Ejercicio 28 - Lavadero

Kevin Manuel Gómez Rojas

Universidad de Cartagena, Campus Piedra Bolívar

Nota del autor

**Ingeniería de software semestre 2**

**Profesor**: Jhon Carlos Arrieta Arrieta

**Asignatura**: Programación Orientada a Objetos

**Primera Fase**

**Fecha de entrega:** 22 de abril del 2025

Resumen

Desarrollar un programa en Java usando POO:   
Analizando el documento o la situación presentada en la actividad correspondiente, debemos desarrollar una sección de pasos y requisitos para realizar la entrega de dicho proyecto.

Estos requisitos o pasos son los siguientes:

* Analizar el problema.
* diseñar clases, diagramas, documentación y material probatorio de la realización de la construcción del proyecto.

Palabras clave: Poo, clases, diagramas, UML, requerimientos, modelar, código, dominio, entidades, métodos, propiedades, constructor, relaciones, ejecución.

Tabla de contenido

[Resumen 2](#_Toc196248615)

[Introducción 4](#_Toc196248616)

[Objetivos 5](#_Toc196248617)

[Objetivos Generales 5](#_Toc196248618)

[Objetivos Específicos 5](#_Toc196248619)

[Objetivos transversales 6](#_Toc196248620)

[Justificación 7](#_Toc196248621)

[Desarrollo 8](#_Toc196248622)

Introducción

La Programación Orientada a Objetos (POO) es un paradigma fundamental en el desarrollo de software, basado en la organización del código mediante clases y objetos que interactúan entre sí. Esta actividad busca aplicar estos conceptos mediante el análisis, diseño e implementación de un programa en Java, partiendo de un enunciado específico. El proceso incluye la identificación de clases, sus propiedades y métodos, la modelación con diagramas UML y la codificación en un entorno estructurado (como NetBeans), respetando principios como encapsulamiento, herencia y relaciones entre objetos. Además, se enfatiza la originalidad del trabajo, evitando prácticas como el plagio o la entrega de soluciones idénticas entre estudiantes. El resultado final debe reflejar un entendimiento claro de la POO, demostrado tanto en la documentación técnica como en la funcionalidad del programa desarrollado.

# Objetivos

## Objetivos Generales

1. **Aplicar los principios de la Programación Orientada a Objetos (POO)** en el desarrollo de un programa en Java, utilizando clases, objetos, herencia, encapsulamiento y relaciones entre entidades.
2. **Modelar un sistema real** mediante diagramas UML, identificando las clases necesarias, sus atributos, métodos y las relaciones entre ellas.
3. **Desarrollar habilidades de análisis y diseño** al descomponer un problema en componentes manejables y traducirlos a estructuras de código organizadas.
4. **Fomentar la originalidad y el pensamiento crítico**, evitando el plagio y garantizando que la solución sea producto del análisis y la lógica propia.
5. **Practicar la implementación de código limpio y estructurado**, siguiendo convenciones de nomenclatura, paquetes y jerarquías de clases.

## **Objetivos Específicos**

1. **Identificar requerimientos funcionales** a partir de un enunciado dado, extrayendo las clases y comportamientos clave del sistema.
2. **Definir propiedades y métodos** para cada clase, asegurando un correcto encapsulamiento de los datos.
3. **Establecer relaciones entre clases** (asociación, herencia, composición, etc.) y representarlas en un diagrama de clases UML.
4. **Implementar constructores** (por defecto y parametrizados) para la creación de objetos con estados iniciales definidos.
5. **Sobrescribir el método toString()** para mostrar de manera clara la información de los objetos.
6. **Utilizar métodos set y get** para modificar y acceder a los atributos de las clases, respetando el principio de encapsulamiento.
7. **Crear instancias de objetos** en el método main, demostrando el uso de constructores y la asignación de valores a sus propiedades.
8. **Validar el funcionamiento del programa** ejecutando pruebas que muestren la interacción entre objetos y la correcta salida de datos.
9. **Documentar el proceso** en un informe técnico, justificando las decisiones de diseño y la estructura del código.
10. **Gestionar la entrega técnica**, siguiendo normas de formato, nomenclatura y organización de archivos (ZIP, paquetes, etc.).

## Objetivos transversales

* 1. Fortalecer el aprendizaje autónomo mediante la investigación de conceptos POO y herramientas como UML o NetBeans.
  2. Desarrollar competencias en la detección y corrección de errores durante la compilación y ejecución del código.
  3. Promover la ética académica, entregando un trabajo original y citando correctamente las fuentes de consulta.
  4. Prepararse para evaluaciones futuras, ya que la actividad integra conocimientos clave evaluables en el ámbito profesional.

# Justificación

La presente actividad tiene como propósito fundamental consolidar las competencias en Programación Orientada a Objetos (POO), un pilar indispensable en el desarrollo de software profesional. Su relevancia radica en que este paradigma es la base estructural de sistemas complejos, frameworks modernos y arquitecturas escalables utilizadas en la industria tecnológica actual.

Al implementar esta actividad, se busca:

* Internalizar buenas prácticas de desarrollo mediante el uso de principios POO como encapsulamiento, herencia y abstracción, que son estándares en entornos laborales reales.
* Desarrollar pensamiento algorítmico y de diseño al modelar soluciones mediante diagramas UML, traduciendo problemas conceptuales a estructuras de código organizadas y mantenibles.
* Fortalecer su capacidad de autogestión al investigar, analizar e implementar soluciones originales, simulando el flujo de trabajo de un proyecto de desarrollo real.

La exigencia de originalidad en el código no es solo formalismo académico, sino un reflejo de las necesidades del sector tecnológico, donde el plagio tiene consecuencias éticas, legales y profesionales. Esta actividad, al prohibir prácticas como el maquillaje de código ajeno, nos prepara para:

* Defender nuestras soluciones técnicamente
* Argumentar decisiones de diseño
* Desarrollar criterio propio como programado

# Desarrollo

Al realizar la lectura del ejercicio 28, se sacaron las siguientes clases:

1. Empresa
2. Persona
3. TipoID (enum)
4. Empleado
5. Supervisor
6. Contrato
7. Cargo (enum)
8. Cliente
9. Vehiculo
10. Proveedor
11. InformacionPago
12. MetodoPago (enum)
13. CuentaBancaria
14. Catalogo
15. Categoria
16. Producto
17. Servicio
18. Bodega
19. AreaDeTrabajo
20. Cubiculo
21. Documento (abstracta)
22. Cotizacion
23. DetalleCotizacion
24. Pedido
25. DetallePedido
26. NotaCorrecion
27. DetalleNotaCorrecion
28. ComprobantePago
29. OrdenVenta
30. DetalleVentaProducto
31. DetalleVentaServicio

Descripción de las Clases

1. Empresa

**DESCRIPCIÓN:**  
Representa la entidad principal del sistema que agrupa todos los recursos, empleados, clientes y operaciones comerciales.

**PROPIEDADES:**

* String nombre
* String codigo
* InformacionPago informacionDePago
* List<Empleado> empleados
* List<Cliente> clientes
* List<Proveedor> proveedores
* List<Bodega> bodegas
* Catalogo catalogo
* List<cotizacion> cotizaciones
* List<ComprobantePago> comprobantes
* List<NotaCorrecion> notasCorrecion
* List<AreaDeTrabajo> areasDeTrabajo

**MÉTODOS:**  
*(No se especifican métodos en el diagrama)*

2. Persona

**DESCRIPCIÓN:**  
Clase base que contiene los atributos comunes a todas las personas en el sistema (empleados, clientes).

**PROPIEDADES:**

* String Nombre
* TipoID TipoID
* String Identificacion
* Date FechaNacimiento
* String Correo
* String Telefono
* String Direccion

**MÉTODOS:**  
*(No se especifican métodos en el diagrama)*

3. Empleado

**DESCRIPCIÓN:**  
Representa a los trabajadores de la empresa, hereda de Persona y añade información contractual.

**PROPIEDADES:**

* Contrato contrato (heredado de Persona)

**MÉTODOS:**  
*(No se especifican métodos en el diagrama)*

4. Supervisor

**DESCRIPCIÓN:**  
Tipo especial de empleado que gestiona turnos de clientes y horarios de otros empleados.

**PROPIEDADES:**

* ArrayList<Cliente> turnos

**MÉTODOS:**

* Supervisor(nombre: String, tipoID: TipoID, identificacion: String, correo: String, telefono: String, direccion: String, contrato: Contrato)
* asignarTurnoCliente(turno: int, cliente: Cliente): void
* buscarTurno(turno: int): Cliente
* getTurnoFirst(): Cliente
* getTurnoLast(): Cliente
* asignarHorarioEmpleado(horario: String, empleado: Empleado): void

5. Producto

**DESCRIPCIÓN:**  
Representa los artículos que la empresa comercializa o utiliza en sus servicios.

**PROPIEDADES:**

* String codigo
* String nombreProducto
* String Categoria
* double precio
* int stock
* String marca
* double precioDeCompra
* Boolean disponibilidad
* int alerta
* String codigoDelProveedor

**MÉTODOS:**  
*(No se especifican métodos en el diagrama)*

6. Servicio

**DESCRIPCIÓN:**  
Representa los servicios que ofrece la empresa a sus clientes, asociados a vehículos.

**PROPIEDADES:**

* String codigo
* String nombre
* String descripcion
* double precioDeVenta
* double porcentajeIva
* double valorCorrespondienteIVA
* double precioTotalConIva
* double descuento
* double precioTotal
* Vehiculo vehiculo
* Date fechaInicio
* Date fechaFin
* List<Empleado> funcionarios
* String imagen
* boolean disponibilidad
* Categoria categoria

**MÉTODOS:**  
*(No se especifican métodos en el diagrama)*

7. OrdenVenta

**DESCRIPCIÓN:**  
Documento que registra las transacciones de venta de productos y servicios a clientes.

**PROPIEDADES:**

* Date fechaSolicitud
* Empresa empresa
* Cliente cliente
* Empleado funcionario
* List<DetalleVentaProducto> detallesProductos
* List<DetalleVentaServicio> detallesServicios

**MÉTODOS:**  
*(No se especifican métodos en el diagrama)*

5) Relaciones entre Clases

Relaciones de Herencia:

* Persona <|-- Empleado
* Persona <|-- Cliente
* Empleado <|-- Supervisor
* Documento <|-- Cotizacion
* Documento <|-- Pedido
* Documento <|-- NotaCorrecion
* Documento <|-- ComprobantePago
* Documento <|-- OrdenVenta

Relaciones de Asociación:

* Empresa "1" -- "\*" Empleado (Una empresa tiene muchos empleados)
* Empresa "1" -- "\*" Cliente (Una empresa atiende a muchos clientes)
* Empresa "1" -- "\*" Proveedor (Una empresa trabaja con muchos proveedores)
* Cliente "1" -- "\*" Vehiculo (Un cliente puede tener varios vehículos)
* Proveedor "*" -- "*" Producto (Los proveedores suministran varios productos)
* Categoria "1" -- "\*" Producto (Una categoría contiene varios productos)
* AreaDeTrabajo "1" -- "\*" Cubiculo (Un área contiene varios cubículos)

Relaciones de Composición/Agregación:

* Empresa "1" -- "\*" Bodega (Las bodegas son parte integral de la empresa)
* Empresa "1" -- "1" InformacionPago (La información de pago es exclusiva de la empresa)
* Cotizacion "1" -- "\*" DetalleCotizacion (Los detalles son parte integral de la cotización)
* OrdenVenta "1" -- "\*" DetalleVentaProducto (Los detalles son parte de la orden)

Relaciones de Dependencia:

* Empleado "*" -- "*" OrdenVenta : genera (Los empleados generan órdenes de venta)
* Empleado "*" -- "*" Cotizacion : genera (Los empleados generan cotizaciones)
* Cubiculo "1" -- "\*" Servicio (Los servicios se realizan en cubículos)

Clases Asociación:

* DetalleCotizacion (relaciona Producto con Cotizacion)
* DetalleVentaProducto (relaciona Producto con OrdenVenta)
* DetalleVentaServicio (relaciona Servicio con OrdenVenta)