



ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS



POO

ASIGNATURA:

Programación Orientada a Objetos

PROFESOR:

Ing. Juan Carlos Gonzalez MSc.

PERÍODO ACADÉMICO:

2025-B

Deber VIII

HERENCIA, ABSTRACCIÓN, ARRAYLIST Y EXCEPCIONES

NOMBRES: VASQUEZ MERA JUAN DANIEL

OBJETIVO GENERAL

Desarrollar competencias en Programación Orientada a Objetos mediante la implementación de clases en Java que integren los principios de encapsulamiento, sobrecarga de constructores y validación de datos, aplicando estos conceptos en la resolución de problemas del mundo real.

OBJETIVOS ESPECÍFICOS

- Aplicar los principios de Herencia
- Aplicar los principios de Abstracción
- Aplicar los principios de Polimorfismo
- Aplicar los principios de ArrayList
- Aplicar los principios de Excepciones

Ejercicio 1: Sistema de Gestión de Vehículos

IMPORTANTE: Este ejercicio **NO requiere crear clases de excepciones personalizadas.**

Solo usar excepciones de Java: `IllegalArgumentException`, `IllegalStateException`, `NullPointerException`.

CLASES A CREAR:

1. Clase abstracta Vehiculo

- Atributos **PRIVADOS**: marca, modelo, año, precio
- Constructor que valide:
 - marca y modelo no null ni vacíos → `IllegalArgumentException`
 - año entre 1900 y 2025 → `IllegalArgumentException`
 - precio > 0 → `IllegalArgumentException`
- Método abstracto: `calcularImpuestoCirculacion()`
- Método `toString()`

2. Clase Auto extends Vehiculo

- Atributo privado: `numeroPuertas`
- Constructor que valide `numeroPuertas > 0` → `IllegalArgumentException`
- **SOBRECARGA de constructor:**
 - `Auto(marca, modelo, año, precio)` → puertas = 4 por defecto
 - `Auto(marca, modelo, año, precio, numeroPuertas)`
- **SOBRESCRIBIR** `calcularImpuestoCirculacion()`: return `precio * 0.05`
- **SOBRESCRIBIR** `toString()`

3. Clase Moto extends Vehiculo

- Atributo privado: `cilindrada`
- Constructor que valide `cilindrada > 0` → `IllegalArgumentException`
- **SOBRESCRIBIR** `calcularImpuestoCirculacion()`:
 - Si `cilindrada ≤ 250`: return `precio * 0.02`
 - Si `cilindrada > 250`: return `precio * 0.04`

- **SOBRESCRIBIR** `toString()`

4. Clase Camion extends Vehiculo

- Atributo privado: `capacidadCarga` (toneladas)
- Constructor que valide `capacidadCarga > 0` → `IllegalArgumentException`
- **SOBRESCRIBIR** `calcularImpuestoCirculacion(): return (precio * 0.08) + (capacidadCarga * 50)`
- **SOBRESCRIBIR** `toString()`

5. Clase Concesionaria

- Atributo privado: `ArrayList<Vehiculo> inventario`
- Métodos:
 - `agregarVehiculo(Vehiculo v)` → valida `v` no null, sino `NullPointerException`
 - `calcularTotalImpuestos()` → recorre `ArrayList`, suma impuestos. Si vacío: `IllegalStateException`
 - `obtenerVehiculoMasCaro()` → recorre `ArrayList`, compara precios. Si vacío: `IllegalStateException`
 - `ordenarPorPrecio()` → usar `Collections.sort`

6. Clase Main

- Probar crear vehículo con año < 1900 (capturar `IllegalArgumentException`)
 - Probar crear vehículo con precio negativo (capturar `IllegalArgumentException`)
 - Agregar 6 vehículos válidos (2 de cada tipo) al `ArrayList`
 - Probar `calcularTotalImpuestos()` con lista vacía (capturar `IllegalStateException`)
 - Calcular total de impuestos con lista llena
 - Obtener y mostrar vehículo más caro
 - Ordenar por precio y mostrar
-

IMPORTANTE: Este ejercicio **SÍ requiere crear 3 clases de excepciones personalizadas + usar excepciones de Java.**

EXCEPCIONES PERSONALIZADAS A CREAR:

// 1. Checked

```
public class SaldoInsuficienteException extends Exception {  
    public SaldoInsuficienteException(String mensaje) {  
        super(mensaje);  
    }  
}
```

// 2. Unchecked

```
public class MontoInvalidoException extends RuntimeException {  
    public MontoInvalidoException(String mensaje) {  
        super(mensaje);  
    }  
}
```

// 3. Unchecked

```
public class CuentaInactivaException extends RuntimeException {  
    public CuentaInactivaException(String mensaje) {  
        super(mensaje);  
    }  
}
```

CLASES A CREAR:

1. Clase CuentaBancaria

- Atributos **PRIVADOS**: numeroCuenta, titular, saldo

- Constructor que valide:
 - numeroCuenta y titular no null ni vacíos → IllegalArgumentException
 - saldo ≥ 0 → IllegalArgumentException
- Métodos:
 - depositar(double monto)
 - retirar(double monto) throws SaldoInsuficienteException
 - toString()

2. Clase CuentaAhorros extends CuentaBancaria

- Atributo privado: tasaInteres (entre 0 y 1)
- Constructor que valide tasaInteres entre 0 y 1 → IllegalArgumentException
- **SOBRESCRIBIR** retirar(double monto) throws SaldoInsuficienteException:
 - Si monto ≤ 0 : lanzar MontoInvalidoException
 - Si saldo después del retiro < 50 : lanzar SaldoInsuficienteException
- Método aplicarInteres(): saldo = saldo + (saldo * tasaInteres)

3. Clase CuentaCorriente extends CuentaBancaria

- Atributo privado: limiteCredito
- Constructor que valide limiteCredito ≥ 0 → IllegalArgumentException
- **SOBRESCRIBIR** retirar(double monto) throws SaldoInsuficienteException:
 - Si monto ≤ 0 : lanzar MontoInvalidoException
 - Permitir retiro hasta saldo + limiteCredito
 - Si excede: lanzar SaldoInsuficienteException

4. Clase CuentaInversion extends CuentaBancaria

- Atributos privados: montoMinimo, rendimientoAnual
- Constructor que valide montoMinimo > 0 y rendimientoAnual ≥ 0 → IllegalArgumentException
- **SOBRESCRIBIR** depositar(double monto):
 - Si monto $< montoMinimo$: lanzar MontoInvalidoException
- **SOBRECARGA:**

- calcularRendimiento(): return saldo * (rendimientoAnual / 12)
- calcularRendimiento(int meses): return saldo * (rendimientoAnual / 12) * meses

5. Clase Banco

- Atributos privados: ArrayList<CuentaBancaria> cuentas, nombre
- Constructor que valide nombre no vacío → IllegalArgumentException
- **Método ESTÁTICO:** generarNúmeroCuenta() → retorna String aleatorio de 10 dígitos
- Métodos:
 - abrirCuenta(CuentaBancaria cuenta) → valida no null, sino NullPointerException
 - transferir(String origen, String destino, double monto) throws SaldoInsuficienteException:
 - Valida monto > 0, sino MontoInvalidoException
 - Busca ambas cuentas en ArrayList
 - Si no existen: IllegalArgumentException
 - Retira de origen y deposita en destino
 - aplicarInteresesAhorros():
 - Recorre ArrayList
 - Para cada CuentaAhorros: si saldo == 0, lanzar CuentaInactivaException
 - Aplicar interés
 - obtenerSaldoTotal() → suma saldos. Si vacío: IllegalStateException
 - ordenarPorSaldo() → **implementar algoritmo manual**

6. Clase Main

- Probar crear cuenta con titular vacío (IllegalArgumentException)
- Probar crear cuenta con saldo negativo (IllegalArgumentException)
- Agregar 6 cuentas válidas (2 de cada tipo)
- Realizar depósitos y retiros
- Probar retiro con saldo insuficiente (SaldoInsuficienteException)

- Probar transferencia exitosa y fallida
 - Probar monto negativo (MontolnvalidoException)
 - Calcular saldo total del banco
 - Aplicar intereses y probar CuentalnactivaException
 - Ordenar por saldo
-

Ejercicio 3: Tienda de Productos con Inventario

IMPORTANTE: Este ejercicio **requiere crear 3 clases de excepciones personalizadas + usar excepciones de Java.**

EXCEPCIONES PERSONALIZADAS A CREAR:

java

// 1. Checked

```
public class StockInsuficienteException extends Exception {  
    public StockInsuficienteException(String mensaje) {  
        super(mensaje);  
    }  
}
```

// 2. Unchecked

```
public class ProductoNoEncontradoException extends RuntimeException {  
    public ProductoNoEncontradoException(String mensaje) {  
        super(mensaje);  
    }  
}
```

// 3. Unchecked

```
public class PrecioInvalidoException extends RuntimeException {
```

```
public PrecioInvalidoException(String mensaje) {  
    super(mensaje);  
}  
}
```

CLASES A CREAR:

1. Clase abstracta Producto

- Atributos **PRIVADOS**: codigo, nombre, precioBase, stock
- Constructor que valide:
 - codigo y nombre no null ni vacíos → IllegalArgumentException
 - precioBase > 0 → PrecioInvalidoException
 - stock ≥ 0 → IllegalArgumentException
- Método abstracto: calcularPrecioFinal()
- Método toString()

2. Clase ProductoElectronico extends Producto

- Atributos privados: marca, garantiaMeses
- Constructor que valide marca no vacía y garantiaMeses > 0 → IllegalArgumentException
- **SOBRESCRIBIR** calcularPrecioFinal(): return precioBase * 1.12 (IVA 12%)
- **SOBRECARGA:**
 - aplicarGarantia(): return garantiaMeses == 12
 - aplicarGarantia(int mesesExtras): valida mesesExtras > 0 y retorna costo adicional

3. Clase ProductoAlimenticio extends Producto

- Atributos privados: fechaVencimiento, requiereRefrigeracion
- Constructor que valide fechaVencimiento no vacío → IllegalArgumentException
- **SOBRESCRIBIR** calcularPrecioFinal(): return precioBase (IVA 0%)

4. Clase ProductoRopa extends Producto

- Atributos privados: talla, material

- Constructor que valide talla y material no vacíos → IllegalArgumentException
- **SOBRESCRIBIR** calcularPrecioFinal(): return precioBase * 1.12 (IVA 12%)

5. Clase Inventario

- Atributos privados: ArrayList<Producto> productos, nombreTienda
- Constructor que valide nombreTienda no vacío → IllegalArgumentException
- **Método ESTÁTICO:** generarCodigo(String categoria) → retorna código con prefijo (ej: "ELEC-1234")
- Métodos:
 - agregarProducto(Producto p) → valida no null, sino NullPointerException
 - buscarPorCodigo(String codigo) throws ProductoNoEncontradoException:
 - Recorre ArrayList buscando
 - Si no existe: lanzar excepción
 - venderProducto(String codigo, int cantidad) throws StockInsuficienteException:
 - Valida cantidad > 0, sino IllegalArgumentException
 - Busca producto
 - Si stock < cantidad: lanzar StockInsuficienteException
 - Reduce stock y retorna precio total
 - calcularValorInventario():
 - Recorre ArrayList
 - Suma (precioFinal * stock) de cada producto
 - Si vacío: IllegalStateException
 - listarProductosBajoStock(int minimo) → retorna ArrayList con productos donde stock < minimo
 - ordenarPorPrecio() → **implementar algoritmo manual**
 - **SOBRESCRIBIR** toString()

6. Clase Main

- Probar crear producto con precio negativo (PrecioInvalidoException)

- Probar crear producto con código vacío (IllegalArgumentException)
 - Agregar 10 productos válidos (mix de tipos)
 - Realizar ventas exitosas
 - Probar venta con cantidad negativa (IllegalArgumentException)
 - Probar venta con stock insuficiente (StockInsuficienteException)
 - Probar búsqueda de producto inexistente (ProductoNoEncontradoException)
 - Calcular valor total del inventario
 - Listar productos con stock < 5
 - Ordenar por precio
-

Ejercicio 4: Sistema de Biblioteca Digital (SIMPLIFICADO)

IMPORTANTE: Este ejercicio **requiere crear 3 clases de excepciones personalizadas + usar excepciones de Java.**

EXCEPCIONES PERSONALIZADAS A CREAR:

java

// 1. Checked

```
public class CódigoInvalidoException extends Exception {  
    public CódigoInvalidoException(String mensaje) {  
        super(mensaje);  
    }  
}
```

// 2. Unchecked

```
public class MaterialNoDisponibleException extends RuntimeException {  
    public MaterialNoDisponibleException(String mensaje) {  
        super(mensaje);  
    }  
}
```

```
}
```

```
// 3. Unchecked
```

```
public class MaterialNoEncontradoException extends RuntimeException {  
    public MaterialNoEncontradoException(String mensaje) {  
        super(mensaje);  
    }  
}
```

CLASES A CREAR:

1. Clase abstracta MaterialBiblioteca

- Atributos **PRIVADOS**: codigo, titulo, autor, anioPublicacion, estaPrestado
- Constructor que valide:
 - codigo, titulo, autor no null ni vacíos → IllegalArgumentException
 - anioPublicacion entre 1000 y 2025 → IllegalArgumentException
- Métodos abstractos:
 - calcularMultaPorRetraso(int diasRetraso)
 - calcularTiempoMaximoPrestamo()
- Métodos: prestar(), devolver(), toString()

2. Clase Libro extends MaterialBiblioteca

- Atributos privados: numPaginas, editorial, esDigital
- Constructor que valide numPaginas > 0 y editorial no vacía → IllegalArgumentException
- **SOBRESCRIBIR** calcularMultaPorRetraso(int diasRetraso):
 - Valida diasRetraso ≥ 0, sino IllegalArgumentException
 - Si esDigital: return diasRetraso * 0.50
 - Si físico: return diasRetraso * 1.00
- **SOBRESCRIBIR** calcularTiempoMaximoPrestamo():

- Si esDigital: return 7
- Si físico: return 15

3. Clase Revista extends MaterialBiblioteca

- Atributos privados: numeroEdicion, mesPublicacion, esEspecializada
- Constructor que valide numeroEdicion > 0 y mesPublicacion no vacío → IllegalArgumentException
- **SOBRESCRIBIR** calcularMultaPorRetraso(int diasRetraso):
 - Valida diasRetraso ≥ 0, sino IllegalArgumentException
 - Si esEspecializada: return diasRetraso * 2.00
 - Si normal: return diasRetraso * 0.75
- **SOBRESCRIBIR** calcularTiempoMaximoPrestamo():
 - Si esEspecializada: return 5
 - Si normal: return 7

4. Clase DVD extends MaterialBiblioteca

- Atributos privados: duracionMinutos, genero, tieneSubtitulos
- Constructor que valide duracionMinutos > 0 y genero no vacío → IllegalArgumentException
- **SOBRESCRIBIR** calcularMultaPorRetraso(int diasRetraso):
 - Valida diasRetraso ≥ 0, sino IllegalArgumentException
 - return diasRetraso * 1.50
- **SOBRESCRIBIR** calcularTiempoMaximoPrestamo(): return 3
- **SOBRECARGA:**
 - calcularMultaPorRetraso(int diasRetraso)
 - calcularMultaPorRetraso(int diasRetraso, boolean esEstreno) → si esEstreno: multa * 2

5. Clase Biblioteca

- Atributos privados: ArrayList<MaterialBiblioteca> catalogo, nombre
- Constructor que valide nombre no vacío → IllegalArgumentException

- **Método ESTÁTICO:** validarCodigo(String codigo) throws CódigoInvalidoException:
 - Valida formato: "LIB-XXX" o "REV-XXX" o "DVD-XXX"
 - Si no cumple: lanzar excepción
- **Método ESTÁTICO:** generarCodigoAleatorio(String tipo) → genera código según tipo
- Métodos:
 - agregarMaterial(MaterialBiblioteca m) throws CódigoInvalidoException:
 - Valida m no null, sino NullPointerException
 - Valida código usando método estático
 - buscarMaterial(String codigo) throws MaterialNoEncontradoException:
 - Recorre ArrayList buscando
 - Si no existe: lanzar excepción
 - prestarMaterial(String codigo) throws MaterialNoDisponibleException, MaterialNoEncontradoException:
 - Busca material
 - Si estaPrestado: lanzar MaterialNoDisponibleException
 - Marca como prestado
 - devolverMaterial(String codigo, int diasRetraso) throws MaterialNoEncontradoException:
 - Valida diasRetraso ≥ 0, sino IllegalArgumentException
 - Busca material
 - Calcula y retorna multa
 - listarMaterialesDisponibles() → recorre y muestra materiales con estaPrestado = false
 - ordenarPorAnio() → **implementar algoritmo manual** (bubble sort o selection sort)

6. Clase Main

- Probar crear material con título vacío (IllegalArgumentException)
- Probar crear material con año < 1000 (IllegalArgumentException)

- Probar agregar material con código inválido (CodigoInvalidoException)
- Agregar 8 materiales válidos (2 Libros, 2 Revistas, 2 DVDs, 2 de cualquier tipo)
- Realizar 4 préstamos exitosos
- Probar préstamo de material ya prestado (MaterialNoDisponibleException)
- Probar buscar material inexistente (MaterialNoEncontradoException)
- Devolver 2 materiales con diferentes días de retraso y mostrar multas
- Listar materiales disponibles
- Ordenar por año y mostrar

Entregables:

- Crear una carpeta general que contenga un proyecto de IntelliJIdea por cada ejercicio comprimirla y subirla al aula virtual : ApellidoNombre_Deber.zip
- En este documento adjuntar el enlace de GitHub de cada ejercicio.

Ejercicio 1: Sistema de Gestión de Vehículos

Enlace: https://github.com/DanielV593/Programacion-Orientada-a-Objetos/tree/main/Deberes/VasquezJuan_Deber8/Sistema%20de%20Gestion%20de%20Vehiculos

Ejercicio 2: Gestión de Cuentas Bancarias

Enlace: https://github.com/DanielV593/Programacion-Orientada-a-Objetos/tree/main/Deberes/VasquezJuan_Deber8/Sistema%20de%20Gestion%20de%20Cuentas%20Bancarias

Ejercicio 3: Tienda de Productos con Inventario

Enlace: https://github.com/DanielV593/Programacion-Orientada-a-Objetos/tree/main/Deberes/VasquezJuan_Deber8/Tienda%20de%20Productos%20-%20Inventario

Ejercicio 4: Sistema de Biblioteca Digital (SIMPLIFICADO)

Enlace: https://github.com/DanielV593/Programacion-Orientada-a-Objetos/tree/main/Deberes/VasquezJuan_Deber8/Sistema%20de%20Biblioteca%20Digital