LEONEL CRUZ
FERNANDO VEGA

Error #1:

- **Ubicación:** Producto.java (línea 5-8)
- **Tipo de error:** Estructural - Encapsulamiento
- **Descripción del problema:** Los atributos de la clase Producto (id, nombre, precio y stock) están públicos, produce una mala práctica de encapsulamiento.
- **Solución implementada:** Dentro de la estructura se le cambio a los atributos de estar public a private, así solucionando los errores de la mala práctica.
- **Impacto:** Al tener atributos públicos, se permite el acceso y modificación directa de los datos, violando el principio de encapsulamiento y haciendo imposible controlar o validar cambios en los atributos.

Error #2:

- **Ubicación:** Producto.java (línea 12-15)
- **Tipo de error:** Mala práctica
- **Descripción del problema:** Variables sin validación dentro del constructor.
- **Solución implementada:** A la variable nombre con la validación para no ingresar nombre vacías, las demás variables para no meter valores negativos.
- **Impacto:** Ingreso de información errónea o mal intencionada.

Error #3:

- **Ubicación:** Main.java (línea 95)
- **Tipo de error:** Lógico-Estructural
- **Descripción del problema:** En la estructura de agregarProductoAPedido dentro de esta la lógica no permite que cuando se hace un pedido este se le reste al stock o inventario, por lo que la función de reducirStock no se utiliza.
- **Solución implementada:** Dentro de la estructura, en el if se llama a la función para que cuando se agregue algún producto en el pedido este se le reste al stock.
- **Impacto:** Errores de inventario en el inventario y cuentas no claras.

Error #4:

- **Ubicación:** Producto.java (líneas 21-23)
- **Tipo de error:** Lógico
- **Descripción del problema:** El método hayStock() usa comparación estricta (>) cuando debería ser >=.
- **Solución implementada:** Cambiar la condición a
- **Impacto:** No permite utilizar todo el stock disponible cuando se pide exactamente la cantidad existente.

Error #5:

- **Ubicación: Pedido.java (línea 28-33)**
- **Tipo de error:** Lógico
- **Descripción del problema:** El método calcularTotal() suma solo los precios de los productos sin considerar las cantidades de cada producto en el pedido.
- **Solución implementada:** Modificar el método para que multiplique el precio de cada producto por su cantidad correspondiente
- **-Impacto:** Sin esta corrección, los totales de los pedidos serán incorrectos (menores al valor real), lo que afectaría los ingresos reportados y podría causar pérdidas económicas.

Error #6:

- **Ubicación:** Pedido.java (línea 35-38)
- **Tipo de error:** Estructural - Manejo de colecciones
- **Descripción del problema:** El método obtenerPrimerProducto() no verifica si la lista productos está vacía antes de intentar acceder al primer elemento, lo que puede causar una IndexOutOfBoundsException.
- **Solución implementada:** Agregar validación para lista vacía y manejar el caso adecuadamente.
- **-Impacto:** Sin esta corrección, la aplicación podría fallar inesperadamente cuando se intente obtener un producto de una lista vacía, afectando la experiencia del usuario y posiblemente interrumpiendo operaciones.

Error #7:

- **Ubicación:** Main.java (línea 120)
- **Tipo de error:** Estructural
- **Descripción del problema:** En la estructura de agregarProductoAPedido dentro de esta la lógica no permite que cuando se hace un pedido este se le reste al stock o inventario, por lo que la función de reducirStock no se utiliza.
- **Solución implementada:** Dentro de la estructura, en el if se llama a la función para que cuando se agregue algún producto en el pedido este se le reste al stock.
- **Impacto:** Errores de inventario en el inventario y cuentas no claras.

Error: 8#

- **Ubicación:** inventarioservice.java (línea 25)
- **Tipo de error:** Control de flujo
- **Descripción del problema:** El bucle while (i <= productos.size()) puede acceder a una posición fuera del rango válido de la lista, ya que el índice máximo permitido es productos.size() - 1. Esto genera un riesgo de IndexOutOfBoundsException.
- **Solución implementada:** Se cambió la condición del bucle de i <= productos.size() a i < productos.size() para asegurar que no se acceda a un índice inválido.

- **Impacto:** El error podía causar fallos en tiempo de ejecución y detener el programa al buscar un producto que no existe, provocando una excepción al acceder a un índice fuera de rango.

Error #9:

- **Ubicación:** [inventarioService.java] (34-43)
- **Tipo de error:** Lógico
- **Descripción del problema:** El método venderProducto no actualizaba el stock después de realizar la venta, lo que hacía que el inventario no reflejara las ventas.
- **Solución implementada:** Se agregó la línea que reduce el stock producto.stock -= cantidad; justo después de verificar que hay stock suficiente.
- **Impacto:** No actualizar el stock permite vender productos que no están disponibles, causando errores en la gestión del inventario.

Error #10:

- **Ubicación:** [InventarioService.java] (56-58)
- **Tipo de error:** [Lógico/Mala práctica]
- **Descripción del problema:** [Se encontró **código duplicado** entre los métodos obtenerTodosLosProductos() y obtenerProductosDisponibles(). Además, la condición original producto.stock >= 0 era incorrecta, ya que incluía productos con stock 0 como "disponibles"
- **Solución implementada:** Se creó el método privado filtrarProductos(boolean soloDisponibles) que centraliza el filtrado. Los métodos públicos llaman a este método y se corrigió la condición a producto.stock > 0.
- **Impacto:** Se eliminó la duplicación que dificultaba el mantenimiento y se corrigió el listado incorrecto de productos sin existencia.

Error #11:

- **Ubicación:** PedidoService.java (línea X)
- **Tipo de error:** Lógico / Mala práctica
- **Descripción del problema:**
  En el método crearPedido, el contador de pedidos se estaba **decrementando** cuando debería **incrementarse** para asignar IDs únicos y consecutivos a cada pedido.
- **Solución implementada:**
  Se corrigió el incremento de contadorPedidos usando contadorPedidos++ para asegurar que cada nuevo pedido tenga un identificador único y ascendente.
- **Impacto:**
  La decrementación causaba repetidos o incorrectos, lo que podía generar confusión o errores al manejar pedidos.

Error #12:

- **Ubicación:** PedidoService.java (37)
- **Tipo de error:** Lógico / Control de flujo
- **Descripción del problema:**
  El bucle usa una condición incorrecta i <= cantidad, lo que hace que el ciclo itere una vez más de lo necesario (se itera cantidad+1 veces en vez de cantidad). Esto genera un intento de vender más productos de los que se deben.

- **Solución implementada:**
  Se corrigió la condición del bucle a i < cantidad para iterar exactamente la cantidad requerida.
- **Impacto:**
  La condición incorrecta causaba errores en la venta, con sobreventa o intentos fallidos que afectan la lógica y la consistencia del p