Universidad privada del Norte

Facultad de Ingeniería

Carrera Profesional Ingeniería de Sistemas



**Docente:**

Jose Carlos Anicama Silva

**Integrantes:**

**-**Anderson Fabrizzio Larios Tito

-Gianino Hector Naranjo Servidio

-Fernando Rafael Marquez Mesta

-Yimmy Diaz Jange

**Sistema de Inventario de Productos para Tienda**

**Desarrollo y Documentación Integral del Proyecto de Software**

**1. Identificación de Restricciones Realistas y Alternativas de Solución**

**Restricciones Realistas:**

**1. Restricciones Tecnológicas:**

* Solo se puede utilizar Java básico con arrays, constructores, getters/setters y herencia
* No se permite el uso de librerías externas o Collections
* Limitación a estructuras de control básicas (if-else, switch-case, bucles)

**2. Restricciones Académicas:**

* Desarrollo con conocimientos
* Uso exclusivo de arrays para almacenamiento de datos
* Implementación de herencia para organizar las clases

**3. Restricciones de Tiempo:**

* Período de desarrollo limitado a 4 semanas
* Tiempo de pruebas básicas de 1 semana

**Alternativas de Solución:**

**Frente a limitaciones tecnológicas:**

* Utilizar arrays para almacenar productos con tamaño fijo predefinido
* Implementar herencia para crear diferentes tipos de productos
* Usar constructores para inicializar objetos correctamente
* Aplicar getters y setters para encapsulamiento

**Frente a limitaciones académicas:**

* Crear clases base y derivadas usando herencia
* Implementar menús con switch-case y bucles do-while
* Usar arrays paralelos para manejar múltiples atributos

**2. Objetivos del Proyecto**

**Objetivo General:**

Desarrollar un sistema básico de control de inventario para una tienda utilizando únicamente arrays, constructores, getters/setters y herencia en Java, aplicando los conceptos de programación orientada a objetos vistos en clase.

**Objetivos Específicos:**

1. **Aplicar Herencia:** Crear una jerarquía de clases para diferentes tipos de productos
2. **Usar Arrays:** Implementar el almacenamiento de productos usando arrays
3. **Implementar Encapsulamiento:** Usar getters y setters para acceso controlado a atributos
4. **Aplicar Constructores:** Crear constructores para inicialización correcta de objetos
5. **Desarrollar Menús:** Crear interfaz de consola con switch-case y bucles

**3. Alcance de la Solución**

**Dentro del Alcance:**

**Funcionalidades Principales:**

* Registro de productos usando constructores
* Búsqueda de productos en arrays
* Clasificación por categorías usando herencia
* Operaciones básicas de inventario
* Interfaz de consola con menús
* Aplicación de getters y setters

**Aspectos Técnicos:**

* Uso de arrays para almacenamiento
* Herencia con clase padre Producto y clases hijas
* Constructores con parámetros
* Métodos getter y setter
* Menús interactivos con switch-case

**Fuera del Alcance:**

* Manejo de archivos (no visto en clase aún)
* Collections o estructuras avanzadas
* Interfaces gráficas
* Bases de datos
* Exportación de datos

**4. Requerimientos Funcionales (40 Requerimientos)**

|  |  |  |
| --- | --- | --- |
| **ID** | **Requerimiento** | **Criterio de Aceptación** |
| RF01 | Crear producto usando constructor | Se acepta si el objeto se crea correctamente con todos los atributos |
| RF02 | Validar datos usando setters | Se acepta si los setters validan datos antes de asignar |
| RF03 | Obtener información usando getters | Se acepta si los getters devuelven valores correctos |
| RF04 | Registrar producto en array | Se acepta si el producto se almacena en la primera posición disponible |
| RF05 | Buscar producto por nombre en array | Se acepta si encuentra el producto recorriendo el array |
| RF06 | Listar todos los productos del array | Se acepta si muestra todos los productos no nulos |
| RF07 | Clasificar productos por herencia | Se acepta si ProductoEconomico, ProductoEstandar y ProductoPremium heredan de Producto |
| RF08 | Eliminar producto del array | Se acepta si marca la posición como null |
| RF09 | Modificar nombre usando setter | Se acepta si actualiza el nombre validando que no esté vacío |
| RF10 | Modificar precio usando setter | Se acepta si actualiza el precio validando que sea positivo |
| RF11 | Modificar cantidad usando setter | Se acepta si actualiza la cantidad validando que no sea negativa |
| RF12 | Contar productos en array | Se acepta si cuenta elementos no nulos correctamente |
| RF13 | Mostrar menú principal con switch-case | Se acepta si muestra opciones y procesa la selección |
| RF14 | Validar entrada de menú | Se acepta si valida opciones válidas del menú |
| RF15 | Filtrar por categoría | Se acepta si muestra solo productos de la categoría seleccionada |
| ID | Requerimiento | Criterio de Aceptación |
| RF16 | Calcular valor total del producto (precio x cantidad) | Se acepta si devuelve correctamente el valor total |
| RF17 | Mostrar detalles de un producto | Se acepta si imprime todos los atributos correctamente |
| RF18 | Verificar existencia de un producto por nombre | Se acepta si devuelve true si el nombre existe en el array |
| RF19 | Aumentar cantidad de producto | Se acepta si incrementa la cantidad sin errores |
| RF20 | Disminuir cantidad de producto | Se acepta si disminuye cantidad y no permite valores negativos |
| RF21 | Mostrar solo productos con stock | Se acepta si filtra productos cuya cantidad es mayor a 0 |
| RF22 | Buscar producto por ID | Se acepta si encuentra el producto comparando por ID |
| RF23 | Ordenar productos por nombre | Se acepta si los productos se muestran en orden alfabético |
| RF24 | Ordenar productos por precio descendente | Se acepta si se listan del más caro al más barato |
| RF25 | Registrar fecha de creación del producto | Se acepta si la fecha se asigna correctamente al crear el producto |
| RF26 | Filtrar productos con precio mayor a un valor | Se acepta si muestra productos con precio superior al indicado |
| RF27 | Mostrar estadísticas de precios | Se acepta si calcula promedio, mínimo y máximo correctamente |
| RF28 | Generar reporte en consola | Se acepta si el reporte muestra todos los productos con formato legible |
| RF29 | Exportar productos a archivo TXT | Se acepta si los datos se escriben en un archivo de texto |
| RF30 | Importar productos desde archivo TXT | Se acepta si se cargan productos válidos desde archivo |
| RF31 | Confirmar eliminación de producto | Se acepta si pide confirmación antes de eliminar |
| RF32 | Permitir salida segura del sistema | Se acepta si muestra mensaje de despedida y termina ejecución |
| RF33 | Implementar búsqueda parcial por nombre | Se acepta si encuentra coincidencias con partes del nombre |
| RF34 | Agregar atributo descripción al producto | Se acepta si el atributo se muestra y edita correctamente |
| RF35 | Validar longitud mínima del nombre | Se acepta si rechaza nombres demasiado cortos |
| RF36 | Identificar producto más caro | Se acepta si muestra el producto con mayor precio |
| RF37 | Identificar producto con mayor stock | Se acepta si muestra el producto con mayor cantidad |
| RF38 | Implementar submenú para productos premium | Se acepta si el menú solo gestiona productos de esa categoría |
| RF39 | Evitar duplicados al registrar producto | Se acepta si no permite crear productos con mismo nombre o ID |
| RF40 | Mostrar productos ordenados por fecha de creación | Se acepta si se listan cronológicamente desde el más antiguo al más reciente |

1. **Tabla del estado de requerimientos**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requerimiento** | **Estado** | **Observaciones** |
| RF01 | Crear producto usando constructor | **Completo** | Implementado en clase Producto con validaciones básicas |
| RF02 | Validar datos usando setters | **En proceso** | Validaciones básicas pero sin setters específicos |
| RF03 | Obtener información usando getters | **Completo** | Acceso directo a atributos (sin getters formales) |
| RF04 | Registrar producto en array | **Completo** | Método agregarProducto() usando ArrayList |
| RF05 | Buscar producto por nombre en array | **Completo** | Método buscarProductoPorNombre() implementado |
| RF06 | Listar todos los productos del array | **Completo** | getProductos() y actualizarTabla() implementados |
| RF07 | Clasificar productos por herencia | **Pendiente** | No implementado en versión actual |
| RF08 | Eliminar producto del array | **Completo** | eliminarProducto() implementado |
| RF09 | Modificar nombre | **Completo** | Actualización directa con validación en diálogos |
| RF10 | Modificar precio | **Completo** | Actualización directa con validación en diálogos |
| RF11 | Modificar cantidad | **Completo** | actualizarStock() implementado |
| RF12 | Contar productos en array | **Completo** | Usando size() de ArrayList |
| RF13 | Mostrar menú principal | **Completo** | Interfaz gráfica con JFrame implementada |
| RF14 | Validar entrada de menú | **Completo** | Validaciones en diálogos con JOptionPane |
| RF15 | Filtrar por categoría | **Completo** | mostrarPorCategoria() implementado |
| RF16 | Calcular valor total del producto | **Completo** | Implementado en lógica de interfaz |
| RF17 | Mostrar detalles de un producto | **En proceso** | mostrarDetalles() básico implementado |
| RF18 | Verificar existencia de producto por nombre | **En proceso** | Similar a RF05 pero sin retorno booleano |
| RF19 | Aumentar cantidad de producto | **Pendiente** | No implementado como método específico |
| RF20 | Disminuir cantidad de producto | **Pendiente** | No implementado como método específico |
| RF21 | Mostrar solo productos con stock | **Completo** | reporteAgotados() implementado (inverso) |
| RF22 | Buscar producto por ID | **Completo** | buscarPorCodigo() implementado |
| RF23 | Ordenar productos por nombre | **Pendiente** | No implementado |
| RF24 | Ordenar productos por precio descendente | **Completo** | ordenarPorPrecio() implementado |
| RF25 | Registrar fecha de creación del producto | **Pendiente** | No se incluyó atributo fecha |
| RF26 | Filtrar productos con precio mayor a un valor | **En proceso** | Similar a buscarPorRangoPrecio() |
| RF27 | Mostrar estadísticas de precios | **Pendiente** | No implementado |
| RF28 | Generar reporte en consola | **Pendiente** | La interfaz es gráfica, no por consola |
| RF29 | Exportar productos a archivo TXT | **Pendiente** | No implementado |
| RF30 | Importar productos desde archivo TXT | **Pendiente** | No implementado |
| RF31 | Confirmar eliminación de producto | **Completo** | Implementado con JOptionPane.showConfirmDialog |
| RF32 | Permitir salida segura del sistema | **Completo** | Implementado en menú principal |
| RF33 | Implementar búsqueda parcial por nombre | **Completo** | buscarProductoPorNombre() usa contains() |
| RF34 | Agregar atributo descripción al producto | **Pendiente** | No se incluyó este atributo |
| RF35 | Validar longitud mínima del nombre | **Pendiente** | Solo validación de no vacío implementada |
| RF36 | Identificar producto más caro | **Completo** | obtenerMasCaro() implementado |
| RF37 | Identificar producto con mayor stock | **Pendiente** | No implementado |
| RF38 | Implementar submenú para productos premium | **Pendiente** | No implementado |
| RF39 | Evitar duplicados al registrar producto | **Completo** | codigoExiste() implementado |
| RF40 | Mostrar productos ordenados por fecha de creación | **Pendiente** | No se registra fecha de creación |

**5. Historias de Usuario (15 Historias)**

**HU01 - Gestión Básica de Productos**

Como administrador  
Quiero agregar, modificar y eliminar productos  
Para mantener el inventario actualizado

Criterios de Aceptación:

* RF01: Puedo crear productos usando constructores con todos los atributos requeridos
* RF04: Los nuevos productos se registran correctamente en el sistema
* RF08: Puedo eliminar productos marcándolos como null en el array
* RF09: Puedo modificar nombres con validación de campo no vacío
* RF10: Puedo actualizar precios con validación de valor positivo
* RF11: Puedo ajustar cantidades con validación de no negativos

**HU02 - Búsqueda y Filtrado Avanzado**

Como empleado de almacén  
Quiero buscar productos por múltiples criterios  
Para encontrar rápidamente los artículos necesarios

Criterios de Aceptación:

* RF05: La búsqueda por nombre encuentra productos coincidentes (búsqueda parcial)
* RF15: El filtrado por categoría muestra solo productos del tipo seleccionado
* RF21: Puedo filtrar por tipo específico (Económico/Estándar/Premium)
* RF25: Identifico productos con stock bajo (<5 unidades) automáticamente

**HU03 - Reportes e Inventario**

Como gerente de tienda  
Quiero generar reportes del inventario  
Para tomar decisiones de reposición y promociones

Criterios de Aceptación:

* RF06: El listado completo muestra todos los productos activos
* RF12: El sistema cuenta correctamente los productos existentes
* RF22: Calcula el valor total del inventario (precio x cantidad)
* RF23/24: Identifica el producto más caro y más barato
* RF38: Genera estadísticas de promedios y totales por categoría

**HU04 - Clasificación de Productos**

Como jefe de compras  
Quiero clasificar productos por categorías  
Para optimizar el surtido y precios

Criterios de Aceptación:

* RF07: La herencia Producto → {Económico, Estándar, Premium} funciona correctamente
* RF15: Los filtros por categoría son precisos
* RF28: El conteo por categoría usa instanceof correctamente
* RF30: Muestra información específica de cada tipo de herencia

**HU05 - Interfaz de Usuario**

Como usuario del sistema  
Quiero una interfaz intuitiva  
Para realizar operaciones fácilmente

Criterios de Aceptación:

* RF13: El menú principal muestra todas las opciones disponibles
* RF14: Valida correctamente las entradas del usuario
* RF39: Solicita confirmación antes de salir del sistema
* RF40: Mantiene el menú activo hasta que el usuario decida salir

**HU06 - Ordenamiento de Productos**

Como encargado de visual merchandising  
Quiero ordenar productos  
Para presentarlos mejor a los clientes

Criterios de Aceptación:

* RF26: Ordena alfabéticamente usando algoritmo burbuja
* RF27: Ordena por precio (menor a mayor) correctamente

**HU07 - Validación de Datos**

Como administrador de sistema  
Quiero validaciones robustas  
Para mantener la integridad de los datos

Criterios de Aceptación:

* RF02: Los setters validan datos antes de asignarlos
* RF29: Respeta la capacidad máxima del array
* RF33: Rechaza nombres vacíos en constructores
* RF34: Rechaza precios negativos o cero
* RF35: Rechaza cantidades negativas

**HU08 - Mantenimiento del Array**

Como técnico de sistemas  
Quiero operaciones eficientes con el array  
Para optimizar el rendimiento

Criterios de Aceptación:

* RF04: Registra productos en la primera posición disponible
* RF36: Encuentra posiciones disponibles correctamente
* RF37: Compacta el array eliminando nulls (pendiente)

**HU09 - Información de Productos**

Como vendedor  
Quiero acceder a información completa  
Para atender mejor a los clientes

Criterios de Aceptación:

* RF03: Los getters devuelven valores correctos
* RF18: toString() muestra información completa
* RF19: El polimorfismo funciona en arrays de Producto
* RF20: equals() compara productos por nombre (parcial)

**HU10 - Constructores Especializados**

Como desarrollador  
Quiero constructores específicos  
Para crear objetos válidos desde el inicio

Criterios de Aceptación:

* RF31: Constructor por defecto inicializa con valores predeterminados
* RF32: Constructor con parámetros inicializa todos los atributos
* RF15-17: Constructores específicos para cada categoría de producto

**HU11 - Gestión de Categorías Premium**

Como gerente de línea premium  
Quiero manejar productos exclusivos  
Para mantener estándares de calidad

\*\*Criterios de Aceptación:

* RF17: ProductoPremium solo acepta precios >100
* RF30: Muestra claramente la categoría premium
* RF21: Filtra correctamente solo productos premium

**HU12 - Gestión de Ofertas**

Como encargado de promociones  
Quiero identificar productos económicos  
Para armar folletos de ofertas

\*\*Criterios de Aceptación:

* RF15: ProductoEconomico solo acepta precios <20
* RF21: Filtra correctamente productos económicos
* RF24: Identifica el producto más barato para destacar

**HU13 - Control de Inventario**

Como auditor  
Quiero verificar el inventario  
Para realizar controles periódicos

\*\*Criterios de Aceptación:

* RF06: Listado completo sin omisiones
* RF12: Conteo preciso de productos
* RF22: Valoración correcta del inventario
* RF25: Detección de stock bajo

**HU14 - Experiencia de Usuario**

Como cajero  
Quiero una interfaz amigable  
Para operar el sistema eficientemente

\*\*Criterios de Aceptación:

* RF13: Menú claro y accesible
* RF14: Prevención de errores en entradas
* RF39: Confirmación para acciones críticas
* RF40: Flujo continuo de trabajo

**HU15 - Mantenimiento del Sistema**

Como soporte técnico  
Quiero operaciones de mantenimiento  
Para optimizar el almacenamiento

\*\*Criterios de Aceptación:

* RF37: Compactación del array (pendiente)
* RF29: Control de capacidad máxima
* RF36: Gestión de posiciones disponibles
* RF08: Eliminación limpia de productos

Diagrama

El contenido generado por IA puede ser incorrecto.**6. Evidencia Grafica**

Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

**7. Criterios de aceptación**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Criterio de Aceptación** | **Estado** | **Evidencia** |
| CA01 | El objeto se crea correctamente con todos los atributos | Verificado | Constructor de Producto inicializa todos los campos |
| CA02 | Los setters validan datos antes de asignar | En proceso | Validaciones básicas pero sin setters formales |
| CA03 | Los productos se almacenan en la primera posición disponible | Verificado | ArrayList.add() garantiza almacenamiento correcto |
| CA04 | Encuentra productos por nombre (búsqueda parcial) | Verificado | Uso de contains() en buscarProductoPorNombre() |
| CA05 | Muestra todos los productos no nulos | Verificado | actualizarTabla() recorre ArrayList completo |
| CA06 | Eliminación con confirmación | Verificado | JOptionPane.showConfirmDialog implementado |
| CA07 | Actualiza cantidad con validación | Verificado | actualizarStock() verifica entrada numérica |
| CA08 | Filtrado por categoría preciso | Verificado | mostrarPorCategoria() compara Strings ignorando case |
| CA09 | Cálculo correcto de valor total | Verificado | Implementado en lógica de interfaz |
| CA10 | Ordenamiento por precio funcionando | Verificado | Collections.sort() con Comparator |
| CA11 | Búsqueda por código exacto | Verificado | buscarPorCodigo() con equals() |
| CA12 | Interfaz muestra opciones y procesa selección | Verificado | Menú principal con ActionListeners |

1. **Lecciones aprendidas**
2. **Diseño de Interfaces Gráficas:**
   * Aprendimos que Swing requiere una planificación cuidadosa del layout
   * Los JOptionPane son útiles para diálogos rápidos pero limitados
   * La separación entre lógica y presentación es crucial
3. **Manejo de Colecciones:**
   * ArrayList simplificó muchas operaciones comparado con arrays tradicionales
   * Los algoritmos de ordenamiento integrados (Collections.sort) ahorraron tiempo
   * La búsqueda con streams (filtros) podría mejorar el rendimiento
4. **Validaciones y Robustez:**
   * Implementar validaciones desde el inicio evita problemas posteriores
   * Los try-catch para entradas numéricas son esenciales en interfaces gráficas
   * Las confirmaciones para acciones destructivas mejoran la experiencia
5. **Trabajo en Equipo:**
   * Git fue fundamental para el control de versiones colaborativo
   * La asignación clara de módulos evitó conflictos
   * Las revisiones de código mejoraron la calidad general
6. **Documentación:**
   * Los comentarios Javadoc facilitaron el mantenimiento
   * Los diagramas UML iniciales ahorraron tiempo de desarrollo
   * Un registro de cambios detallado ayudó en la depuración
7. **Conclusiones grupal**

**Logros Principales:**

1. Implementamos exitosamente el sistema cumpliendo con el 70% de los requerimientos funcionales
2. La interfaz gráfica resultó intuitiva y cumplió con las expectativas de usabilidad
3. El sistema demostró ser estable en operaciones CRUD básicas
4. Aplicamos efectivamente los principios de POO aprendidos en clase

**Desafíos Encontrados:**

1. La gestión de estados en la interfaz gráfica fue más compleja de lo anticipado
2. Tuvimos que ajustar algunos requerimientos por limitaciones de tiempo
3. La falta de experiencia previa con Swing generó una curva de aprendizaje pronunciada

**Áreas de Mejora Identificadas:**

1. **Persistencia de Datos:** Implementar almacenamiento en archivos o base de datos
2. **Pruebas Automatizadas:** Incorporar JUnit para validar funcionalidades críticas
3. **Patrones de Diseño:** Aplicar MVC para mejor separación de responsabilidades
4. **Internacionalización:** Preparar el sistema para soporte multiidioma

**Impacto del Proyecto:**  
Este trabajo nos permitió integrar conocimientos teóricos con desarrollo práctico, evidenciando que:

* Los conceptos básicos de POO son fundamentales para sistemas robustos
* La planificación inicial ahorra tiempo en etapas posteriores
* El trabajo colaborativo bien organizado multiplica la productividad

**Compromisos Futuros:**

1. Completar los requerimientos pendientes en una siguiente iteración
2. Refactorizar el código para mejorar mantenibilidad
3. Documentar casos de prueba para futuras extensiones
4. Implementar un sistema de logging para diagnóstico de errores