar la lógica de creación y facilita la personalización y extensión para otros tipos de pedidos en el futuro.

**Observer para actualizar a los Repartidores sobre el estado de los Pedidos**  
 En este contexto, el patrón Observer sería útil para gestionar la asignación y actualización de los Repartidores en relación con los Pedidos. Cuando un pedido cambia de estado (por ejemplo, de "en preparación" a "listo para entregar"), el Repartidor correspondiente podría recibir notificaciones automáticas y actualizar su lista de pedidos asignados. Este patrón asegura que cualquier cambio en el estado del pedido se comunique de inmediato a los objetos interesados (repartidores), mejorando la eficiencia de las entregas y el flujo de trabajo.

**-Descripción de algunos Frameworks para POO**

En la elaboración de software un Framework es un entorno de trabajo para el desarrollo de aplicaciones que sirve para hacer más eficiente y recursivo el trabajo del programador logrando que el código sea más proclive a evolucionar y más sencillo de mantener. El presente trabajo se implemento en Smalltalk un lenguaje de programación orientado a objetos (POO), si bien actualmente no es de los más utilizados a ejercido una fuerte influencia en los otros lenguajes que soportan POO. Por ejemplo en Java y C#, aunque estos si distinguen entre objetos y tipos primitivos a diferencia de Smalltalk. A continuación una breve descripción de dos Frameworks muy conocidos, uno fue pensado para Java y el otro para C#.

El **Framework Spring** es un marco popular para Java que simplifica el desarrollo web, la inyección de dependencias y las pruebas. Proporciona un modelo integral de programación y configuración para aplicaciones basadas en Java en cualquier tipo de plataforma de implementación. Se podría decir que Spring se un soporte de infraestructura a nivel de aplicación porque se centra en la “plomería” de las aplicaciones permitiendo que los desarrolladores se focalicen en la lógica del negocio a nivel aplicación sin vínculos innecesarios con entornos de implementación específicos. Spring ofrece una inyección de dependencias que permite a los objetos definir sus propias dependencias que el contenedor de Spring luego inyecta en ellos. Es gratuito y de código abierto.



El **Framework .Net** es un marco que proporciona un entorno de tiempo de ejecución común, una biblioteca de clases y un conjunto de herramientas para C# y otros idiomas. También admite el desarrollo web, las aplicaciones multiplataforma y los servicios en la nube. Se lo podría definir como el conjunto de elementos necesarios para la creación de software bajo la matriz Microsoft, que permite a los desarrolladores crear aplicaciones (Windows, Web, Servicios, Mobile, etc), teniendo una experiencia consistente entre todos los tipos y con un entorno que permita su ejecución.

Dichos elementos son: Herramientas de ejecución de aplicaciones (CLR) +  Librerías de uso común (BCL -> FCL) + Entorno de desarrollo (Visual Studio) + Lenguajes (C#, VB.Net, F#).

