

Semana 4

Desarrollo Orientado a Objetos II (PRY2203)

Formato de respuesta

| **Nombre estudiante:** | LUIS EVANGELO REBOLLEDO CORVALAN  NICOLAS CAVIERES CARVAJAL |
| --- | --- |
| **Asignatura:** | Desarrollo Orientado a Objetos II |
| **Profesor:** | FRANCESCO EDUARDO TOSSI BRANTE |

# Desarrollo de la actividad

**Paso 1: Requisitos funcionales y no funcionales**

|  |  |
| --- | --- |
| **Requisitos funcionales** | **Requisitos no funcionales** |
| * Agregar productos al inventario con código único, nombre, descripción, precio y stock | * Validación de datos (precios o stock no pueden ser menores a 0, códigos no vacíos y únicos, etc) |
| * Actualizar información de productos existentes | * Robustez ante entradas inválidas por consola (manejo de excepciones) |
| * Eliminar productos por código | * Código modular y probado |
| * Buscar productos por nombre o descripción | * Eficiencia razonable (búsqueda por nombre/desc en memoria, aprovechamiento de colecciones Java eficientes como HashMap, ArrayList, List, Optional, etc) |
| * Listar todos los productos disponibles | * Tener comentarios y documentación que facilite el debugeo y mantenibilidad |
| * Generar informe de inventario | * La interfaz de usuario debe ser intuitiva y simple |
| * Interfaz de usuario por consola que permita ejecutar, agregar, actualizar, eliminar, buscar, listar y salir |  |

**Paso 2: Fase de diseño**

**2.1 Clase Producto**

|  |
| --- |
| public class Producto {  private String codigo;  private String nombre;  private String descripcion;  private double precio;  private int stock;  public Producto(String codigo, String nombre, String descripcion, double precio, int stock) {  setCodigo(codigo);  setNombre(nombre);  setDescripcion(descripcion);  setPrecio(precio);  setStock(stock);  }  public String getCodigo() { return codigo; }  public void setCodigo(String codigo) {  if (codigo == null || codigo.trim().isEmpty()) {  throw new IllegalArgumentException("El código no puede estar vacío.");  }  this.codigo = codigo.trim();  }  public String getNombre() { return nombre; }  public void setNombre(String nombre) {  if (nombre == null || nombre.trim().isEmpty()) {  throw new IllegalArgumentException("El nombre no puede estar vacío.");  }  this.nombre = nombre.trim();  }  public String getDescripcion() { return descripcion; }  public void setDescripcion(String descripcion) {  this.descripcion = (descripcion == null) ? "" : descripcion.trim();  }  public double getPrecio() { return precio; }  public void setPrecio(double precio) {  if (precio < 0) throw new IllegalArgumentException("El precio no puede ser negativo.");  this.precio = precio;  }  public int getStock() { return stock; }  public void setStock(int stock) {  if (stock < 0) throw new IllegalArgumentException("El stock no puede ser negativo.");  this.stock = stock;  }  // actualiza el precio validando no-negatividad  public void actualizarPrecio(double nuevoPrecio) {  setPrecio(nuevoPrecio);  }  // Fija el stock a un valor válido (>=0)  public void actualizarStock(int nuevoStock) {  setStock(nuevoStock);  }  // ajusta en delta (puede ser negativo) y valida que no quede < 0  public void ajustarStock(int delta) {  int nuevo = this.stock + delta;  if (nuevo < 0) throw new IllegalArgumentException("El ajuste dejaría stock negativo.");  this.stock = nuevo;  }  // utilidad para búsqueda por nombre/descripcion  public boolean coincideCon(String query) {  if (query == null || query.trim().isEmpty()) return false;  String q = query.toLowerCase();  return nombre.toLowerCase().contains(q) || descripcion.toLowerCase().contains(q);  }  @Override public boolean equals(Object o) {  if (this == o) return true;  if (!(o instanceof Producto)) return false;  Producto that = (Producto) o;  return Objects.equals(codigo, that.codigo);  }  @Override public int hashCode() {  return Objects.hash(codigo);  }  @Override public String toString() {  return String.format("[%s] %s | %s | $%.2f | stock: %d", codigo, nombre, descripcion, precio, stock);  }  } |

**2.2 Clase Inventario**

|  |
| --- |
| **public class Inventario {**  **private final Map<String, Producto> productos = new HashMap<>();**  **// agrega si el producto es válido y el código no existe**  **public boolean agregarProducto(Producto p) {**  **if (p == null) throw new IllegalArgumentException("No se puede agregar un producto nulo.");**  **String codigo = p.getCodigo();**  **if (productos.containsKey(codigo)) return false;**  **productos.put(codigo, p);**  **return true;**  **}**  **// elimina y retorna el producto, o null si no existía**  **public Producto eliminarProducto(String codigo) {**  **if (codigo == null) return null;**  **return productos.remove(codigo);**  **}**  **public Optional<Producto> buscarPorCodigo(String codigo) {**  **if (codigo == null) return Optional.empty();**  **return Optional.ofNullable(productos.get(codigo));**  **}**  **// busca por nombre o descripción**  **public List<Producto> buscarPorNombreODescripcion(String query) {**  **if (query == null || query.trim().isEmpty()) return Collections.emptyList();**  **String q = query.toLowerCase();**  **return productos.values().stream()**  **.filter(p -> p.getNombre().toLowerCase().contains(q) ||**  **p.getDescripcion().toLowerCase().contains(q))**  **.sorted(Comparator.comparing(Producto::getCodigo))**  **.collect(Collectors.toList());**  **}**  **// lista todos productos ordenados por código**  **public List<Producto> listarTodos() {**  **return productos.values().stream()**  **.sorted(Comparator.comparing(Producto::getCodigo))**  **.collect(Collectors.toList());**  **}**  **// reemplaza un producto existente por código**  **public boolean actualizarProducto(Producto p) {**  **if (p == null) throw new IllegalArgumentException("Producto nulo.");**  **String codigo = p.getCodigo();**  **if (!productos.containsKey(codigo)) return false;**  **productos.put(codigo, p);**  **return true;**  **}**  **public boolean actualizarPrecio(String codigo, double nuevoPrecio) {**  **Optional<Producto> op = buscarPorCodigo(codigo);**  **if (!op.isPresent()) return false;**  **op.get().actualizarPrecio(nuevoPrecio);**  **return true;**  **}**  **public boolean actualizarStock(String codigo, int nuevoStock) {**  **Optional<Producto> op = buscarPorCodigo(codigo);**  **if (!op.isPresent()) return false;**  **op.get().actualizarStock(nuevoStock);**  **return true;**  **}**  **// informe tabulado simple**  **public String generarInforme() {**  **StringBuilder sb = new StringBuilder();**  **sb.append(String.format("%-10s %-20s %-10s %-8s %-6s%n", "CODIGO", "NOMBRE", "PRECIO", "STOCK", "DESC?"));**  **sb.append("------------------------------------------------------------------\n");**  **for (Producto p : listarTodos()) {**  **sb.append(String.format("%-10s %-20s %-10.2f %-8d %-6s%n",**  **p.getCodigo(), p.getNombre(), p.getPrecio(), p.getStock(),**  **(p.getDescripcion() != null && !p.getDescripcion().isEmpty()) ? "si" : "no"));**  **}**  **return sb.toString();**  **}**  **}** |

**Paso 3: Implementación**

**3.1 Interacción con el usuario:**

**3.2 Gestión de productos:**

**3.3 Búsqueda y listado de productos:**

|  |
| --- |
| public class MenuPrincipal {    private final Inventario inventario;  private final Scanner scanner;    public MenuPrincipal() {  this.inventario = new Inventario();  this.scanner = new Scanner(System.in);  precargarEjemplo();  }  private void precargarEjemplo() {  inventario.agregarProducto(new Producto("A001", "Teclado", "Teclado mecánico", 29990, 15));  inventario.agregarProducto(new Producto("A002", "Mouse", "Mouse óptico", 9990, 40));  inventario.agregarProducto(new Producto("A003", "Monitor", "24 pulgadas", 129990, 8));  }  public void menu() {  int opcion;  do {  mostrarMenu();  opcion = leerEntero("Selecciona una opción: ");  try {  switch (opcion) {  case 1: agregarProducto(); break;  case 2: actualizarProducto(); break;  case 3: eliminarProducto(); break;  case 4: buscarProductos(); break;  case 5: listarProductos(); break;  case 6: mostrarInforme(); break;  case 0: System.out.println("Saliendo... Gracias!"); break;  default: System.out.println("Opción no válida");  }  } catch (Exception ex) {  System.out.println("Error: " + ex.getMessage());  }  System.out.println();  } while (opcion != 0);  }  private void mostrarMenu() {  System.out.println("\*\*\* GESTIÓN DE INVENTARIO \*\*\*");  System.out.println("1. Agregar producto");  System.out.println("2. Actualizar producto");  System.out.println("3. Eliminar producto");  System.out.println("4. Buscar porductos (nombre/descripcion)");  System.out.println("5. Listar productos");  System.out.println("6. Mostrar informe");  System.out.println("0. Salir");  }  private void agregarProducto() {  System.out.println(">> Agregar producto");  String codigo = leerTexto("Código: ");  String nombre = leerTexto("Nombre: ");  String descripcion = leerTexto("Descripción: ");  double precio = leerDouble("Precio: ");  int stock = leerEntero("Stock: ");  boolean ok = inventario.agregarProducto(new Producto(codigo, nombre, descripcion, precio, stock));  System.out.println(ok ? "Producto agregado." : "Ya existe un producto con ese código.");  }  private void actualizarProducto() {  System.out.println(">> Actualizar producto por código");  String codigo = leerTexto("Código: ");  Optional<Producto> op = inventario.buscarPorCodigo(codigo);  if (!op.isPresent()) {  System.out.println("No existe producto con ese código.");  return;  }  Producto p = op.get();  System.out.println("Actualizando: " + p);  String nombre = leerTextoOpcional("Nuevo nombre (ENTER para mantener): ");  if (!nombre.isEmpty()) p.setNombre(nombre);  String descripcion = leerTextoOpcional("Nueva descripción (ENTER para mantener): ");  if (!descripcion.isEmpty()) p.setDescripcion(descripcion);  String precioStr = leerTextoOpcional("Nuevo precio (ENTER para mantener): ");  if (!precioStr.isEmpty()) p.setPrecio(Double.parseDouble(precioStr));  String stockStr = leerTextoOpcional("Nuevo stock (ENTER para mantener): ");  if (!stockStr.isEmpty()) p.setStock(Integer.parseInt(stockStr));  boolean ok = inventario.actualizarProducto(p);  System.out.println(ok ? "Producto actualizado." : "No se pudo actualizar.");  }  private void eliminarProducto() {  System.out.println(">> Eliminar producto");  String codigo = leerTexto("Código: ");  Producto eliminado = inventario.eliminarProducto(codigo);  System.out.println(eliminado != null ? "Eliminado: " + eliminado : "No existía ese código.");  }  private void buscarProductos() {  System.out.println(">> Búsqueda");  String query = leerTexto("Texto a buscar (nombre/descripcion): ");  List<Producto> resultados = inventario.buscarPorNombreODescripcion(query);  if (resultados.isEmpty()) {  System.out.println("Sin resultados.");  } else {  resultados.forEach(System.out::println);  }  }  private void listarProductos() {  System.out.println(">> Listado completo");  List<Producto> lista = inventario.listarTodos();  if (lista.isEmpty()) {  System.out.println("Inventario vacío.");  } else {  lista.forEach(System.out::println);  }  }  private void mostrarInforme() {  System.out.println(">> Informe de Inventario");  System.out.println(inventario.generarInforme());  }  private String leerTexto(String prompt) {  System.out.print(prompt);  String s = scanner.nextLine();  while (s == null || s.trim().isEmpty()) {  System.out.print("Valor requerido. " + prompt);  s = scanner.nextLine();  }  return s.trim();  }  private String leerTextoOpcional(String prompt) {  System.out.print(prompt);  String s = scanner.nextLine();  return (s == null) ? "" : s.trim();  }  private int leerEntero(String prompt) {  while (true) {  System.out.print(prompt);  String s = scanner.nextLine();  try { return Integer.parseInt(s.trim()); }  catch (Exception e) { System.out.println("Debe ser un número entero"); }  }  }  private double leerDouble(String prompt) {  while (true) {  System.out.print(prompt);  String s = scanner.nextLine();  try { return Double.parseDouble(s.trim()); }  catch (Exception e) { System.out.println("Debe ser un número (use punto decimal)"); }  }  }  public static void main(String[] args) {  new MenuPrincipal().menu();  }  } |

**Paso 4: Fase de pruebas**

**4.1 Pruebas de integración**

|  |
| --- |
| public class IntegracionProductoInventarioTest {    @Test  void flujoBasico\_agregar\_buscar\_eliminar\_y\_coherencia() {  Inventario inv = new Inventario();   // Agregar  Producto p = new Producto("K1", "Cargador", "USB-C", 7990, 10);  assertTrue(inv.agregarProducto(p));   // Buscar  Optional<Producto> buscado = inv.buscarPorCodigo("K1");  assertTrue(buscado.isPresent());  assertEquals("Cargador", buscado.get().getNombre());   // Actualizar  assertTrue(inv.actualizarPrecio("K1", 8990));  assertTrue(inv.actualizarStock("K1", 12));  Producto actualizado = inv.buscarPorCodigo("K1").get();  assertEquals(8990, actualizado.getPrecio(), 0.0001);  assertEquals(12, actualizado.getStock());   // Eliminar  Producto eliminado = inv.eliminarProducto("K1");  assertNotNull(eliminado);   // Coherencia del estado  assertFalse(inv.buscarPorCodigo("K1").isPresent());  } } |

**4.2 Pruebas unitarias**

Texto

El contenido generado por IA puede ser incorrecto.



Reservados todos los derechos Fundación Instituto Profesional Duoc UC. No se permite copiar, reproducir, reeditar, descargar, publicar, emitir, difundir, de forma total o parcial la presente obra, ni su incorporación a un sistema informático, ni su transmisión en cualquier forma o por cualquier medio (electrónico, mecánico, fotocopia, grabación u otros) sin autorización previa y por escrito de Fundación Instituto Profesional Duoc UC La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.