

# Reporte de Evaluación - Fork de GitHub

## Información General

Estudiante: Santiago Zapata Villada  
Repositorio: SantiagoZVcesde/act\_b1\_s7  
Fecha de evaluación: 5/10/2025, 10:51:03  
Evaluado por: Sistema de Evaluación Masiva

## Resumen de Calificaciones

Calificación general: 3.0/5.0  
Actividades completadas: 3/4  
Porcentaje de completitud: 75.0%

## Detalle de Actividades

#	Descripción	Archivo	Encontrado	Calificación
1	SISTEMA TIENDA - Ejercicio 1: Implementa...	src/main/java/com/example/Main.java	Sí	0.0
2	CLASE PADRE - Ejercicio 2: Implementa la...	src/main/java/com/example/Producto.java	Sí	3.0
3	CLASE HIJA ALIMENTICIO - Ejercicio 3: Im...	src/main/java/com/example/ProductoAlimenticio.java	Sí	4.0
4	CLASE HIJA ELECTRONICO - Ejercicio 4: Im...	src/main/java/com/example/ProductoElectronico.java	Sí	5.0

## Retroalimentación Detallada

**Actividad 1: SISTEMA TIENDA - Ejercicio 1: Implementa la clase SistemaTienda para probar el sistema completo. INDICACIONES:** 1) Crear inventario diverso con productos electrónicos y alimenticios, 2) Crear ProductoElectronico laptop importada con 24 meses garantía, 110V, precio base \$2,500,000, stock 5 unidades, 3) Crear ProductoElectronico celular con parámetros específicos, 4) Crear ProductoAlimenticio leche refrigerada con 5 días para vencer, precio base \$4,500, stock 20 unidades, 5) Crear ProductoAlimenticio pan con parámetros específicos, 6) Probar venta exitosa y fallida por falta de stock, 7) Calcular precios usando métodos específicos de cada clase, 8) Probar reabastecimiento de inventario, 9) Verificar compatibilidad de voltaje, 10) Comparar precio base vs precios específicos, 11) Demostrar funcionalidades únicas de cada tipo.

Archivo esperado: src/main/java/com/example/Main.java  
Estado: Archivo encontrado  
Calificación: 0.0/5.0  
Retroalimentación:

La clase Main está vacía, no hay implementación de la lógica del sistema de tienda. No se cumple con ninguno de los requisitos de la actividad.

**Actividad 2: CLASE PADRE - Ejercicio 2: Implementa la clase base Producto con atributos protegidos y métodos obligatorios. INDICACIONES:** 1) Declarar atributos protegidos: String nombre, double precioBase, String codigo (formato PROD-XXXX), int cantidadStock, String categoria, boolean activo, 2) Constructor: public Producto(String nombre, double precioBase, String codigo, int cantidadStock, String categoria), 3) Getters y setters con validaciones: setPrecioBase() solo acepta valores > 0, setCantidadStock() solo acepta valores >= 0, setCodigo() debe seguir formato PROD-XXXX, 4) Método calcularPrecioFinal(): calcular precio base + IVA (19%), retornar precioBase \* 1.19, 5) Método hayStock(int cantidad): verificar cantidadStock >= cantidad Y producto activo, 6) Método vender(int cantidad): verificar stock con hayStock(), reducir cantidadStock si hay stock, mostrar mensajes apropiados, 7) Método reabastecer(int cantidad): sumar cantidad al stock, mostrar mensaje de reabastecimiento, 8) Método calcularDescuento(): retornar 0.0 en clase padre.

Archivo esperado: src/main/iava/com/example/Producto.iava

**Actividad 3: CLASE HIJA ALIMENTICIO - Ejercicio 3: Implementa ProductoAlimenticio que herede de Producto con atributos y métodos específicos. INDICACIONES:** 1) Usar extends Producto para heredar, 2) Atributos adicionales: boolean refrigerado, String lote, int diasParaVencer, 3) Constructor que llame a super() con parámetros de clase padre, 4) Método calcularPrecioAlimenticio(): obtener precio con IVA usando calcularPrecioFinal(), si refrigerado agregar 8% sobre precioBase, calcular descuento con calcularDescuentoVencimiento(), aplicar descuento, 5) Método calcularDescuentoVencimiento(): si diasParaVencer <= 3 retornar 0.50, si <= 7 retornar 0.30, si <= 15 retornar 0.15, otros casos 0.0, 6) Método estaProximoAVencer(): retornar true si diasParaVencer <= 7, 7) Método estaVencido(): retornar true si diasParaVencer <= 0, 8) Método obtenerEstadoFrescura(): retornar VENCIDO, URGENTE, PRÓXIMO A VENCER, CONSUMIR PRONTO o FRESCO según días, 9) Método calcularPerdidaPorVencimiento(): si vencido retornar precioBase \* cantidadStock, sino 0.0.

Archivo esperado: src/main/java/com/example/ProductoAlimenticio.java

Estado: Archivo encontrado

Calificación: 4.0/5.0

Retroalimentación:

La solución es correcta y funcional, cumpliendo con todos los requerimientos. Se sugiere usar constantes (enums) para mejorar la legibilidad del estado de frescura y aplicar formato al código.

**Actividad 4: CLASE HIJA ELECTRONICO - Ejercicio 4: Implementa ProductoElectronico que herede de Producto con atributos y métodos específicos. INDICACIONES:** 1) Usar extends Producto para heredar, 2) Atributos adicionales: int garantiaMeses, String marca, double voltaje, boolean esImportado, 3) Constructor que llame a super() con parámetros de clase padre, 4) Método calcularPrecioElectronico(): obtener precio con IVA usando calcularPrecioFinal(), si esImportado agregar impuesto 5% sobre precioBase, calcular descuento con calcularDescuentoGarantia(), aplicar descuento, 5) Método calcularDescuentoGarantia(): si garantiaMeses >= 24 retornar 0.10, si >= 12 retornar 0.05, otros casos 0.0, 6) Método esGarantiaExtendida(): retornar true si garantiaMeses > 12, 7) Método calcularCostoGarantia(): retornar precioBase \* 0.02 \* garantiaMeses, 8) Método esCompatibleVoltaje(double voltajeLocal): calcular diferencia absoluta con Math.abs(), retornar true si diferencia <= 10% del voltajeLocal.

Archivo esperado: src/main/java/com/example/ProductoElectronico.java

Estado: Archivo encontrado

Calificación: 5.0/5.0

Retroalimentación:

Excelente solución. Cumple con todos los requisitos de la actividad de manera clara y concisa. El código es legible y bien estructurado, aplicando buenas prácticas.

## Resumen General

Buen trabajo general. Completó 3/4 actividades (75%) con una calificación promedio de 3.0/5. Hay oportunidades de mejora en algunos aspectos.

## Recomendaciones

- Revisar y mejorar las actividades con calificación baja