

Semana 5

Desarrollo Orientado a Objetos II (PRY2203)

Formato de respuesta

Nombre estudiante:	Nicolas Cavieres
Asignatura:	Desarrollo Orientado a Objetos II
Profesor:	Francesco Tossi
Fecha:	16/09/2025

Descripción de la actividad

Instrucciones específicas

1. Fase de análisis

Requisitos funcionales	Requisitos no funcionales
Crear productos según las especificaciones, como nombre, precio, categoría, etc.	Tener una interfaz clara y fácil de usar. Con mensajes e instrucciones claras para el usuario en todo su recorrido.
Actualizar los productos.	Mostrar mensajes de confirmación al usuario.
Evitar la creación de productos duplicados o inconsistentes con el modelo de negocio.	Tener un desarrollo que permita escalar, mantener y modificar el programa a futuro.
Gestionar entradas inválidas o inconsistentes.	Tener una buena organización de carpetas.
Agregar y eliminar productos de un inventario.	Permitir ejecutarse en cualquier plataforma con Java.
Listar los productos del inventario.	Permitir agregar nuevas funciones.
Generar un informe.	Estar documentado en su etapa de desarrollo y pruebas.
Permitir cerrar la aplicación de forma segura.	

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();
}
}

```

3. Implementación

Clase 'MenuPrincipal'

```

public class MenuPrincipal {
    private final Inventario inventario;
    private final Scanner scanner;

    public MenuPrincipal(Inventario inventario) {
        this.inventario = inventario;
        this.scanner = new Scanner(System.in);
    }

    public void ejecutar() {
        int opcion;
        do {
            mostrarMenu();
            opcion = leerEntero("Seleccione una opción: ");
            switch (opcion) {
                case 1 -> agregarProducto();
                case 2 -> actualizarProducto();
                case 3 -> eliminarProducto();
                case 4 -> buscarProducto();
                case 5 -> listarProductos();
                case 6 -> mostrarInforme();
                case 0 -> System.out.println(";Hasta pronto!");
                default -> System.out.println("Opción no válida.");
            }
        } while (opcion != 0);
    }

    private void mostrarMenu() {
        System.out.println("\n--- MENÚ 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 producto");
        System.out.println("5. Listar todos los productos");
        System.out.println("6. Informe de inventario");
        System.out.println("0. Salir");
    }

    private int leerEntero(String mensaje) {

```

```

while (true) {
    System.out.print(mensaje);
    try {
        return Integer.parseInt(scanner.nextLine());
    } catch (NumberFormatException ex) {
        System.out.println("Ingrese un número válido.");
    }
}

private double leerDouble(String mensaje) {
    while (true) {
        System.out.print(mensaje);
        try {
            return Double.parseDouble(scanner.nextLine());
        } catch (NumberFormatException ex) {
            System.out.println("Ingrese un número válido.");
        }
    }
}

private void agregarProducto() {
    try {
        System.out.print("Código: ");
        String codigo = scanner.nextLine();
        System.out.print("Nombre: ");
        String nombre = scanner.nextLine();
        System.out.print("Descripción: ");
        String descripcion = scanner.nextLine();
        double precio = leerDouble("Precio: ");
        int stock = leerEntero("Stock: ");
        Producto p = new Producto(codigo, nombre, descripcion, precio, stock);
        if (inventario.agregarProducto(p)) {
            System.out.println("Producto agregado correctamente.");
        } else {
            System.out.println("Ya existe un producto con ese código.");
        }
    } catch (IllegalArgumentException ex) {
        System.out.println("Error: " + ex.getMessage());
    }
}

private void actualizarProducto() {
    System.out.print("Código del producto a actualizar: ");
    String codigo = scanner.nextLine();
    Optional<Producto> op = inventario.buscarPorCodigo(codigo);
    if (op.isEmpty()) {
        System.out.println("No existe producto con ese código.");
        return;
    }
    Producto p = op.get();
    System.out.println("Producto actual: " + p);
    System.out.println("1. Cambiar precio");
    System.out.println("2. Cambiar stock");
    System.out.println("3. Cambiar nombre y descripción");
    System.out.print("Seleccione campo a actualizar: ");
    String opcion = scanner.nextLine();
    switch (opcion) {
        case "1" -> {
            double nuevoPrecio = leerDouble("Nuevo precio: ");
            if (inventario.actualizarPrecio(codigo, nuevoPrecio))
                System.out.println("Precio actualizado.");
            else
                System.out.println("Error al actualizar precio.");
        }
        case "2" -> {
            int nuevoStock = leerEntero("Nuevo stock: ");

```

```

        if (inventario.actualizarStock(codigo, nuevoStock))
            System.out.println("Stock actualizado.");
        else
            System.out.println("Error al actualizar stock.");
    }
    case "3" -> {
        System.out.print("Nuevo nombre: ");
        String nuevoNombre = scanner.nextLine();
        System.out.print("Nueva descripción: ");
        String nuevaDesc = scanner.nextLine();
        Producto nuevo = new Producto(codigo, nuevoNombre, nuevaDesc, p.getPrecio(), p.getStock());
        if (inventario.actualizarProducto(nuevo))
            System.out.println("Nombre y descripción actualizados.");
        else
            System.out.println("Error al actualizar.");
    }
    default -> System.out.println("Opción inválida.");
}
}

private void eliminarProducto() {
    System.out.print("Código del producto a eliminar: ");
    String codigo = scanner.nextLine();
    Producto eliminado = inventario.eliminarProducto(codigo);
    if (eliminado != null) {
        System.out.println("Producto eliminado: " + eliminado);
    } else {
        System.out.println("No existe producto con ese código.");
    }
}

private void buscarProducto() {
    System.out.print("Ingrese nombre o descripción a buscar: ");
    String query = scanner.nextLine();
    List<Producto> resultados = inventario.buscarPorNombreODescripcion(query);
    if (resultados.isEmpty()) {
        System.out.println("No se encontraron productos.");
    } else {
        resultados.forEach(System.out::println);
    }
}

private void listarProductos() {
    List<Producto> productos = inventario.listarTodos();
    if (productos.isEmpty()) {
        System.out.println("Inventario vacío.");
    } else {
        productos.forEach(System.out::println);
    }
}

private void mostrarInforme() {
    System.out.println("\n\nINFORME DE INVENTARIO");
    System.out.println(inventario.generarInforme());
}
}

```


4. Fase de pruebas

4.1 Pruebas de integración

...

```
public class InventarioIntegracionTest {  
    @Test  
    void testAgregarBuscarEliminarProducto() {  
        Inventario inv = new Inventario();  
  
        // Agregar  
        Producto p1 = new Producto("A1", "Mouse", "Mouse óptico", 4000, 5);  
        assertTrue(inv.agregarProducto(p1));  
  
        // Buscar por código  
        assertTrue(inv.buscarPorCodigo("A1").isPresent());  
        assertEquals("Mouse", inv.buscarPorCodigo("A1").get().getNombre());  
  
        // Buscar por nombre/descripción  
        List<Producto> encontrados = inv.buscarPorNombreODescripcion("mouse");  
        assertFalse(encontrados.isEmpty());  
        assertEquals("A1", encontrados.get(0).getCodigo());  
  
        // Eliminar  
        Producto eliminado = inv.eliminarProducto("A1");  
        assertNotNull(eliminado);  
        assertFalse(inv.buscarPorCodigo("A1").isPresent());  
  
        // Estado final: Inventario vacío  
        assertTrue(inv.listarTodos().isEmpty());  
    }  
}
```

}

...

4.2 Pruebas unitarias

...

```
public class InventarioTest {

    @Test
    void testAgregarProducto() {
        Inventario inv = new Inventario();
        Producto p = new Producto("C1", "Cable", "Cable USB-C", 2000, 25);
        assertTrue(inv.agregarProducto(p));
        // Producto duplicado
        assertFalse(inv.agregarProducto(p));
        // Producto nulo
        assertThrows(IllegalArgumentException.class, () -> inv.agregarProducto(null));
    }

    @Test
    void testEliminarProducto() {
        Inventario inv = new Inventario();
        Producto p = new Producto("C2", "Disco Duro", "SSD", 40000, 3);
        inv.agregarProducto(p);
        assertNotNull(inv.eliminarProducto("C2"));
        // Eliminar inexistente
        assertNull(inv.eliminarProducto("ZZZ"));
    }

    @Test
    void testBuscarPorNombreODescripcion() {
        Inventario inv = new Inventario();
        inv.agregarProducto(new Producto("C3", "Tablet", "Tablet gráfica", 55000, 8));
        inv.agregarProducto(new Producto("C4", "Tablet", "No gráfica", 35000, 2));
        // Buscar por nombre
        List<Producto> res = inv.buscarPorNombreODescripcion("tablet");
        assertEquals(2, res.size());
        // Buscar inexistente
        assertTrue(inv.buscarPorNombreODescripcion("notebook").isEmpty());
    }

    @Test
    void testListarTodos() {
        Inventario inv = new Inventario();
```

```

    Producto p1 = new Producto("C5", "Parlante", "", 8000, 9);
    Producto p2 = new Producto("C6", "Pendrive", "", 3000, 15);
    inv.agregarProducto(p1);
    inv.agregarProducto(p2);
    List<Producto> lista = inv.listarTodos();
    assertEquals(2, lista.size());
    assertEquals("C5", lista.get(0).getCodigo());
    assertEquals("C6", lista.get(1).getCodigo());
}
}
...

...

public class ProductoTest {

    @Test
    void testCreacionProducto() {
        Producto p = new Producto("B2", "Teclado", "Teclado mecánico", 9900, 10);
        assertEquals("B2", p.getCodigo());
        assertEquals("Teclado", p.getNombre());
        assertEquals("Teclado mecánico", p.getDescripcion());
        assertEquals(9900, p.getPrecio());
        assertEquals(10, p.getStock());
    }

    @Test
    void testActualizarAtributos() {
        Producto p = new Producto("B3", "Monitor", "", 25000, 4);
        p.actualizarPrecio(30000);
        assertEquals(30000, p.getPrecio());
        p.actualizarStock(7);
        assertEquals(7, p.getStock());
    }
}
...

```

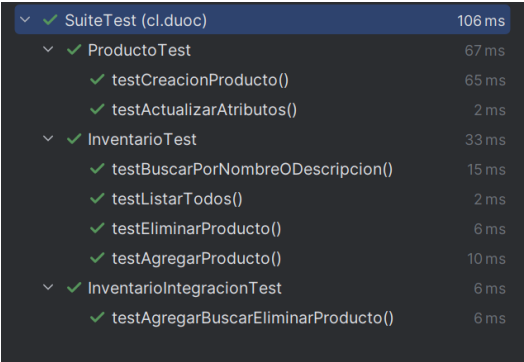
Notas: He dejado fuera de la copia de los códigos los imports y la SuiteTest que se encarga de llamar todos los test listados arriba, pero pueden ser revisados en el enlace [git](#).

4.2 Evidencias

Desde el terminal

```
Terminal Local x + v
[INFO] Scanning for projects...
[INFO]
[INFO] -----< cl.duoc.demo:GestionInventario >-----
[INFO] Building GestionInventario 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ GestionInventario ---
[INFO] Deleting C:\Users\nikoc\OneDrive\Documentos\DuocUC\Semestre3\P00 II\Proyectos\Semana5\GestionInventario\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ GestionInventario ---
[INFO] skip non existing resourceDirectory C:\Users\nikoc\OneDrive\Documentos\DuocUC\Semestre3\P00 II\Proyectos\Semana5\GestionInventario\src\main\resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ GestionInventario ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 4 source files with javac [debug target 22] to target\classes
[WARNING] location of system modules is not set in conjunction with -source 22
not setting the location of system modules may lead to class files that cannot run on JDK 22
--release 22 is recommended instead of -source 22 -target 22 because it sets the location of system modules automatically
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ GestionInventario ---
[INFO] skip non existing resourceDirectory C:\Users\nikoc\OneDrive\Documentos\DuocUC\Semestre3\P00 II\Proyectos\Semana5\GestionInventario\src\test\resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ GestionInventario ---
[INFO] Recompiling the module because of changed dependency.
[INFO] Compiling 4 source files with javac [debug target 22] to target\test-classes
[WARNING] location of system modules is not set in conjunction with -source 22
not setting the location of system modules may lead to class files that cannot run on JDK 22
--release 22 is recommended instead of -source 22 -target 22 because it sets the location of system modules automatically
[INFO]
[INFO] --- surefire:3.2.5:test (default-test) @ GestionInventario ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running cl.duoc.modelo.InventarioIntegracionTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.090 s -- in cl.duoc.modelo.InventarioIntegracionTest
[INFO] Running cl.duoc.modelo.InventarioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 s -- in cl.duoc.modelo.InventarioTest
[INFO] Running cl.duoc.modelo.ProductoTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.008 s -- in cl.duoc.modelo.ProductoTest
[INFO] Running cl.duoc.SuiteTest
[INFO] Running cl.duoc.modelo.ProductoTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.017 s -- in cl.duoc.modelo.ProductoTest
[INFO] Running cl.duoc.modelo.InventarioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.021 s -- in cl.duoc.modelo.InventarioTest
[INFO] Running cl.duoc.modelo.InventarioIntegracionTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 s -- in cl.duoc.modelo.InventarioIntegracionTest
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.083 s -- in cl.duoc.SuiteTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.311 s
[INFO] Finished at: 2025-09-15T13:30:13-03:00
```

Desde el IDE.



A screenshot of an IDE's test results window. The background is dark. The text is white, with green checkmarks and green text for successful tests. The table lists the test suite and its sub-tests with their respective execution times.

✓ SuiteTest (cl.duoc)	106 ms
✓ ProductoTest	67 ms
✓ testCreacionProducto()	65 ms
✓ testActualizarAtributos()	2 ms
✓ InventarioTest	33 ms
✓ testBuscarPorNombreODescripcion()	15 ms
✓ testListarTodos()	2 ms
✓ testEliminarProducto()	6 ms
✓ testAgregarProducto()	10 ms
✓ InventarioIntegracionTest	6 ms
✓ testAgregarBuscarEliminarProducto()	6 ms

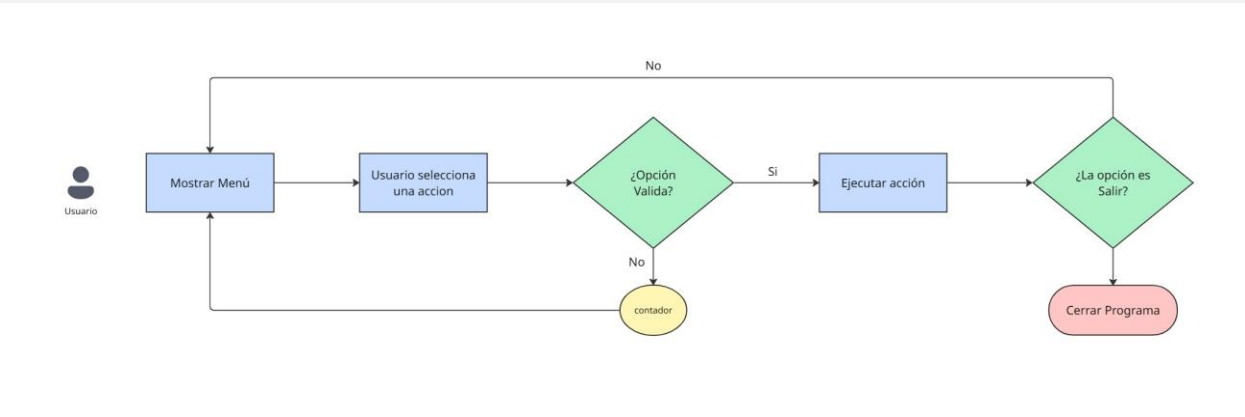
Paso 1: Roadmap con la herramienta Miro

[Enlace Miro](#)

Roadmap



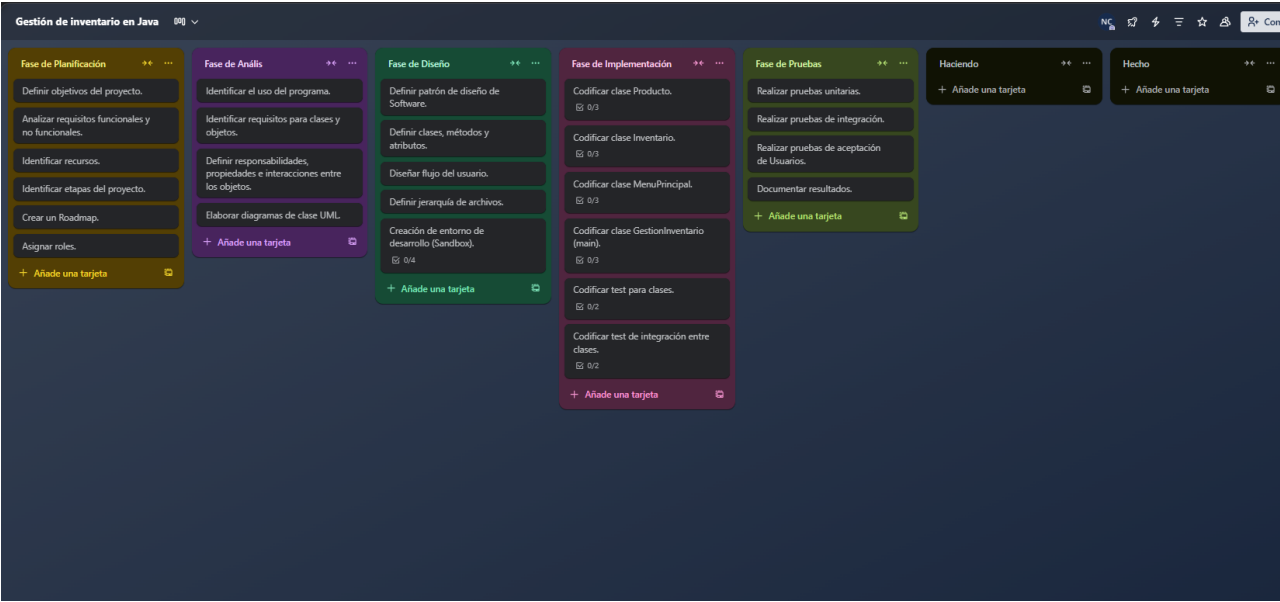
Flujo de Usuario



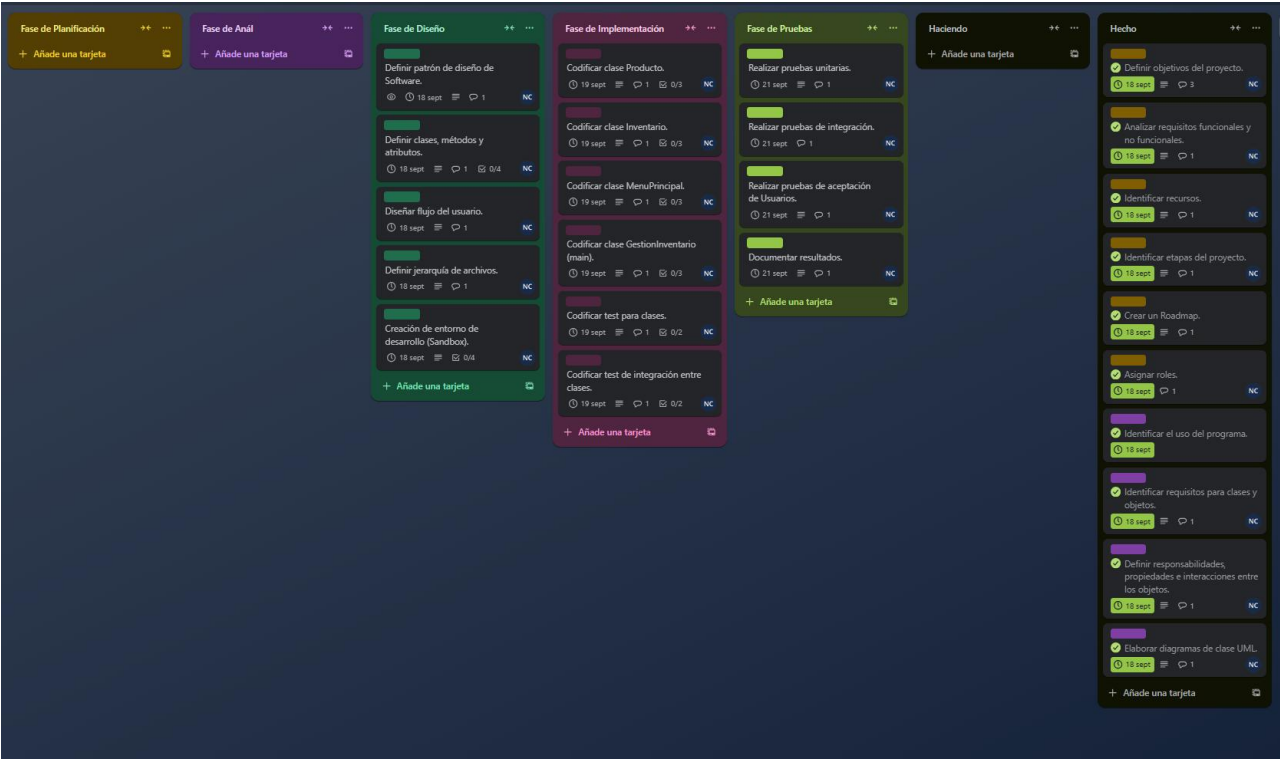
Paso 2: Implementa la herramienta colaborativa Trello

[Enlace Trello](#)

En desarrollo:



Con Tareas completadas





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.