

2025

FACULTAD DE INGENIERIA DE SISTEMAS COMPUTACIONALES

TEMA: EXAMEN T1

CURSO:

TECNICAS DE PROGRAMACION ORIE. OBJET.

DOCENTE:

ING. MARTIN EDUARDO TORRES RODRIGUEZ

ALUMNO:

ID UPN

- | | |
|--------------------------------|-----------|
| - SUCLUPE VELA STEVEN EDSON | N00254991 |
| - ORTEGA VALVERDE YULISSA | N00366132 |
| - GAYOSO ORDOÑEZ KEVIN BRYAN | N00298302 |
| - ESTRADA LLANOS ERICK LEONARD | N00322858 |
| - JESUS MARTIN QUINTANA SUAREZ | N00333353 |



TRUJILLO - 2025

INDICE

Caso 1 – Sistema de Biblioteca Virtual

1.1 Requerimientos

1.2 Distribución del proyecto

- Paquete: caso1_sistemaBiblioteca

- Clases: Autor, Libro, BibliotecaVirtual, Main

1.3 Código fuente

- Autor.java

- Libro.java

- BibliotecaVirtual.java

- Main.java

1.4 Compilación (Evidencia)

1.5 Diagrama UML

1.6 Preguntas de análisis

Caso 2 – Gestión de Estudiantes

2.1 Requerimientos

2.2 Distribución del proyecto

- Paquete: caso2_gestionEstudiantes

- Clases e interfaces: Evaluable, Estudiante, EstudianteRegular, EstudianteBecado, Curso, GestionEstudiantes, Main

2.3 Código fuente

- Evaluable.java

- Estudiante.java

- EstudianteRegular.java

- EstudianteBecado.java

- Curso.java

- GestionEstudiantes.java

- Main.java

2.4 Compilación (Evidencia)

2.5 Diagrama UML

2.6 Preguntas de análisis

Caso 3 – Sistema de Ventas en Línea

3.1 Requerimientos

3.2 Distribución del proyecto

- Paquete: caso3_sistemaVentas

- Clases: StockInsuficienteException, ProductoNoEncontradoException, Producto, ItemVenta, Venta, TiendaOnline, Main

3.3 Código fuente

- StockInsuficienteException.java
- ProductoNoEncontradoException.java
- Producto.java
- ItemVenta.java
- Venta.java
- TiendaOnline.java
- Main.java

3.4 Compilación (Evidencia)

3.5 Diagrama UML

3.6 Preguntas de análisis

 **Evidencia del Programa Eclipse**

 **Repositorio Git**

EXAMEN T1

Caso 1 – Sistema de Biblioteca Virtual

Una universidad necesita un sistema básico para registrar libros y autores en una biblioteca virtual. Requerimientos:

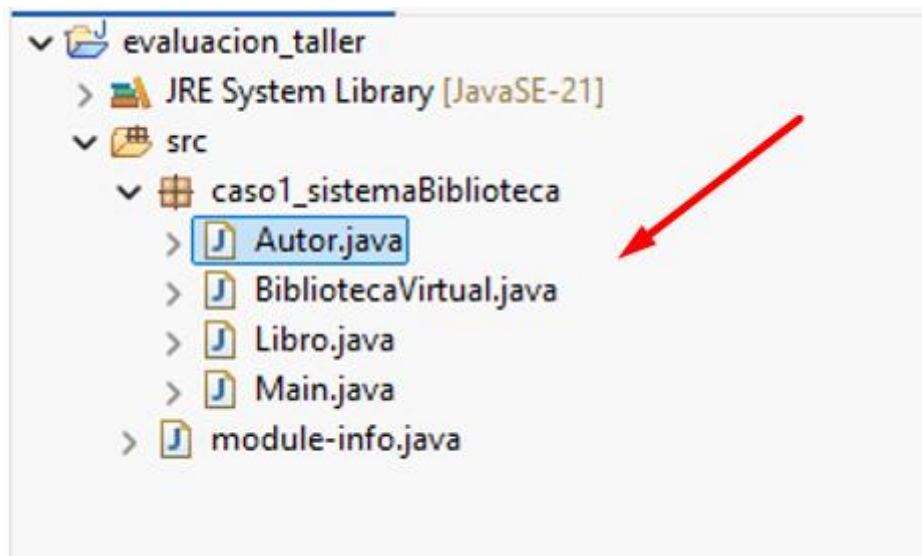
1. Crear las clases Libro y Autor con sus atributos principales.
2. Implementar métodos para:
 - Registrar un libro (título, autor, año).
 - Mostrar la información del libro.
3. Usar sobrecarga de métodos para permitir registrar un libro con o sin ISBN.
4. Manejar errores si se intenta registrar un libro sin título.

Distribución del proyecto:

Paquete: caso1_sistemaBiblioteca

Clases:

- Autor
- Libro
- Biblioteca
- Main



Autor.java

```
package caso1_sistemaBiblioteca;

public class Autor {

    private String nombre;

    private String apellido;

    private String nacionalidad;

    // Constructor

    public Autor(String nombre, String apellido, String nacionalidad) {

        this.nombre = nombre;

        this.apellido = apellido;

        this.nacionalidad = nacionalidad;

    }

    // Métodos getter y setter

    public String getNombre() {

        return nombre;

    }

    public void setNombre(String nombre) {

        this.nombre = nombre;

    }

    public String getApellido() {

        return apellido;

    }

    public void setApellido(String apellido) {

        this.apellido = apellido;

    }

    public String getNacionalidad() {

        return nacionalidad;

    }

}
```

```
}

public void setNacionalidad(String nacionalidad) {
    this.nacionalidad = nacionalidad;
}

public String getNombreCompleto() {
    return nombre + " " + apellido;
}

@Override
public String toString() {
    return nombre + " " + apellido + " (" + nacionalidad + ")";
}
}
```

Clase Autor.java (EVIDENCIA)

```
Autor.java X Libro.java BibliotecaVirtual.java Main.java
1 package caso1_sistemaBiblioteca;
2
3 public class Autor {
4     private String nombre;
5     private String apellido;
6     private String nacionalidad;
7
8     // Constructor
9     public Autor(String nombre, String apellido, String nacionalidad) {
10         this.nombre = nombre;
11         this.apellido = apellido;
12         this.nacionalidad = nacionalidad;
13     }
14
15     // Métodos getter y setter
16     public String getNombre() {
17         return nombre;
18     }
19
20     public void setNombre(String nombre) {
21         this.nombre = nombre;
22     }
23
24     public String getApellido() {
25         return apellido;
26     }
27
28     public void setApellido(String apellido) {
29         this.apellido = apellido;
30     }
31
32     public String getNacionalidad() {
33         return nacionalidad;
34     }
35
36     public void setNacionalidad(String nacionalidad) {
37         this.nacionalidad = nacionalidad;
38     }
39
40     public String getNombreCompleto() {
41         return nombre + " " + apellido;
42     }
43
44     @Override
45     public String toString() {
46         return nombre + " " + apellido + " (" + nacionalidad + ")";
47     }
48 }
```



Libro.java

```
package caso1_sistemaBiblioteca;

public class Libro {

    private String titulo;

    private Autor autor;

    private int año;

    private String isbn;

    // Constructor con ISBN (sobrecarga 1)
```

```
public Libro(String titulo, Autor autor, int año, String isbn) throws IllegalArgumentException {  
    if (titulo == null || titulo.trim().isEmpty()) {  
        throw new IllegalArgumentException("Error: No se puede registrar un libro sin título");  
    }  
    this.titulo = titulo;  
    this.autor = autor;  
    this.año = año;  
    this.isbn = isbn;  
}
```

// Constructor sin ISBN (sobrecarga 2)

```
public Libro(String titulo, Autor autor, int año) throws IllegalArgumentException {  
    if (titulo == null || titulo.trim().isEmpty()) {  
        throw new IllegalArgumentException("Error: No se puede registrar un libro sin título");  
    }  
    this.titulo = titulo;  
    this.autor = autor;  
    this.año = año;  
    this.isbn = "Sin ISBN";  
}
```

// Métodos getter y setter

```
public String getTitulo() {  
    return titulo;  
}
```

```
public void setTitulo(String titulo) {  
    if (titulo == null || titulo.trim().isEmpty()) {  
        throw new IllegalArgumentException("Error: El título no puede estar vacío");  
    }  
    this.titulo = titulo;  
}
```

```
public Autor getAutor() {
```



```
    return autor;
}

public void setAutor(Autor autor) {
    this.autor = autor;
}

public int getAño() {
    return año;
}

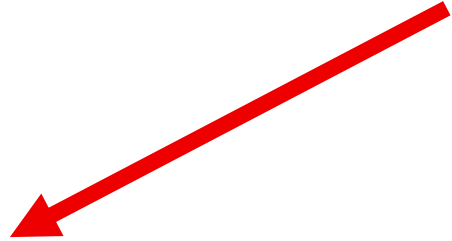
public void setAño(int año) {
    this.año = año;
}

public String getIsbn() {
    return isbn;
}

public void setIsbn(String isbn) {
    this.isbn = isbn;
}

// Método para mostrar información del libro
public void mostrarInformacion() {
    System.out.println("=== INFORMACIÓN DEL LIBRO ===");
    System.out.println("Título: " + titulo);
    System.out.println("Autor: " + autor.toString());
    System.out.println("Año: " + año);
    System.out.println("ISBN: " + isbn);
    System.out.println("=====\n");
}

@Override
```



```

public String toString() {

    return "\"" + titulo + "\" por " + autor.getNombreCompleto() + " (" + año + ")";

}

}

```

Clase Libro.java (EVIDENCIA)

```

1 package caso1_sistemaBiblioteca;
2
3 public class Libro {
4     private String titulo;
5     private Autor autor;
6     private int año;
7     private String isbn;
8
9     // Constructor con ISBN (sobrecarga 1)
10    public Libro(String titulo, Autor autor, int año, String isbn) throws IllegalArgumentException {
11        if (titulo == null || titulo.trim().isEmpty()) {
12            throw new IllegalArgumentException("Error: No se puede registrar un libro sin título");
13        }
14        this.titulo = titulo;
15        this.autor = autor;
16        this.año = año;
17        this.isbn = isbn;
18    }
19
20    // Constructor sin ISBN (sobrecarga 2)
21    public Libro(String titulo, Autor autor, int año) throws IllegalArgumentException {
22        if (titulo == null || titulo.trim().isEmpty()) {
23            throw new IllegalArgumentException("Error: No se puede registrar un libro sin título");
24        }
25        this.titulo = titulo;
26        this.autor = autor;
27        this.año = año;
28        this.isbn = "Sin ISBN";
29    }
30
31    // Métodos getter y setter
32    public String getTitulo() {
33        return titulo;
34    }
35
36    public void setTitulo(String titulo) {
37        if (titulo == null || titulo.trim().isEmpty()) {
38            throw new IllegalArgumentException("Error: El título no puede estar vacío");
39        }
40        this.titulo = titulo;
41    }
42
43    public Autor getAutor() {
44        return autor;
45    }
46
47    public void setAutor(Autor autor) {
48        this.autor = autor;
49    }
50
51    public int getAño() {
52        return año;
53    }
54
55    public void setAño(int año) {
56        this.año = año;
57    }
58

```

```

59 public String getIsbn() {
60     return isbn;
61 }
62
63 public void setIsbn(String isbn) {
64     this.isbn = isbn;
65 }
66
67 // Método para mostrar información del libro
68 public void mostrarInformacion() {
69     System.out.println("=== INFORMACIÓN DEL LIBRO ===");
70     System.out.println("Título: " + titulo);
71     System.out.println("Autor: " + autor.toString());
72     System.out.println("Año: " + año);
73     System.out.println("ISBN: " + isbn);
74     System.out.println("=====\n");
75 }
76
77 @Override
78 public String toString() {
79     return "\"" + titulo + "\" por " + autor.getNombreCompleto() + " (" + año + ")";
80 }
81 }

```

BibliotecaVirtual.java

```

package caso1_sistemaBiblioteca;

import java.util.ArrayList;
import java.util.List;

public class BibliotecaVirtual {

    private List<Libro> libros;

    public BibliotecaVirtual() {
        this.libros = new ArrayList<>();
    }

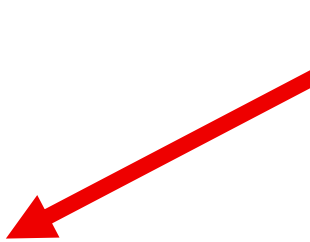
    // Método para registrar libro con ISBN
    public void registrarLibro(String titulo, Autor autor, int año, String isbn) {

        try {
            Libro libro = new Libro(titulo, autor, año, isbn);
            libros.add(libro);

            System.out.println("✓ Libro registrado exitosamente con ISBN: " + isbn);
        } catch (IllegalArgumentException e) {
            System.err.println(e.getMessage());
        }
    }

    // Método para registrar libro sin ISBN (sobrecarga)

```



```

public void registrarLibro(String titulo, Autor autor, int año) {
    try {
        Libro libro = new Libro(titulo, autor, año);
        libros.add(libro);

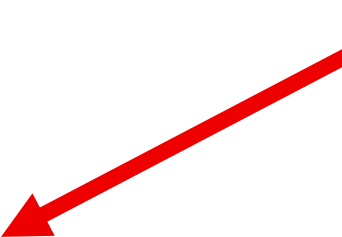
        System.out.println("✓ Libro registrado exitosamente sin ISBN");
    } catch (IllegalArgumentException e) {
        System.err.println(e.getMessage());
    }
}

// Método para mostrar todos los libros
public void mostrarTodosLosLibros() {
    if (libros.isEmpty()) {
        System.out.println("No hay libros registrados en la biblioteca.");
        return;
    }

    System.out.println("\n=== CATÁLOGO DE LIBROS ===");
    for (int i = 0; i < libros.size(); i++) {
        System.out.println((i + 1) + ". " + libros.get(i).toString());
    }
    System.out.println("=====\n");
}

public List<Libro> getLibros() {
    return new ArrayList<>(libros); // Retorna una copia para proteger la colección
}
}

```



Clase BibliotecaVirtual.java (EVIDENCIA)

```
1 package casol_sistemaBiblioteca;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class BibliotecaVirtual {
7     private List<Libro> libros;
8
9     public BibliotecaVirtual() {
10         this.libros = new ArrayList<>();
11     }
12
13     // Método para registrar libro con ISBN
14     public void registrarLibro(String titulo, Autor autor, int año, String isbn) {
15         try {
16             Libro libro = new Libro(titulo, autor, año, isbn);
17             libros.add(libro);
18             System.out.println("✓ Libro registrado exitosamente con ISBN: " + isbn);
19         } catch (IllegalArgumentException e) {
20             System.err.println(e.getMessage());
21         }
22     }
23
24     // Método para registrar libro sin ISBN (sobrecarga)
25     public void registrarLibro(String titulo, Autor autor, int año) {
26         try {
27             Libro libro = new Libro(titulo, autor, año);
28             libros.add(libro);
29             System.out.println("✓ Libro registrado exitosamente sin ISBN");
30         } catch (IllegalArgumentException e) {
31             System.err.println(e.getMessage());
32         }
33     }
34
35     // Método para mostrar todos los libros
36     public void mostrarTodosLosLibros() {
37         if (libros.isEmpty()) {
38             System.out.println("No hay libros registrados en la biblioteca.");
39             return;
40         }
41
42         System.out.println("\n=== CATÁLOGO DE LIBROS ===");
43         for (int i = 0; i < libros.size(); i++) {
44             System.out.println((i + 1) + ". " + libros.get(i).toString());
45         }
46         System.out.println("===== \n");
47     }
48
49     public List<Libro> getLibros() {
50         return new ArrayList<>(libros); // Retorna una copia para proteger la colección
51     }
52 }
53
54
```

Main.java

```
package caso1_sistemaBiblioteca;

public class Main {

    public static void main(String[] args) {

        System.out.println("=== SISTEMA DE BIBLIOTECA VIRTUAL ===\n");

        // Crear instancia de la biblioteca
        BibliotecaVirtual biblioteca = new BibliotecaVirtual();

        // Crear autores
        Autor autor1 = new Autor("Gabriel", "García Márquez", "Colombiano");
        Autor autor2 = new Autor("Isabel", "Allende", "Chilena");
        Autor autor3 = new Autor("Mario", "Vargas Llosa", "Peruano");
        System.out.println("--- Probando registro de libros ---");

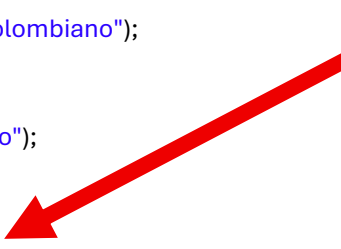
        // Registrar libros con ISBN
        biblioteca.registrarLibro("Cien años de soledad", autor1, 1967, "978-0-06-088328-7");
        biblioteca.registrarLibro("La casa de los espíritus", autor2, 1982, "978-84-204-2676-0");

        // Registrar libro sin ISBN (usando sobrecarga)
        biblioteca.registrarLibro("La ciudad y los perros", autor3, 1963);

        // Intentar registrar libro sin título (debe manejar error)
        System.out.println("\n--- Probando manejo de errores ---");
        biblioteca.registrarLibro("", autor1, 2000);
        biblioteca.registrarLibro(null, autor2, 2000, "123456789");

        // Mostrar todos los libros registrados
        biblioteca.mostrarTodosLosLibros();

        // Mostrar información detallada de cada libro
        System.out.println("--- Información detallada ---");
        for (Libro libro : biblioteca.getLibros()) {
            libro.mostrarInformacion();
        }
        System.out.println("=== FIN DEL PROGRAMA ===");
    }
}
```



Clase Main.java (EVIDENCIA)

```
1 package casol_sistemaBiblioteca;
2
3 public class Main {
4     public static void main(String[] args) {
5         System.out.println("=== SISTEMA DE BIBLIOTECA VIRTUAL ===\n");
6
7         // Crear instancia de la biblioteca
8         BibliotecaVirtual biblioteca = new BibliotecaVirtual();
9
10        // Crear autores
11        Autor autor1 = new Autor("Gabriel", "García Márquez", "Colombiano");
12        Autor autor2 = new Autor("Isabel", "Allende", "Chilena");
13        Autor autor3 = new Autor("Mario", "Vargas Llosa", "Peruano");
14
15        System.out.println("--- Probando registro de libros ---");
16
17        // Registrar libros con ISBN
18        biblioteca.registrarLibro("Cien años de soledad", autor1, 1967, "978-0-06-088328-7");
19        biblioteca.registrarLibro("La casa de los espíritus", autor2, 1982, "978-84-204-2676-0");
20
21        // Registrar libro sin ISBN (usando sobrecarga)
22        biblioteca.registrarLibro("La ciudad y los perros", autor3, 1963);
23
24        // Intentar registrar libro sin título (debe manejar error)
25        System.out.println("\n--- Probando manejo de errores ---");
26        biblioteca.registrarLibro("", autor1, 2000);
27        biblioteca.registrarLibro(null, autor2, 2000, "123456789");
28
29        // Mostrar todos los libros registrados
30        biblioteca.mostrarTodosLosLibros();
31
32        // Mostrar información detallada de cada libro
33        System.out.println("--- Información detallada ---");
34        for (Libro libro : biblioteca.getLibros()) {
35            libro.mostrarInformacion();
36        }
37
38        System.out.println("=== FIN DEL PROGRAMA ===");
39    }
40 }
```

COMPILACIÓN: (EVIDENCIA)

```
terminated> Main [Java Application] C:\Users\R_Jea\Downloads\eclipse-jee-2025-09-R-win32-x86_64\eclipse\pl
=== SISTEMA DE BIBLIOTECA VIRTUAL ===

--- Probando registro de libros ---
Error: No se puede registrar un libro sin título
Error: No se puede registrar un libro sin título
/ Libro registrado exitosamente con ISBN: 978-0-06-088328-7
/ Libro registrado exitosamente con ISBN: 978-84-204-2676-0
/ Libro registrado exitosamente sin ISBN

--- Probando manejo de errores ---

=== CATÁLOGO DE LIBROS ===
1. "Cien años de soledad" por Gabriel García Márquez (1967)
2. "La casa de los espíritus" por Isabel Allende (1982)
3. "La ciudad y los perros" por Mario Vargas Llosa (1963)
=====

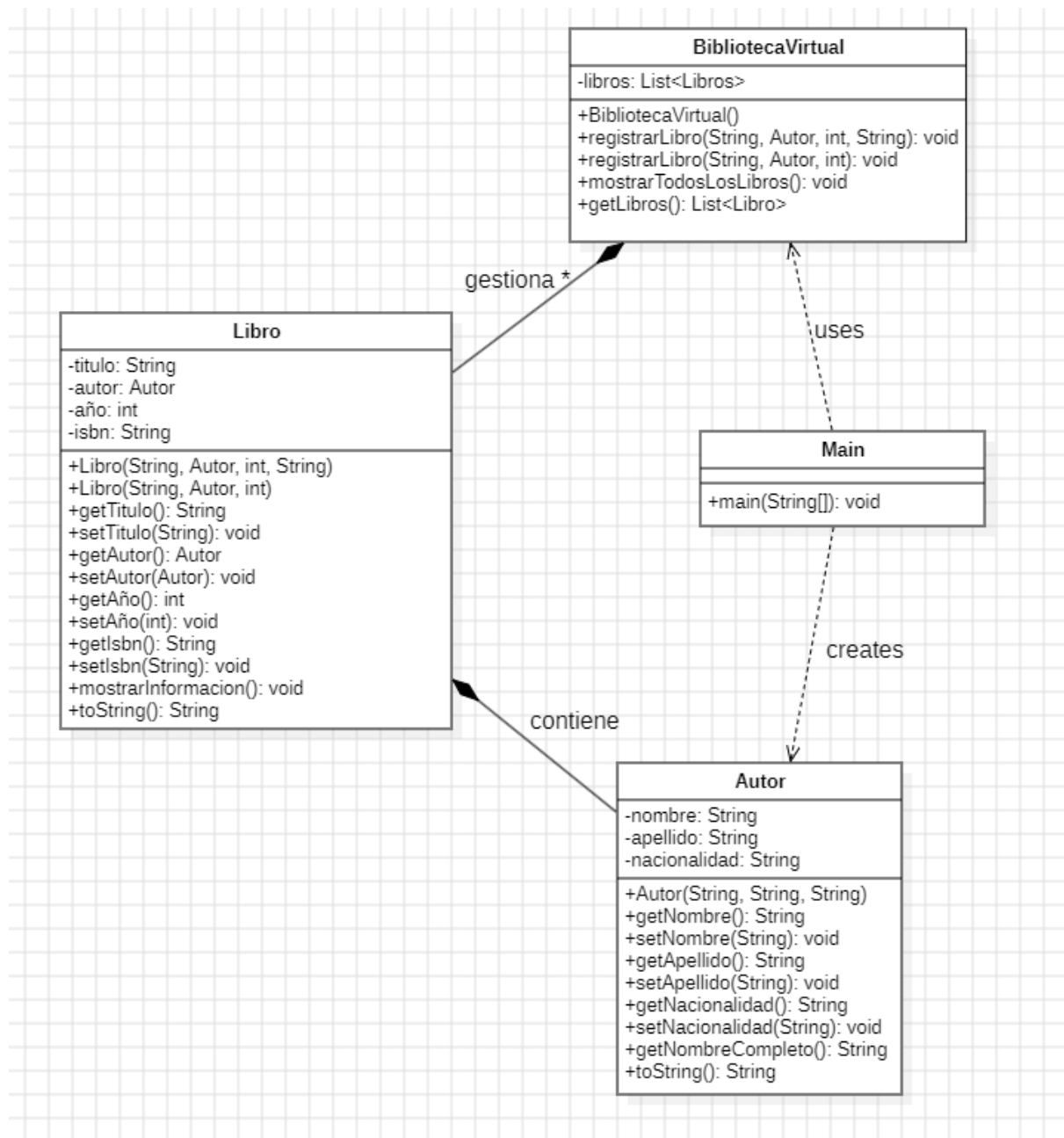
--- Información detallada ---
=== INFORMACIÓN DEL LIBRO ===
Título: Cien años de soledad
Autor: Gabriel García Márquez (Colombiano)
Año: 1967
ISBN: 978-0-06-088328-7
=====

=== INFORMACIÓN DEL LIBRO ===
Título: La casa de los espíritus
Autor: Isabel Allende (Chilena)
Año: 1982
ISBN: 978-84-204-2676-0
=====

=== INFORMACIÓN DEL LIBRO ===
Título: La ciudad y los perros
Autor: Mario Vargas Llosa (Peruano)
Año: 1963
ISBN: Sin ISBN
=====

=== FIN DEL PROGRAMA ===
```


DIAGRAMA UML



Preguntas de análisis:

- **¿Por qué se utilizó sobrecarga en este caso y qué ventajas aporta?**
Se utilizó la sobrecarga para permitir registrar libros con o sin ISBN mediante métodos que tienen el mismo nombre pero diferentes parámetros. Esto aporta flexibilidad al sistema ya que no todos los libros cuentan con ISBN.

Las principales ventajas que aportan son la usabilidad mejorada para el usuario, pues puede elegir la opción más conveniente sin pasar parámetros nulos, y el mantenimiento del código, ya que utiliza un solo nombre de método con comportamientos específicos según los parámetros dados.

- **¿Cómo aplicarías modificadores de acceso para proteger la información?**

Los modificadores de acceso se aplican mediante encapsulación, donde todos los atributos son privados (`private`), para evitar acceso directo desde fuera de la clase. Los métodos públicos (`public`) actúan como interfaces controladas que permiten interactuar con los datos. Los getters y setters no solo proporcionan acceso controlado, sino que también incluyen validaciones para proteger la integridad de los datos, como verificar que un libro no se registre sin título. Además, métodos como `getLibros()` retornan copias de las colecciones para evitar modificaciones no autorizadas de los datos internos.

- **Si tuvieras que escalar este sistema, ¿Qué colecciones usarías para almacenar múltiples libros y por qué?**

Para escalar el sistema, se me viene a la mente, `ArrayList` que es adecuado para el acceso secuencial y cuando el orden de inserción es importante, pero es ineficiente para búsquedas. Luego, para búsquedas rápidas por título, usaría `HashMap<String, Libro>` que ofrece acceso $O(1)$, aunque no permite títulos duplicados, si necesito mantener los datos ordenados automáticamente, `TreeSet<Libro>` sería ideal.

También, para un sistema robusto, recomendaría una combinación de estructuras: `Map<String, List<Libro>>` para agrupar por autor, `Map<String, Libro>` para búsquedas por ISBN, y `Set<Libro>` para el catálogo general, lo que optimizaría diferentes tipos de consultas simultáneamente.

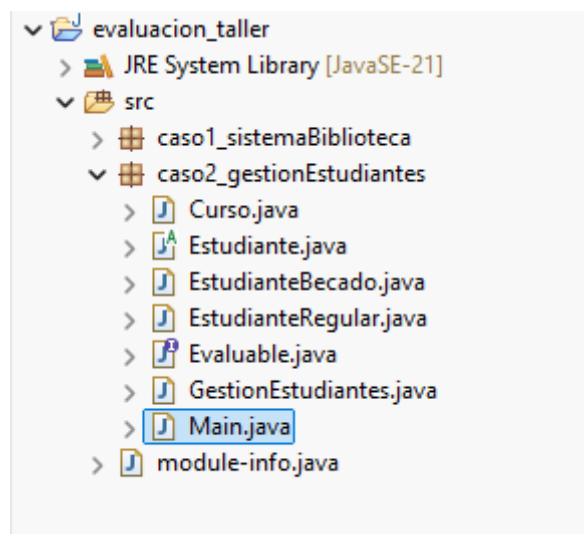
Caso 2 – Gestión de Estudiantes

Un colegio requiere un sistema para gestionar estudiantes y cursos.

Requerimientos:

1. Crear la clase Estudiante y la clase Curso.
2. Establecer una relación entre ellas usando herencia y polimorfismo:
 - EstudianteRegular y EstudianteBecado deben heredar de Estudiante.
 - Cada uno debe implementar un método calcularMensualidad() (sobrescrito).
3. Usar interfaces para definir el contrato Evaluable, que incluya el método evaluar().
4. Representar las relaciones en un diagrama UML.

Distribución del proyecto:



Evaluable.java

```
package caso2_gestionEstudiantes;

public interface Evaluable {

    void evaluar();

    double obtenerPromedio(); }
```

Interfaz Evaluable.java (EVIDENCIA)

```
Evaluable.java × Estudiante.java Estu
1 package caso2_gestionEstudiantes;
2
3 public interface Evaluable {
4     void evaluar();
5     double obtenerPromedio();
6 }
7
```

Estudiante.java

```
package caso2_gestionEstudiantes;

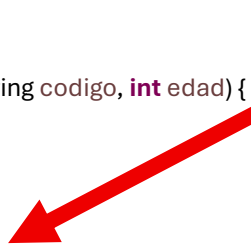
public abstract class Estudiante implements Evaluable {

    protected String nombre;
    protected String apellido;
    protected String codigo;
    protected int edad;
    protected double[] notas;
    protected static int totalEstudiantes = 0; // Atributo estático

    // Constructor
    public Estudiante(String nombre, String apellido, String codigo, int edad) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.codigo = codigo;
        this.edad = edad;
        this.notas = new double[5]; // Array para 5 notas
        totalEstudiantes++; // Incrementar contador estático
    }

    // Método abstracto que debe ser implementado por las subclases
    public abstract double calcularMensualidad();

    // Método estático para obtener el total de estudiantes
    public static int getTotalEstudiantes() {
```



```
    return totalEstudiantes;
}

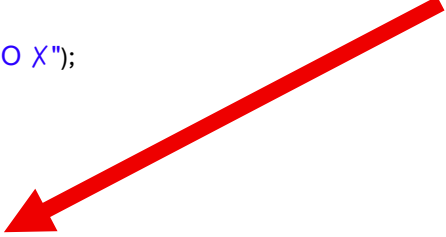
// Implementación de la interfaz Evaluable
@Override
public void evaluar() {
    System.out.println("Evaluando a " + getNombreCompleto());
    System.out.println("Promedio actual: " + obtenerPromedio());

    if (obtenerPromedio() >= 14.0) {
        System.out.println("Estado: APROBADO ✓");
    } else if (obtenerPromedio() >= 10.5) {
        System.out.println("Estado: EN OBSERVACIÓN △");
    } else {
        System.out.println("Estado: DESAPROBADO X");
    }
}

@Override
public double obtenerPromedio() {
    double suma = 0;
    int notasValidas = 0;

    for (double nota : notas) {
        if (nota > 0) { // Solo contar notas asignadas
            suma += nota;
            notasValidas++;
        }
    }

    return notasValidas > 0 ? suma / notasValidas : 0.0;
}
```



// Métodos getter y setter

```
public String getNombre() {  
    return nombre;  
}
```

```
public String getApellido() {  
    return apellido;  
}
```

```
public String getCodigo() {  
    return codigo;  
}
```

```
public int getEdad() {  
    return edad;  
}
```

```
public String getNombreCompleto() {  
    return nombre + " " + apellido;  
}
```

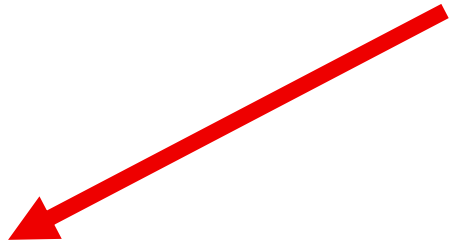
```
public void agregarNota(int posicion, double nota) {  
    if (posicion >= 0 && posicion < notas.length && nota >= 0 && nota <= 20) {  
        notas[posicion] = nota;  
    }  
}
```

```
public double[] getNotas() {  
    return notas.clone(); // Retorna una copia para proteger el array  
}
```

// Método toString común

@Override

```
public String toString() {
```



```
        return getNombreCompleto() + "(Código: " + codigo + ")";
    }
}
```

Clase Abstracta Estudiante.java (EVIDENCIA)

```
1 package caso2_gestionEstudiantes;
2
3 public abstract class Estudiante implements Evaluable {
4     protected String nombre;
5     protected String apellido;
6     protected String codigo;
7     protected int edad;
8     protected double[] notas;
9     protected static int totalEstudiantes = 0; // Atributo estático
10
11     // Constructor
12 public Estudiante(String nombre, String apellido, String codigo, int edad) {
13     this.nombre = nombre;
14     this.apellido = apellido;
15     this.codigo = codigo;
16     this.edad = edad;
17     this.notas = new double[5]; // Array para 5 notas
18     totalEstudiantes++; // Incrementar contador estático
19 }
20
```

```

21 // Método abstracto que debe ser implementado por las subclases
22 public abstract double calcularMensualidad();
23
24 // Método estático para obtener el total de estudiantes
25 public static int getTotalEstudiantes() {
26     return totalEstudiantes;
27 }
28
29 // Implementación de la interfaz Evaluable
30 @Override
31 public void evaluar() {
32     System.out.println("Evaluando a " + getNombreCompleto());
33     System.out.println("Promedio actual: " + obtenerPromedio());
34
35     if (obtenerPromedio() >= 14.0) {
36         System.out.println("Estado: APROBADO ✓");
37     } else if (obtenerPromedio() >= 10.5) {
38         System.out.println("Estado: EN OBSERVACIÓN ⚠");
39     } else {
40         System.out.println("Estado: DESAPROBADO ✗");
41     }
42 }
43
44 @Override
45 public double obtenerPromedio() {
46     double suma = 0;
47     int notasValidas = 0;
48
49     for (double nota : notas) {
50         if (nota > 0) { // Solo contar notas asignadas
51             suma += nota;
52             notasValidas++;
53         }
54     }
55
56     return notasValidas > 0 ? suma / notasValidas : 0.0;
57 }
58
59 // Métodos getter y setter
60 public String getNombre() {
61     return nombre;
62 }
63
64 public String getApellido() {
65     return apellido;
66 }
67
68 public String getCodigo() {
69     return codigo;
70 }
71
72 public int getEdad() {
73     return edad;
74 }
75
76 public String getNombreCompleto() {
77     return nombre + " " + apellido;
78 }

```



```

79
80 public void agregarNota(int posicion, double nota) {
81     if (posicion >= 0 && posicion < notas.length && nota >= 0 && nota <= 20) {
82         notas[posicion] = nota;
83     }
84 }
85
86 public double[] getNotas() {
87     return notas.clone(); // Retorna una copia para proteger el array
88 }
89
90 // Método toString común
91 @Override
92 public String toString() {
93     return getNombreCompleto() + " (Código: " + codigo + ")";
94 }
95 }
96

```

Clase EstudianteRegular.java

```

package caso2_gestionEstudiantes;

public class EstudianteRegular extends Estudiante {

    private static final double MENSUALIDAD_BASE = 500.0;

    private double descuentoPorPromedio;

    public EstudianteRegular(String nombre, String apellido, String codigo, int edad) {

        super(nombre, apellido, codigo, edad);

        this.descuentoPorPromedio = 0.0;

    }

    @Override
    public double calcularMensualidad() {

        double promedio = obtenerPromedio();

        // Aplicar descuento por buen rendimiento académico

        if (promedio >= 17.0) {

            descuentoPorPromedio = 0.15; // 15% de descuento

        } else if (promedio >= 15.0) {

            descuentoPorPromedio = 0.10; // 10% de descuento

        } else if (promedio >= 13.0) {

            descuentoPorPromedio = 0.05; // 5% de descuento

        } else {

            descuentoPorPromedio = 0.0; // Sin descuento

        }

    }

}

```



```

    }

    return MENSUALIDAD_BASE * (1 - descuentoPorPromedio);
}

@Override

public void evaluar() {
    System.out.println("=== EVALUACIÓN ESTUDIANTE REGULAR ===");
    super.evaluar();
    System.out.println("Mensualidad: S/. " + String.format("%.2f", calcularMensualidad()));
    if (descuentoPorPromedio > 0) {
        System.out.println("Descuento aplicado: " + (descuentoPorPromedio * 100) + "%");
    }
    System.out.println("=====\n");
}

public double getDescuentoPorPromedio() {
    return descuentoPorPromedio;
}
}

```

Clase EstudianteRegular.java con (EVIDENCIA)

```
1 package caso2_gestionEstudiantes;
2
3 public class EstudianteRegular extends Estudiante {
4     private static final double MENSUALIDAD_BASE = 500.0;
5     private double descuentoPorPromedio;
6
7     public EstudianteRegular(String nombre, String apellido, String codigo, int edad) {
8         super(nombre, apellido, codigo, edad);
9         this.descuentoPorPromedio = 0.0;
10    }
11
12    @Override
13    public double calcularMensualidad() {
14        double promedio = obtenerPromedio();
15
16        // Aplicar descuento por buen rendimiento académico
17        if (promedio >= 17.0) {
18            descuentoPorPromedio = 0.15; // 15% de descuento
19        } else if (promedio >= 15.0) {
20            descuentoPorPromedio = 0.10; // 10% de descuento
21        } else if (promedio >= 13.0) {
22            descuentoPorPromedio = 0.05; // 5% de descuento
23        } else {
24            descuentoPorPromedio = 0.0; // Sin descuento
25        }
26
27        return MENSUALIDAD_BASE * (1 - descuentoPorPromedio);
28    }
29
30    @Override
31    public void evaluar() {
32        System.out.println("=== EVALUACIÓN ESTUDIANTE REGULAR ===");
33        super.evaluar();
34        System.out.println("Mensualidad: S/. " + String.format("%.2f", calcularMensualidad()));
35        if (descuentoPorPromedio > 0) {
36            System.out.println("Descuento aplicado: " + (descuentoPorPromedio * 100) + "%");
37        }
38        System.out.println("=====\n");
39    }
40
41    public double getDescuentoPorPromedio() {
42        return descuentoPorPromedio;
43    }
44 }
45
```

EstudianteBecado.java

```
package caso2_gestionEstudiantes;

public class EstudianteBecado extends Estudiante {

    private static final double MENSUALIDAD_BASE = 500.0;

    private double porcentajeBeca;

    private String tipoBeca;

    public EstudianteBecado(String nombre, String apellido, String codigo, int edad,

        double porcentajeBeca, String tipoBeca) {

        super(nombre, apellido, codigo, edad);

        this.porcentajeBeca = porcentajeBeca;

        this.tipoBeca = tipoBeca;

    }

    @Override

    public double calcularMensualidad() {

        // La beca reduce la mensualidad según el porcentaje

        double descuentoBeca = MENSUALIDAD_BASE * (porcentajeBeca / 100);

        double mensualidadFinal = MENSUALIDAD_BASE - descuentoBeca;

        // Si tiene muy buen rendimiento, aplicar descuento adicional

        double promedio = obtenerPromedio();

        if (promedio >= 16.0) {

            mensualidadFinal *= 0.95; // 5% adicional por excelencia académica

        }

        return Math.max(mensualidadFinal, 0); // Nunca menor que 0

    }

    @Override

    public void evaluar() {

        System.out.println("=== EVALUACIÓN ESTUDIANTE BECADO ===");

        super.evaluar();

    }

}
```

```
System.out.println("Tipo de beca: " + tipoBeca);

System.out.println("Porcentaje de beca: " + porcentajeBeca + "%");

System.out.println("Mensualidad: S/. " + String.format("%.2f", calcularMensualidad()));

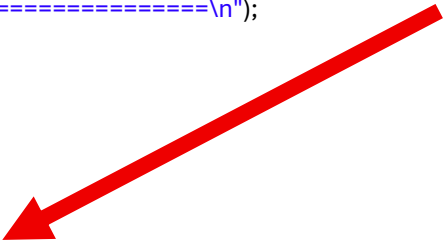
if (obtenerPromedio() < 13.0) {
    System.out.println("⚠ ADVERTENCIA: Promedio bajo puede afectar la beca");
}

System.out.println("=====\n");
}

public double getPorcentajeBeca() {
    return porcentajeBeca;
}

public String getTipoBeca() {
    return tipoBeca;
}

public void setPorcentajeBeca(double porcentajeBeca) {
    if (porcentajeBeca >= 0 && porcentajeBeca <= 100) {
        this.porcentajeBeca = porcentajeBeca;
    }
}
}
```



Clase EstudianteBecado.java (EVIDENCIA)

```
1 package caso2_gestionEstudiantes;
2
3 public class EstudianteBecado extends Estudiante {
4     private static final double MENSUALIDAD_BASE = 500.0;
5     private double porcentajeBeca;
6     private String tipoBeca;
7
8     public EstudianteBecado(String nombre, String apellido, String codigo, int edad,
9                             double porcentajeBeca, String tipoBeca) {
10         super(nombre, apellido, codigo, edad);
11         this.porcentajeBeca = porcentajeBeca;
12         this.tipoBeca = tipoBeca;
13     }
14
15     @Override
16     public double calcularMensualidad() {
17         // La beca reduce la mensualidad según el porcentaje
18         double descuentoBeca = MENSUALIDAD_BASE * (porcentajeBeca / 100);
19         double mensualidadFinal = MENSUALIDAD_BASE - descuentoBeca;
20
21         // Si tiene muy buen rendimiento, aplicar descuento adicional
22         double promedio = obtenerPromedio();
23         if (promedio >= 16.0) {
24             mensualidadFinal *= 0.95; // 5% adicional por excelencia académica
25         }
26
27         return Math.max(mensualidadFinal, 0); // Nunca menor que 0
28     }
29
30     @Override
31     public void evaluar() {
32         System.out.println("=== EVALUACIÓN ESTUDIANTE BECADO ===");
33         super.evaluar();
34         System.out.println("Tipo de beca: " + tipoBeca);
35         System.out.println("Porcentaje de beca: " + porcentajeBeca + "%");
36         System.out.println("Mensualidad: S/. " + String.format("%.2f", calcularMensualidad()));
37
38         if (obtenerPromedio() < 13.0) {
39             System.out.println("⚠ ADVERTENCIA: Promedio bajo puede afectar la beca");
40         }
41         System.out.println("=====\n");
42     }
43
44     public double getPorcentajeBeca() {
45         return porcentajeBeca;
46     }
47
48     public String getTipoBeca() {
49         return tipoBeca;
50     }
51
52     public void setPorcentajeBeca(double porcentajeBeca) {
53         if (porcentajeBeca >= 0 && porcentajeBeca <= 100) {
54             this.porcentajeBeca = porcentajeBeca;
55         }
56     }
57 }
```

Curso.java

```
package caso2_gestionEstudiantes;

public class Curso {

    private String nombre;

    private String codigo;

    private int credits;

    private String profesor;

    private static int totalCursos = 0; // Atributo estático

    public Curso(String nombre, String codigo, int credits, String profesor) {

        this.nombre = nombre;

        this.codigo = codigo;

        this.credits = credits;

        this.profesor = profesor;

        totalCursos++;

    }

    // Método estático

    public static int getTotalCursos() {

        return totalCursos;

    }

    // Método no estático para mostrar información del curso

    public void mostrarInformacion() {

        System.out.println("=== INFORMACIÓN DEL CURSO ===");

        System.out.println("Nombre: " + nombre);

        System.out.println("Código: " + codigo);

        System.out.println("Créditos: " + credits);

        System.out.println("Profesor: " + profesor);

        System.out.println("=====\\n");

    }

    // Getters y Setters
```



```
public String getNombre() {  
    return nombre;  
}  
  
public String getCodigo() {  
    return codigo;  
}  
  
public int getCreditos() {  
    return creditos;  
}  
  
public String getProfesor() {  
    return profesor;  
}  
  
@Override  
public String toString() {  
    return nombre + " (" + codigo + ") - " + creditos + " créditos";  
}  
}
```


Clase Curso.java (EVIDENCIA)

```
1 package caso2_gestionEstudiantes;
2
3 public class Curso {
4     private String nombre;
5     private String codigo;
6     private int credits;
7     private String profesor;
8     private static int totalCursos = 0; // Atributo estático
9
10    public Curso(String nombre, String codigo, int credits, String profesor) {
11        this.nombre = nombre;
12        this.codigo = codigo;
13        this.credits = credits;
14        this.profesor = profesor;
15        totalCursos++;
16    }
17
18    // Método estático
19    public static int getTotalCursos() {
20        return totalCursos;
21    }
22
23    // Método no estático para mostrar información del curso
24    public void mostrarInformacion() {
25        System.out.println("=== INFORMACIÓN DEL CURSO ===");
26        System.out.println("Nombre: " + nombre);
27        System.out.println("Código: " + codigo);
28        System.out.println("Créditos: " + credits);
29        System.out.println("Profesor: " + profesor);
30        System.out.println("=====\n");
31    }
32
33    // Getters y Setters
34    public String getNombre() {
35        return nombre;
36    }
37
38    public String getCodigo() {
39        return codigo;
40    }
41
42    public int getCredits() {
43        return credits;
44    }
45
46    public String getProfesor() {
47        return profesor;
48    }
49
50    @Override
51    public String toString() {
52        return nombre + " (" + codigo + ") - " + credits + " créditos";
53    }
54 }
55
```

GestionEstudiantes.java

```
package caso2_gestionEstudiantes;

import java.util.ArrayList;
import java.util.List;

public class GestionEstudiantes {

    private List<Estudiante> estudiantes;
    private List<Curso> cursos;

    public GestionEstudiantes() {
        this.estudiantes = new ArrayList<>();
        this.cursos = new ArrayList<>();
    }

    // Método que demuestra polimorfismo
    public void procesarEstudiantes() {
        System.out.println("=== PROCESAMIENTO POLIMÓRFICO ===\n");

        for (Estudiante estudiante : estudiantes) {
            // Polimorfismo: el método evaluar() se ejecuta según el tipo real del objeto
            estudiante.evaluar();

            // Polimorfismo: calcularMensualidad() se ejecuta según la implementación de cada
            // subclase
            System.out.println("Cálculo de mensualidad para " + estudiante.getNombreCompleto() +
                ": $/. " + String.format("%.2f", estudiante.calcularMensualidad()) + "\n");
        }
    }

    public void agregarEstudiante(Estudiante estudiante) {
        estudiantes.add(estudiante);
    }

    public void agregarCurso(Curso curso) {
        cursos.add(curso);
    }
}
```

```
}

public void mostrarEstadisticas() {
    System.out.println("=== ESTADÍSTICAS GENERALES ===");
    System.out.println("Total de estudiantes registrados: " + Estudiante.getTotalEstudiantes());
    System.out.println("Total de cursos disponibles: " + Curso.getTotalCursos());
    System.out.println("=====\n");
}

public List<Estudiante> getEstudiantes() {
    return new ArrayList<>(estudiantes);
}

public List<Curso> getCursos() {
    return new ArrayList<>(cursos);
}
}
```

Clase GestiónEstudiantes.java (EVIDENCIA)

```
1 package caso2_gestionEstudiantes;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class GestionEstudiantes {
7     private List<Estudiante> estudiantes;
8     private List<Curso> cursos;
9
10    public GestionEstudiantes() {
11        this.estudiantes = new ArrayList<>();
12        this.cursos = new ArrayList<>();
13    }
14
15    // Método que demuestra polimorfismo
16    public void procesarEstudiantes() {
17        System.out.println("=== PROCESAMIENTO POLIMÓRFICO ===\n");
18
19        for (Estudiante estudiante : estudiantes) {
20            // Polimorfismo: el método evaluar() se ejecuta según el tipo real del objeto
21            estudiante.evaluar();
22
23            // Polimorfismo: calcularMensualidad() se ejecuta según la implementación de cada subclase
24            System.out.println("Cálculo de mensualidad para " + estudiante.getNombreCompleto() +
25                ": S/. " + String.format("%.2f", estudiante.calcularMensualidad()) + "\n");
26        }
27    }
28
29    public void agregarEstudiante(Estudiante estudiante) {
30        estudiantes.add(estudiante);
31    }
32
33    public void agregarCurso(Curso curso) {
34        cursos.add(curso);
35    }
36
37    public void mostrarEstadisticas() {
38        System.out.println("=== ESTADÍSTICAS GENERALES ===");
39        System.out.println("Total de estudiantes registrados: " + Estudiante.getTotalEstudiantes());
40        System.out.println("Total de cursos disponibles: " + Curso.getTotalCursos());
41        System.out.println("===== \n");
42    }
43
44    public List<Estudiante> getEstudiantes() {
45        return new ArrayList<>(estudiantes);
46    }
47
48    public List<Curso> getCursos() {
49        return new ArrayList<>(cursos);
50    }
51 }
52
```

Main.java

```
package caso2_gestionEstudiantes;

public class Main {

    public static void main(String[] args) {

        System.out.println("=== SISTEMA DE GESTIÓN DE ESTUDIANTES ===\n");

        // Crear instancia del sistema

        GestionEstudiantes sistema = new GestionEstudiantes();
    }
}
```

// Crear cursos

```
Curso curso1 = new Curso("Programación Orientada a Objetos", "POO001", 4, "Prof. García");
```

```
Curso curso2 = new Curso("Base de Datos", "BD002", 3, "Prof. López");
```

```
Curso curso3 = new Curso("Algoritmos", "ALG003", 4, "Prof. Martínez");
```

```
sistema.agregarCurso(curso1);
```

```
sistema.agregarCurso(curso2);
```

```
sistema.agregarCurso(curso3);
```

// Crear estudiantes regulares

```
EstudianteRegular estudiante1 = new EstudianteRegular("Ana", "Rodríguez", "E001", 20);
```

```
EstudianteRegular estudiante2 = new EstudianteRegular("Carlos", "Mendoza", "E002", 19);
```

// Crear estudiantes becados

```
EstudianteBecado estudiante3 = new EstudianteBecado("María", "Torres", "E003", 21, 50, "Beca de Excelencia");
```

```
EstudianteBecado estudiante4 = new EstudianteBecado("Luis", "Vargas", "E004", 20, 75, "Beca Socioeconómica");
```

// Agregar estudiantes al sistema

```
sistema.agregarEstudiante(estudiante1);
```

```
sistema.agregarEstudiante(estudiante2);
```

```
sistema.agregarEstudiante(estudiante3);
```

```
sistema.agregarEstudiante(estudiante4);
```

// Asignar notas a los estudiantes

```
System.out.println("--- Asignando notas ---");
```

```
estudiante1.agregarNota(0, 16.5);
```

```
estudiante1.agregarNota(1, 17.0);
```

```
estudiante1.agregarNota(2, 15.5);
```

```
estudiante2.agregarNota(0, 12.0);
```

```
estudiante2.agregarNota(1, 13.5);
```

```
estudiante2.agregarNota(2, 11.8);
```

```
estudiante3.agregarNota(0, 18.0);
estudiante3.agregarNota(1, 17.5);
estudiante3.agregarNota(2, 19.0);

estudiante4.agregarNota(0, 14.5);
estudiante4.agregarNota(1, 15.0);
estudiante4.agregarNota(2, 13.8);

// Mostrar información de cursos
System.out.println("--- Información de Cursos ---");
for (Curso curso : sistema.getCursos()) {
    curso.mostrarInformacion();
}

// Demostrar polimorfismo y uso de interfaces
sistema.procesarEstudiantes();

// Mostrar estadísticas usando métodos estáticos
sistema.mostrarEstadisticas();

// Demostrar diferencia entre métodos estáticos y no estáticos
System.out.println("--- Ejemplo: Métodos Estáticos vs No Estáticos ---");
System.out.println("Método estático - Total estudiantes: " +
    Estudiante.getTotalEstudiantes());
System.out.println("Método estático - Total cursos: " + Curso.getTotalCursos());
System.out.println("Método no estático - Promedio de " +
    estudiante1.getNombreCompleto() +
        ": " + estudiante1.obtenerPromedio());

System.out.println("\n=== FIN DEL PROGRAMA ===");
}
}
```

Clase Main.java (EVIDENCIA)

```
1 package caso2_gestionEstudiantes;
2
3 public class Main {
4     public static void main(String[] args) {
5         System.out.println("=== SISTEMA DE GESTIÓN DE ESTUDIANTES ===\n");
6
7         // Crear instancia del sistema
8         GestionEstudiantes sistema = new GestionEstudiantes();
9
10        // Crear cursos
11        Curso curso1 = new Curso("Programación Orientada a Objetos", "POO001", 4, "Prof. García");
12        Curso curso2 = new Curso("Base de Datos", "BD002", 3, "Prof. López");
13        Curso curso3 = new Curso("Algoritmos", "ALG003", 4, "Prof. Martínez");
14
15        sistema.agregarCurso(curso1);
16        sistema.agregarCurso(curso2);
17        sistema.agregarCurso(curso3);
18
19        // Crear estudiantes regulares
20        EstudianteRegular estudiante1 = new EstudianteRegular("Ana", "Rodríguez", "E001", 20);
21        EstudianteRegular estudiante2 = new EstudianteRegular("Carlos", "Mendoza", "E002", 19);
22
23        // Crear estudiantes becados
24        EstudianteBecado estudiante3 = new EstudianteBecado("María", "Torres", "E003", 21, 50, "Beca de Excelencia");
25        EstudianteBecado estudiante4 = new EstudianteBecado("Luis", "Vargas", "E004", 20, 75, "Beca Socioeconómica");
26
27        // Agregar estudiantes al sistema
28        sistema.agregarEstudiante(estudiante1);
29        sistema.agregarEstudiante(estudiante2);
30        sistema.agregarEstudiante(estudiante3);
31        sistema.agregarEstudiante(estudiante4);
32
33        // Asignar notas a los estudiantes
34        System.out.println("--- Asignando notas ---");
35        estudiante1.agregarNota(0, 16.5);
36        estudiante1.agregarNota(1, 17.0);
37        estudiante1.agregarNota(2, 15.5);
38
39        estudiante2.agregarNota(0, 12.0);
40        estudiante2.agregarNota(1, 13.5);
41        estudiante2.agregarNota(2, 11.8);
42
43        estudiante3.agregarNota(0, 18.0);
44        estudiante3.agregarNota(1, 17.5);
45        estudiante3.agregarNota(2, 19.0);
46
47        estudiante4.agregarNota(0, 14.5);
48        estudiante4.agregarNota(1, 15.0);
49        estudiante4.agregarNota(2, 13.8);
50
51        // Mostrar información de cursos
52        System.out.println("--- Información de Cursos ---");
53        for (Curso curso : sistema.getCursos()) {
54            curso.mostrarInformacion();
55        }
56
57        // Demostrar polimorfismo y uso de interfaces
58        sistema.procesarEstudiantes();
59
60        // Mostrar estadísticas usando métodos estáticos
61        sistema.mostrarEstadisticas();
62
63        // Demostrar diferencia entre métodos estáticos y no estáticos
64        System.out.println("--- Ejemplo: Métodos Estáticos vs No Estáticos ---");
65        System.out.println("Método estático - Total estudiantes: " + Estudiante.getTotalEstudiantes());
66        System.out.println("Método estático - Total cursos: " + Curso.getTotalCursos());
67        System.out.println("Método no estático - Promedio de " + estudiante1.getNombreCompleto() +
68            ": " + estudiante1.obtenerPromedio());
69
70        System.out.println("\n=== FIN DEL PROGRAMA ===");
71    }
72 }
73
```

COMPILACIÓN: (EVIDENCIA)

=== SISTEMA DE GESTIÓN DE ESTUDIANTES ===

--- Asignando notas ---
--- Información de Cursos ---
=== INFORMACIÓN DEL CURSO ===
Nombre: Programación Orientada a Objetos
Código: POO001
Créditos: 4
Profesor: Prof. García
=====

=== INFORMACIÓN DEL CURSO ===
Nombre: Base de Datos
Código: BD002
Créditos: 3
Profesor: Prof. López
=====

=== INFORMACIÓN DEL CURSO ===
Nombre: Algoritmos
Código: ALG003
Créditos: 4
Profesor: Prof. Martínez
=====

=== PROCESAMIENTO POLIMÓRFICO ===

=== EVALUACIÓN ESTUDIANTE REGULAR ===
Evaluando a Ana Rodríguez
Promedio actual: 16.33333333333332
Estado: APROBADO ✓
Mensualidad: S/. 450.00
Descuento aplicado: 10.0%
=====

Cálculo de mensualidad para Ana Rodríguez: S/. 450.00

=== EVALUACIÓN ESTUDIANTE REGULAR ===
Evaluando a Carlos Mendoza
Promedio actual: 12.43333333333332
Estado: EN OBSERVACIÓN ⚠
Mensualidad: S/. 500.00
=====

Cálculo de mensualidad para Carlos Mendoza: S/. 500.00

=== EVALUACIÓN ESTUDIANTE BECADO ===
Evaluando a María Torres
Promedio actual: 18.166666666666668
Estado: APROBADO ✓
Tipo de beca: Beca de Excelencia
Porcentaje de beca: 50.0%
Mensualidad: S/. 237.50
=====

Cálculo de mensualidad para María Torres: S/. 237.50

=== EVALUACIÓN ESTUDIANTE BECADO ===
Evaluando a Luis Vargas
Promedio actual: 14.43333333333332
Estado: APROBADO ✓
Tipo de beca: Beca Socioeconómica
Porcentaje de beca: 75.0%
Mensualidad: S/. 125.00
=====

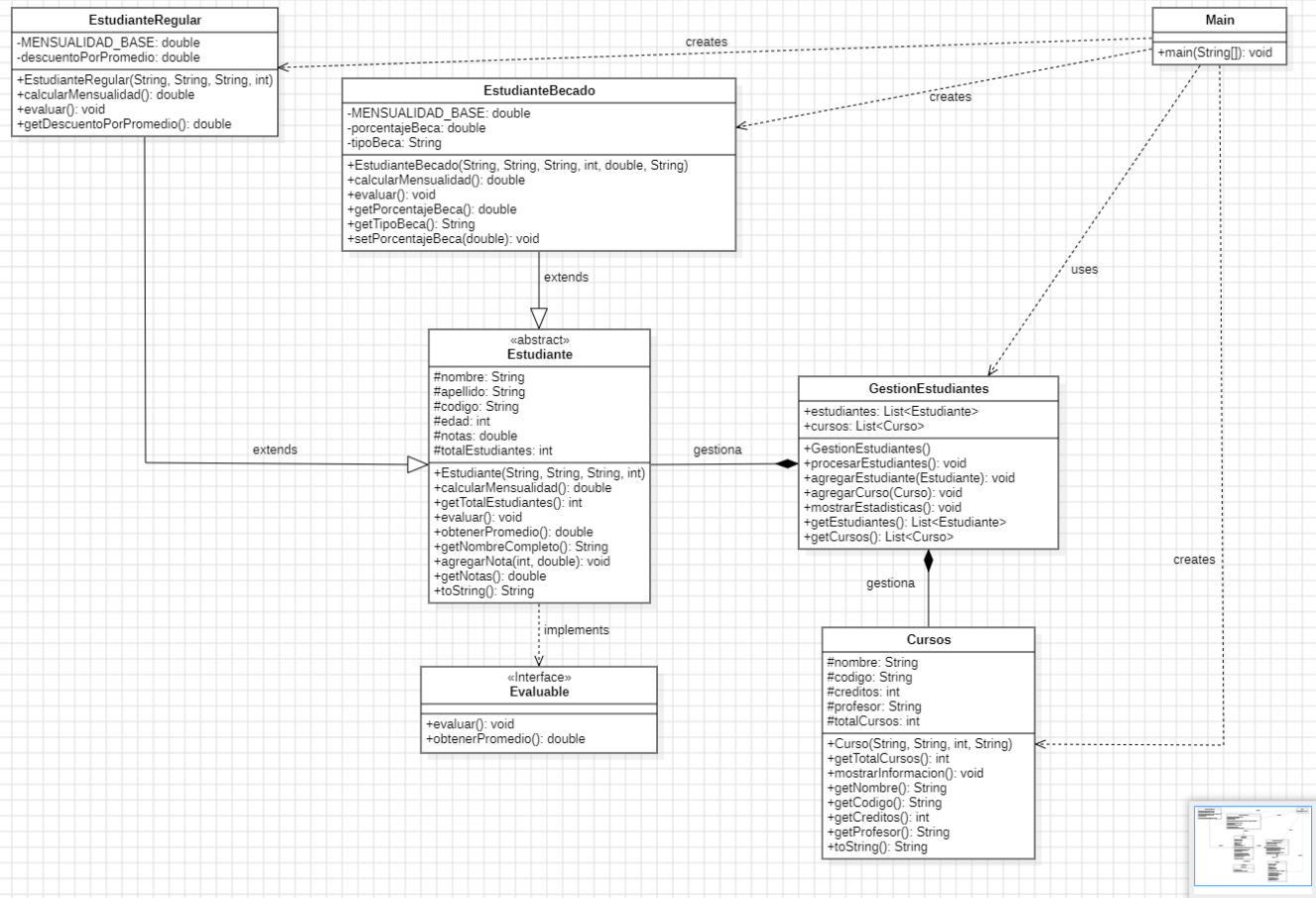
Cálculo de mensualidad para Luis Vargas: S/. 125.00

=== ESTADÍSTICAS GENERALES ===
Total de estudiantes registrados: 4
Total de cursos disponibles: 3
=====

--- Ejemplo: Métodos Estáticos vs No Estáticos ---
Método estático - Total estudiantes: 4
Método estático - Total cursos: 3
Método no estático - Promedio de Ana Rodríguez: 16.33333333333332

=== FIN DEL PROGRAMA ===

DIAGRAMA UML



Preguntas de análisis:

- ¿Por qué se utilizó sobrecarga en este caso y qué ventajas aporta?

Se utilizó sobrecarga porque la herencia permite crear una estructura jerárquica donde `EstudianteRegular` y `EstudianteBecado` comparten características comunes de la clase padre `Estudiante`, evitando duplicación de código. Luego en el polimorfismo es especialmente útil porque permite que diferentes tipos de estudiantes implementen el cálculo de mensualidad de manera específica: los regulares aplican descuentos por rendimiento académico, mientras que los becados calculan según su porcentaje de beca. Esto hace que el sistema sea extensible y mantenible, ya que se puede agregar nuevos tipos de estudiantes sin modificar el código existente, y el método `procesarEstudiantes()` puede trabajar con cualquier tipo de estudiante de manera uniforme.

- **¿Qué problemas podrían surgir si no se usarán interfaces en el diseño?**

Sin interfaces, el sistema perdería flexibilidad y acoplamiento débil. Pues la interfaz `Evaluable` garantiza que todas las clases que la implementen tengan los métodos `evaluar()` y `obtenerPromedio()`, creando un contrato que asegura consistencia en el comportamiento, donde sin esto podríamos tener clases con métodos de nombres diferentes para la misma funcionalidad. Además, las interfaces permiten que clases no relacionadas por herencia puedan ser tratadas de manera similar, y facilitan las pruebas unitarias mediante la creación de implementaciones mock. Sin interfaces, el código se vuelve más rígido y menos escalable.

- **¿Qué diferencia hay entre atributos/métodos estáticos y no estáticos en este contexto?**

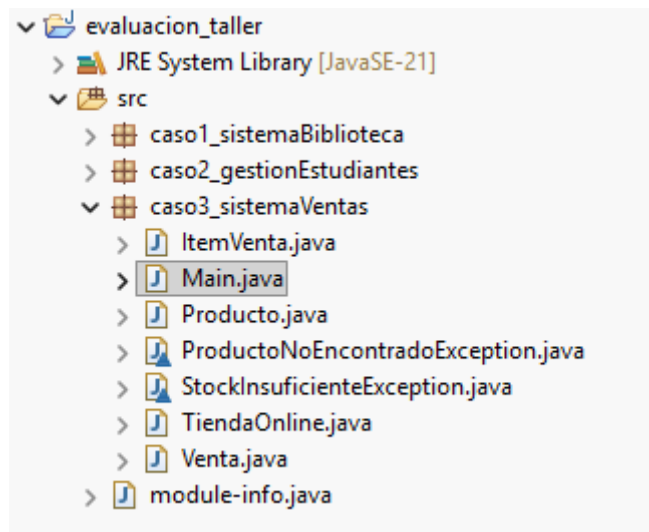
La diferencia entre los atributos estáticos como `totalEstudiantes` y `totalCursos` pertenecen a la clase, no a las instancias individuales, y mantienen información compartida entre todos los objetos. Además, los métodos estáticos como `getTotalEstudiantes()` pueden ser llamados sin crear una instancia de la clase y acceden solo a datos estáticos. Por el contrario, los atributos no estáticos como `nombre`, `notas`, y `porcentajeBeca` son únicos para cada objeto, y los métodos no estáticos como `calcularMensualidad()` y `obtenerPromedio()` operan sobre los datos específicos de cada instancia. En este sistema, los elementos estáticos son útiles para llevar contadores globales y estadísticas generales, mientras que los no estáticos manejan la información particular de cada estudiante.

Caso 3 – Sistema de Ventas en Línea

Una tienda en línea necesita un sistema sencillo para gestionar productos y ventas. Requerimientos:

1. Crear la clase Producto (nombre, precio, stock).
2. Implementar un ArrayList para registrar múltiples productos.
3. Crear un método que realice una compra:
 - Disminuir el stock.
 - Manejar errores si el stock es insuficiente.
4. Incluir una clase Venta que guarde información de cada transacción.
5. Representar las clases y relaciones en un diagrama UML.

Distribución del proyecto:



StockInsuficienteException.java

```
package caso3_sistemaVentas;

class StockInsuficienteException extends Exception {

    private static final long serialVersionUID = 1L;

    public StockInsuficienteException(String mensaje) {
```

```
        super(mensaje);  
    }  
}
```

Clase StockInsuficienteException.java (EVIDENCIA)

```
1 package caso3_sistemaVentas;  
2  
3 class StockInsuficienteException extends Exception {  
4     private static final long serialVersionUID = 1L;  
5  
6     public StockInsuficienteException(String mensaje) {  
7         super(mensaje);  
8     }  
9 }  
10
```

ProductoNoEncontradoException.java

```
package caso3_sistemaVentas;  
  
class ProductoNoEncontradoException extends Exception {  
    private static final long serialVersionUID = 1L;  
  
    public ProductoNoEncontradoException(String mensaje) {  
        super(mensaje);  
    }  
}
```

Clase ProductoNoEncontradoException.java (EVIDENCIA)

```
1 package caso3_sistemaVentas;  
2  
3 class ProductoNoEncontradoException extends Exception {  
4     private static final long serialVersionUID = 1L;  
5  
6     public ProductoNoEncontradoException(String mensaje) {  
7         super(mensaje);  
8     }  
9 }  
10
```

Producto.java

```
package caso3_sistemaVentas;

public class Producto {

    private String nombre;
    private double precio;
    private int stock;
    private String codigo;
    private static int contadorCodigo = 1000;

    // Constructor

    public Producto(String nombre, double precio, int stock) throws IllegalArgumentException {
        if (nombre == null || nombre.trim().isEmpty()) {
            throw new IllegalArgumentException("Error: El nombre del producto no puede estar vacío");
        }
        if (precio <= 0) {
            throw new IllegalArgumentException("Error: El precio debe ser mayor que cero");
        }
        if (stock < 0) {
            throw new IllegalArgumentException("Error: El stock no puede ser negativo");
        }

        this.nombre = nombre;
        this.precio = precio;
        this.stock = stock;
        this.codigo = "PROD" + (++contadorCodigo);
    }

    // Método para reducir stock durante una compra

    public void reducirStock(int cantidad) throws StockInsuficienteException {
        if (cantidad <= 0) {
            throw new IllegalArgumentException("La cantidad debe ser mayor que cero");
        }
    }
}
```

```
    if (cantidad > stock) {  
        throw new StockInsuficienteException(  
            "Stock insuficiente para " + nombre + ". Disponible: " + stock + ", Solicitado: " + cantidad  
        );  
    }  
    stock -= cantidad;  
}  
  
// Método para agregar stock  
public void agregarStock(int cantidad) {  
    if (cantidad > 0) {  
        stock += cantidad;  
    }  
}  
  
// Método para verificar disponibilidad  
public boolean estaDisponible(int cantidad) {  
    return stock >= cantidad;  
}  
  
// Getters y Setters  
public String getNombre() {  
    return nombre;  
}  
  
public void setNombre(String nombre) {  
    if (nombre != null && !nombre.trim().isEmpty()) {  
        this.nombre = nombre;  
    }  
}  
  
public double getPrecio() {  
    return precio;  
}
```

```

public void setPrecio(double precio){
    if (precio > 0) {
        this.precio = precio;
    }
}

public int getStock(){
    return stock;
}

public String getCodigo(){
    return codigo;
}

// Método para mostrar información del producto
public void mostrarInformacion(){
    System.out.println("Código: " + codigo + " | " + nombre +
        " | Precio: S/. " + String.format("%.2f", precio) +
        " | Stock: " + stock + " unidades");
}

@Override
public String toString(){
    return nombre + " (S/. " + String.format("%.2f", precio) + ") - Stock: " + stock;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Producto producto = (Producto) obj;
    return codigo.equals(producto.codigo);
}

```

Clase Producto.java (EVIDENCIA)

```
1 package caso3_sistemaVentas;
2
3 public class Producto {
4     private String nombre;
5     private double precio;
6     private int stock;
7     private String codigo;
8     private static int contadorCodigo = 1000;
9
10    // Constructor
11    public Producto(String nombre, double precio, int stock) throws IllegalArgumentException {
12        if (nombre == null || nombre.trim().isEmpty()) {
13            throw new IllegalArgumentException("Error: El nombre del producto no puede estar vacío");
14        }
15        if (precio <= 0) {
16            throw new IllegalArgumentException("Error: El precio debe ser mayor que cero");
17        }
18        if (stock < 0) {
19            throw new IllegalArgumentException("Error: El stock no puede ser negativo");
20        }
21
22        this.nombre = nombre;
23        this.precio = precio;
24        this.stock = stock;
25        this.codigo = "PROD" + (++contadorCodigo);
26    }
27
28    // Método para reducir stock durante una compra
29    public void reducirStock(int cantidad) throws StockInsuficienteException {
30        if (cantidad <= 0) {
31            throw new IllegalArgumentException("La cantidad debe ser mayor que cero");
32        }
33        if (cantidad > stock) {
34            throw new StockInsuficienteException(
35                "Stock insuficiente para " + nombre + ". Disponible: " + stock + ", Solicitado: " + cantidad
36            );
37        }
38        stock -= cantidad;
39    }
40
41    // Método para agregar stock
42    public void agregarStock(int cantidad) {
43        if (cantidad > 0) {
44            stock += cantidad;
45        }
46    }
47
48    // Método para verificar disponibilidad
49    public boolean estaDisponible(int cantidad) {
50        return stock >= cantidad;
51    }
52
53    // Getters y Setters
54    public String getNombre() {
55        return nombre;
56    }
57}
```



```

58~ public void setNombre(String nombre) {
59~     if (nombre != null && !nombre.trim().isEmpty()) {
60~         this.nombre = nombre;
61~     }
62~ }
63
64~ public double getPrecio() {
65~     return precio;
66~ }
67
68~ public void setPrecio(double precio) {
69~     if (precio > 0) {
70~         this.precio = precio;
71~     }
72~ }
73
74~ public int getStock() {
75~     return stock;
76~ }
77
78~ public String getCodigo() {
79~     return codigo;
80~ }
81
82 // Método para mostrar información del producto
83~ public void mostrarInformacion() {
84~     System.out.println("Código: " + codigo + " | " + nombre +
85~         " | Precio: S/. " + String.format("%.2f", precio) +
86~         " | Stock: " + stock + " unidades");
87~ }
88
89 @Override
90~ public String toString() {
91~     return nombre + " (S/. " + String.format("%.2f", precio) + ") - Stock: " + stock;
92~ }
93
94 @Override
95~ public boolean equals(Object obj) {
96~     if (this == obj) return true;
97~     if (obj == null || getClass() != obj.getClass()) return false;
98~     Producto producto = (Producto) obj;
99~     return codigo.equals(producto.codigo);
100~ }
101 }
102

```

ItemVenta.java

```

package caso3_sistemaVentas;

public class ItemVenta {

    private Producto producto;

    private int cantidad;

    private double precioUnitario;

    private double subtotal;

    public ItemVenta(Producto producto, int cantidad) {

        this.producto = producto;

        this.cantidad = cantidad;

        this.precioUnitario = producto.getPrecio();

        this.subtotal = precioUnitario * cantidad;

    }

}

```

```
// Getters
```

```
public Producto getProducto() {  
    return producto;  
}
```

```
public int getCantidad() {  
    return cantidad;  
}
```

```
public double getPrecioUnitario() {  
    return precioUnitario;  
}
```

```
public double getSubtotal() {  
    return subtotal;  
}
```

```
@Override
```

```
public String toString() {  
    return cantidad + "x" + producto.getNombre() +  
        " @ S/. " + String.format("%.2f", precioUnitario) +  
        " = S/. " + String.format("%.2f", subtotal);  
}  
}
```

Clase ItemVenta.java (EVIDENCIA)

```
1 package caso3_sistemaVentas;
2
3 public class ItemVenta {
4     private Producto producto;
5     private int cantidad;
6     private double precioUnitario;
7     private double subtotal;
8
9     public ItemVenta(Producto producto, int cantidad) {
10         this.producto = producto;
11         this.cantidad = cantidad;
12         this.precioUnitario = producto.getPrecio();
13         this.subtotal = precioUnitario * cantidad;
14     }
15
16     // Getters
17     public Producto getProducto() {
18         return producto;
19     }
20
21     public int getCantidad() {
22         return cantidad;
23     }
24
25     public double getPrecioUnitario() {
26         return precioUnitario;
27     }
28
29     public double getSubtotal() {
30         return subtotal;
31     }
32
33     @Override
34     public String toString() {
35         return cantidad + "x " + producto.getNombre() +
36             " @ S/. " + String.format("%.2f", precioUnitario) +
37             " = S/. " + String.format("%.2f", subtotal);
38     }
39 }
40
41
```

Clase Venta.java

```
package caso3_sistemaVentas;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;
```

```
import java.util.List;

public class Venta {

    private String numeroVenta;

    private LocalDateTime fechaHora;

    private List<ItemVenta> items; // Composición: Una venta está compuesta de items

    private double total;

    private static int contadorVentas = 1;

    public Venta() {

        this.numeroVenta = "VTA" + String.format("%04d", contadorVentas++);

        this.fechaHora = LocalDateTime.now();

        this.items = new ArrayList<>();

        this.total = 0.0;

    }

    // Método para agregar un item a la venta

    public void agregarItem(Producto producto, int cantidad) throws
    StockInsuficienteException {

        if (!producto.estaDisponible(cantidad)) {

            throw new StockInsuficienteException(

                "No hay suficiente stock de " + producto.getNombre() +

                ". Disponible: " + producto.getStock()

            );

        }

        ItemVenta item = new ItemVenta(producto, cantidad);

        items.add(item);

        total += item.getSubtotal();

    }

    // Método para finalizar la venta (procesar stock)

    public void finalizarVenta() throws StockInsuficienteException {

        for (ItemVenta item : items) {

            item.getProducto().reducirStock(item.getCantidad());

        }

    }

}
```

```

    }
}

// Método para cancelar venta (restaurar stock si ya se procesó)
public void cancelarVenta() {
    for (ItemVenta item : items) {
        item.getProducto().agregarStock(item.getCantidad());
    }
    items.clear();
    total = 0.0;
}

// Método para mostrar el detalle de la venta
public void mostrarDetalle() {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

    System.out.println("===== DETALLE DE VENTA =====");
    System.out.println("Número: " + numeroVenta);
    System.out.println("Fecha: " + fechaHora.format(formatter));
    System.out.println("-----");

    if (items.isEmpty()) {
        System.out.println("No hay items en esta venta.");
    } else {
        for (ItemVenta item : items) {
            System.out.println(item.toString());
        }
        System.out.println("-----");
        System.out.println("TOTAL: S/. " + String.format("%.2f", total));
    }
    System.out.println("=====\\n");
}

// Getters

```

```

public String getNumeroVenta() {
    return numeroVenta;
}

public LocalDateTime getFechaHora() {
    return fechaHora;
}

public List<ItemVenta> getItems() {
    return new ArrayList<>(items); // Retorna copia para proteger la colección
}

public double getTotal() {
    return total;
}

public boolean tienelItems() {
    return !items.isEmpty();
}
}

```

Clase Venta.java (EVIDENCIA)

```

1 package caso3_sistemaVentas;
2
3 import java.time.LocalDateTime;
4 import java.time.format.DateTimeFormatter;
5 import java.util.ArrayList;
6 import java.util.List;
7
8 public class Venta {
9     private String numeroVenta;
10    private LocalDateTime fechaHora;
11    private List<ItemVenta> items; // Composición: Una venta está compuesta de items
12    private double total;
13    private static int contadorVentas = 1;
14
15    public Venta() {
16        this.numeroVenta = "VTA" + String.format("%04d", contadorVentas++);
17        this.fechaHora = LocalDateTime.now();
18        this.items = new ArrayList<>();
19        this.total = 0.0;
20    }
21

```

```

22 // Método para agregar un item a la venta
23 public void agregarItem(Producto producto, int cantidad) throws StockInsuficienteException {
24     if (!producto.estaDisponible(cantidad)) {
25         throw new StockInsuficienteException(
26             "No hay suficiente stock de " + producto.getNombre() +
27             ". Disponible: " + producto.getStock()
28         );
29     }
30
31     ItemVenta item = new ItemVenta(producto, cantidad);
32     items.add(item);
33     total += item.getSubtotal();
34 }
35
36 // Método para finalizar la venta (procesar stock)
37 public void finalizarVenta() throws StockInsuficienteException {
38     for (ItemVenta item : items) {
39         item.getProducto().reducirStock(item.getCantidad());
40     }
41 }
42
43 // Método para cancelar venta (restaurar stock si ya se procesó)
44 public void cancelarVenta() {
45     for (ItemVenta item : items) {
46         item.getProducto().agregarStock(item.getCantidad());
47     }
48     items.clear();
49     total = 0.0;
50 }
51
52 // Método para mostrar el detalle de la venta
53 public void mostrarDetalle() {
54     DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");
55
56     System.out.println("===== DETALLE DE VENTA =====");
57     System.out.println("Número: " + numeroVenta);
58     System.out.println("Fecha: " + fechaHora.format(formatter));
59     System.out.println("-----");
60
61     if (items.isEmpty()) {
62         System.out.println("No hay items en esta venta.");
63     } else {
64         for (ItemVenta item : items) {
65             System.out.println(item.toString());
66         }
67         System.out.println("-----");
68         System.out.println("TOTAL: S/. " + String.format("%.2f", total));
69     }
70     System.out.println("=====\\n");
71 }
72
73 // Getters
74 public String getNumeroVenta() {
75     return numeroVenta;
76 }
77
78 public LocalDateTime getFechaHora() {
79     return fechaHora;
80 }
81
82 public List<ItemVenta> getItems() {
83     return new ArrayList<>(items); // Retorna copia para proteger la colección
84 }
85
86 public double getTotal() {
87     return total;
88 }
89
90 public boolean tieneItems() {
91     return !items.isEmpty();
92 }
93 }

```

TiendaOnline.java

```
package caso3_sistemaVentas;

import java.util.ArrayList;
import java.util.List;

public class TiendaOnline {

    private List<Producto> productos; // ArrayList para gestionar múltiples productos
    private List<Venta> historialVentas;
    private String nombreTienda;

    public TiendaOnline(String nombreTienda) {

        this.nombreTienda = nombreTienda;

        this.productos = new ArrayList<>();

        this.historialVentas = new ArrayList<>();

    }

    // Método para agregar productos al inventario

    public void agregarProducto(Producto producto) {

        if (!productos.contains(producto)) {

            productos.add(producto);

            System.out.println("✓ Producto agregado: " + producto.getNombre());

        } else {

            System.out.println("⚠ El producto ya existe en el inventario");

        }

    }

    // Método para buscar producto por código

    public Producto buscarProducto(String codigo) throws ProductoNoEncontradoException {

        for (Producto producto : productos) {

            if (producto.getCodigo().equals(codigo)) {

                return producto;

            }

        }

        throw new ProductoNoEncontradoException("Producto con código " + codigo + " no encontrado");

    }

}
```



```

}

// Método para buscar producto por nombre

public Producto buscarProductoPorNombre(String nombre) throws
ProductoNoEncontradoException {

    for (Producto producto : productos) {

        if (producto.getNombre().equalsIgnoreCase(nombre)) {

            return producto;

        }

    }

    throw new ProductoNoEncontradoException("Producto " + nombre + " no encontrado");

}

// Método principal para realizar una compra

public Venta realizarCompra() {

    return new Venta();

}

// Método para procesar una venta completa

public void procesarVenta(Venta venta) throws StockInsuficienteException {

    if (!venta.tieneItems()) {

        throw new IllegalArgumentException("No se puede procesar una venta sin items");

    }

    try {

        venta.finalizarVenta();

        historialVentas.add(venta);

        System.out.println("✓ Venta procesada exitosamente: " + venta.getNumeroVenta());

    } catch (StockInsuficienteException e) {

        System.err.println("X Error al procesar venta: " + e.getMessage());

        throw e;

    }

}

```

// Método para mostrar inventario completo

```
public void mostrarInventario() {  
    System.out.println("=== INVENTARIO DE " + nombreTienda.toUpperCase() + " ===");  
  
    if (productos.isEmpty()) {  
        System.out.println("No hay productos en el inventario.");  
    } else {  
        System.out.println("Total de productos: " + productos.size());  
        System.out.println("-----");  
        for (Producto producto : productos) {  
            producto.mostrarInformacion();  
        }  
    }  
    System.out.println("=====\n");  
}
```

// Método para mostrar historial de ventas

```
public void mostrarHistorialVentas() {  
    System.out.println("=== HISTORIAL DE VENTAS ===");  
  
    if (historialVentas.isEmpty()) {  
        System.out.println("No se han registrado ventas.");  
    } else {  
        double totalVentas = 0;  
        for (Venta venta : historialVentas) {  
            System.out.println(venta.getNumeroVenta() + " - Total: S/. " +  
                String.format("%.2f", venta.getTotal()));  
            totalVentas += venta.getTotal();  
        }  
        System.out.println("-----");  
        System.out.println("Total vendido: S/. " + String.format("%.2f", totalVentas));  
    }  
    System.out.println("=====\n");  
}
```

// Método para obtener productos con stock bajo

```
public List<Producto> getProductosStockBajo(int limite) {  
    List<Producto> stockBajo = new ArrayList<>();  
    for (Producto producto : productos) {  
        if (producto.getStock() <= limite) {  
            stockBajo.add(producto);  
        }  
    }  
    return stockBajo;  
}
```

// Getters

```
public List<Producto> getProductos() {  
    return new ArrayList<>(productos); // Retorna copia para proteger la colección  
}  
  
public List<Venta> getHistorialVentas() {  
    return new ArrayList<>(historialVentas);  
}  
  
public String getNombreTienda() {  
    return nombreTienda;  
}  
}
```

Clase TiendaOnline.java (EVIDENCIA)

```
1 package caso3_sistemaVentas;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class TiendaOnline {
7     private List<Producto> productos; // ArrayList para gestionar múltiples productos
8     private List<Venta> historialVentas;
9     private String nombreTienda;
10
11     public TiendaOnline(String nombreTienda) {
12         this.nombreTienda = nombreTienda;
13         this.productos = new ArrayList<>();
14         this.historialVentas = new ArrayList<>();
15     }
16
17     // Método para agregar productos al inventario
18     public void agregarProducto(Producto producto) {
19         if (!productos.contains(producto)) {
20             productos.add(producto);
21             System.out.println("✓ Producto agregado: " + producto.getNombre());
22         } else {
23             System.out.println("⚠ El producto ya existe en el inventario");
24         }
25     }
26
27     // Método para buscar producto por código
28     public Producto buscarProducto(String codigo) throws ProductoNoEncontradoException {
29         for (Producto producto : productos) {
30             if (producto.getCodigo().equals(codigo)) {
31                 return producto;
32             }
33         }
34         throw new ProductoNoEncontradoException("Producto con código " + codigo + " no encontrado");
35     }
36
37     // Método para buscar producto por nombre
38     public Producto buscarProductoPorNombre(String nombre) throws ProductoNoEncontradoException {
39         for (Producto producto : productos) {
40             if (producto.getNombre().equalsIgnoreCase(nombre)) {
41                 return producto;
42             }
43         }
44         throw new ProductoNoEncontradoException("Producto " + nombre + " no encontrado");
45     }
46
47     // Método principal para realizar una compra
48     public Venta realizarCompra() {
49         return new Venta();
50     }
51
52     // Método para procesar una venta completa
53     public void procesarVenta(Venta venta) throws StockInsuficienteException {
54         if (!venta.tieneItems()) {
55             throw new IllegalArgumentException("No se puede procesar una venta sin items");
56         }
57
58         try {
59             venta.finalizarVenta();
60         }
61     }
62 }
```

```

60         historialVentas.add(venta);
61         System.out.println("✓ Venta procesada exitosamente: " + venta.getNumeroVenta());
62     } catch (StockInsuficienteException e) {
63         System.err.println("x Error al procesar venta: " + e.getMessage());
64         throw e;
65     }
66 }
67
68 // Método para mostrar inventario completo
69 public void mostrarInventario() {
70     System.out.println("=== INVENTARIO DE " + nombreTienda.toUpperCase() + " ===");
71
72     if (productos.isEmpty()) {
73         System.out.println("No hay productos en el inventario.");
74     } else {
75         System.out.println("Total de productos: " + productos.size());
76         System.out.println("-----");
77         for (Producto producto : productos) {
78             producto.mostrarInformacion();
79         }
80     }
81     System.out.println("=====\\n");
82 }
83
84 // Método para mostrar historial de ventas
85 public void mostrarHistorialVentas() {
86     System.out.println("=== HISTORIAL DE VENTAS ===");
87
88     if (historialVentas.isEmpty()) {
89         System.out.println("No se han registrado ventas.");
90     } else {
91         double totalVentas = 0;
92         for (Venta venta : historialVentas) {
93             System.out.println(venta.getNumeroVenta() + " - Total: S/. " +
94                 String.format("%.2f", venta.getTotal()));
95             totalVentas += venta.getTotal();
96         }
97         System.out.println("-----");
98         System.out.println("Total vendido: S/. " + String.format("%.2f", totalVentas));
99     }
100     System.out.println("=====\\n");
101 }
102
103 // Método para obtener productos con stock bajo
104 public List<Producto> getProductosStockBajo(int limite) {
105     List<Producto> stockBajo = new ArrayList<>();
106     for (Producto producto : productos) {
107         if (producto.getStock() <= limite) {
108             stockBajo.add(producto);
109         }
110     }
111     return stockBajo;
112 }
113
114 // Getters
115 public List<Producto> getProductos() {
116     return new ArrayList<>(productos); // Retorna copia para proteger la colección
117 }
118
119 public List<Venta> getHistorialVentas() {
120     return new ArrayList<>(historialVentas);
121 }
122
123 public String getNombreTienda() {
124     return nombreTienda;
125 }
126 }
127

```

Main.java

```
package caso3_sistemaVentas;

public class Main {

    public static void main(String[] args) {

        System.out.println("=== SISTEMA DE VENTAS EN LÍNEA ===\n");

        // Crear tienda

        TiendaOnline tienda = new TiendaOnline("TechStore Peru");

        try {

            // Crear y agregar productos

            System.out.println("--- Agregando productos al inventario ---");

            Producto laptop = new Producto("Laptop Gaming ASUS", 2500.00, 5);

            Producto mouse = new Producto("Mouse Logitech G502", 150.00, 10);

            Producto teclado = new Producto("Teclado Mecánico RGB", 200.00, 8);

            Producto monitor = new Producto("Monitor 24 pulgadas", 800.00, 3);

            Producto auriculares = new Producto("Auriculares HyperX", 120.00, 0); // Sin stock

            tienda.agregarProducto(laptop);

            tienda.agregarProducto(mouse);

            tienda.agregarProducto(teclado);

            tienda.agregarProducto(monitor);

            tienda.agregarProducto(auriculares);

            // Mostrar inventario inicial

            tienda.mostrarInventario();

            // Realizar primera venta

            System.out.println("--- Realizando Venta #1 ---");

            Venta venta1 = tienda.realizarCompra();

            try {

                venta1.agregarItem(laptop, 1);
```

```
venta1.agregarItem(mouse, 2);

venta1.agregarItem(teclado, 1);


venta1.mostrarDetalle();

tienda.procesarVenta(venta1);


} catch (StockInsuficienteException e) {

    System.err.println("Error en venta: " + e.getMessage());

}


// Realizar segunda venta con problema de stock

System.out.println("--- Realizando Venta #2 (con error de stock) ---");

Venta venta2 = tienda.realizarCompra();

try {

    venta2.agregarItem(monitor, 2);

    venta2.agregarItem(auriculares, 1); // Este producto no tiene stock

    venta2.mostrarDetalle();

    tienda.procesarVenta(venta2);

} catch (StockInsuficienteException e) {

    System.err.println("Error en venta: " + e.getMessage());

    System.out.println("Cancelando venta...");

    venta2.cancelarVenta();

}


// Realizar tercera venta exitosa

System.out.println("--- Realizando Venta #3 ---");

Venta venta3 = tienda.realizarCompra();

try {

    venta3.agregarItem(monitor, 1);

    venta3.agregarItem(mouse, 1);
```

```
venta3.mostrarDetalle();

tienda.procesarVenta(venta3);

} catch (StockInsuficienteException e) {
    System.err.println("Error en venta: " + e.getMessage());
}

// Mostrar inventario actualizado

System.out.println("--- Inventario después de las ventas ---");

tienda.mostrarInventario();

// Mostrar historial de ventas

tienda.mostrarHistorialVentas();

// Mostrar productos con stock bajo

System.out.println("--- Productos con stock bajo (≤3 unidades) ---");

java.util.List<Producto> stockBajo = tienda.getProductosStockBajo(3);

if (stockBajo.isEmpty()) {
    System.out.println("Todos los productos tienen stock suficiente.");
} else {
    for (Producto producto : stockBajo) {
        System.out.println("△ " + producto.toString());
    }
}

// Demostrar búsqueda de productos

System.out.println("\n--- Demostrando búsqueda de productos ---");

try {
    Producto encontrado = tienda.buscarProductoPorNombre("Mouse Logitech G502");

    System.out.println("✓ Producto encontrado: " + encontrado.toString());
} catch (ProductoNoEncontradoException e) {
    System.err.println("X " + e.getMessage());
}
```



```
    } catch (IllegalArgumentException e) {  
        System.err.println("Error al crear producto: " + e.getMessage());  
    }  
  
    System.out.println("\n=== FIN DEL PROGRAMA ===");  
}  
}
```

Clase Main.java (EVIDENCIA)

```
1 package caso3_sistemaVentas;
2
3 public class Main {
4     public static void main(String[] args) {
5         System.out.println("=== SISTEMA DE VENTAS EN LÍNEA ===\n");
6
7         // Crear tienda
8         TiendaOnline tienda = new TiendaOnline("TechStore Peru");
9
10        try {
11            // Crear y agregar productos
12            System.out.println("--- Agregando productos al inventario ---");
13            Producto laptop = new Producto("Laptop Gaming ASUS", 2500.00, 5);
14            Producto mouse = new Producto("Mouse Logitech G502", 150.00, 10);
15            Producto teclado = new Producto("Teclado Mecánico RGB", 200.00, 8);
16            Producto monitor = new Producto("Monitor 24 pulgadas", 800.00, 3);
17            Producto auriculares = new Producto("Auriculares HyperX", 120.00, 0); // Sin stock
18
19            tienda.agregarProducto(laptop);
20            tienda.agregarProducto(mouse);
21            tienda.agregarProducto(teclado);
22            tienda.agregarProducto(monitor);
23            tienda.agregarProducto(auriculares);
24
25            // Mostrar inventario inicial
26            tienda.mostrarInventario();
27
28            // Realizar primera venta
29            System.out.println("--- Realizando Venta #1 ---");
30            Venta venta1 = tienda.realizarCompra();
31
32            try {
33                venta1.agregarItem(laptop, 1);
34                venta1.agregarItem(mouse, 2);
35                venta1.agregarItem(teclado, 1);
36
37                venta1.mostrarDetalle();
38                tienda.procesarVenta(venta1);
39
40            } catch (StockInsuficienteException e) {
41                System.err.println("Error en venta: " + e.getMessage());
42            }
43
44            // Realizar segunda venta con problema de stock
45            System.out.println("--- Realizando Venta #2 (con error de stock) ---");
46            Venta venta2 = tienda.realizarCompra();
47
48            try {
49                venta2.agregarItem(monitor, 2);
50                venta2.agregarItem(auriculares, 1); // Este producto no tiene stock
51                venta2.mostrarDetalle();
52                tienda.procesarVenta(venta2);
53
54            } catch (StockInsuficienteException e) {
55                System.err.println("Error en venta: " + e.getMessage());
56                System.out.println("Cancelando venta...");
57                venta2.cancelarVenta();
58            }
59
60            // Realizar tercera venta exitosa
61            System.out.println("--- Realizando Venta #3 ---");
62            Venta venta3 = tienda.realizarCompra();
63
64            try {
65                venta3.agregarItem(monitor, 1);
66                venta3.agregarItem(mouse, 1);
67
68                venta3.mostrarDetalle();
69                tienda.procesarVenta(venta3);
70
71            } catch (StockInsuficienteException e) {
72                System.err.println("Error en venta: " + e.getMessage());
73            }
74
75            // Mostrar inventario actualizado
76            System.out.println("--- Inventario después de las ventas ---");
```

```

77         tienda.mostrarInventario();
78
79         // Mostrar historial de ventas
80         tienda.mostrarHistorialVentas();
81
82         // Mostrar productos con stock bajo
83         System.out.println("--- Productos con stock bajo (≤3 unidades) ---");
84         java.util.List<Producto> stockBajo = tienda.getProductosStockBajo(3);
85         if (stockBajo.isEmpty()) {
86             System.out.println("Todos los productos tienen stock suficiente.");
87         } else {
88             for (Producto producto : stockBajo) {
89                 System.out.println("△ " + producto.toString());
90             }
91         }
92
93         // Demostrar búsqueda de productos
94         System.out.println("\n--- Demostrando búsqueda de productos ---");
95         try {
96             Producto encontrado = tienda.buscarProductoPorNombre("Mouse Logitech G502");
97             System.out.println("✓ Producto encontrado: " + encontrado.toString());
98         } catch (ProductoNoEncontradoException e) {
99             System.err.println("x " + e.getMessage());
100         }
101
102         } catch (IllegalArgumentException e) {
103             System.err.println("Error al crear producto: " + e.getMessage());
104         }
105     }
106     System.out.println("\n=== FIN DEL PROGRAMA ===");
107 }
108 }

```

COMPILACIÓN EVIDENCIA

```
=== SISTEMA DE VENTAS EN LÍNEA ===

--- Agregando productos al inventario ---
✓ Producto agregado: Laptop Gaming ASUS
✓ Producto agregado: Mouse Logitech G502
✓ Producto agregado: Teclado Mecánico RGB
✓ Producto agregado: Monitor 24 pulgadas
✓ Producto agregado: Auriculares HyperX
=== INVENTARIO DE TECHSTORE PERU ===
Total de productos: 5
-----
Código: PROD1001 | Laptop Gaming ASUS | Precio: S/. 2500.00 | Stock: 5 unidades
Código: PROD1002 | Mouse Logitech G502 | Precio: S/. 150.00 | Stock: 10 unidades
Código: PROD1003 | Teclado Mecánico RGB | Precio: S/. 200.00 | Stock: 8 unidades
Código: PROD1004 | Monitor 24 pulgadas | Precio: S/. 800.00 | Stock: 3 unidades
Código: PROD1005 | Auriculares HyperX | Precio: S/. 120.00 | Stock: 0 unidades
=====

--- Realizando Venta #1 ---
===== DETALLE DE VENTA =====
Número: VTA0001
Fecha: 18/09/2025 18:04:21
-----
1x Laptop Gaming ASUS @ S/. 2500.00 = S/. 2500.00
2x Mouse Logitech G502 @ S/. 150.00 = S/. 300.00
1x Teclado Mecánico RGB @ S/. 200.00 = S/. 200.00
-----
TOTAL: S/. 3000.00
=====

✓ Venta procesada exitosamente: VTA0001
--- Realizando Venta #2 (con error de stock) ---
Error en venta: No hay suficiente stock de Auriculares HyperX. Disponible: 0
Cancelando venta...

Error en venta: No hay suficiente stock de Auriculares HyperX. Disponible: 0
Cancelando venta...
--- Realizando Venta #3 ---
===== DETALLE DE VENTA =====
Número: VTA0003
Fecha: 18/09/2025 18:04:21
-----
1x Monitor 24 pulgadas @ S/. 800.00 = S/. 800.00
1x Mouse Logitech G502 @ S/. 150.00 = S/. 150.00
-----
TOTAL: S/. 950.00
=====

✓ Venta procesada exitosamente: VTA0003
--- Inventario después de las ventas ---
=== INVENTARIO DE TECHSTORE PERU ===
Total de productos: 5
-----
Código: PROD1001 | Laptop Gaming ASUS | Precio: S/. 2500.00 | Stock: 4 unidades
Código: PROD1002 | Mouse Logitech G502 | Precio: S/. 150.00 | Stock: 7 unidades
Código: PROD1003 | Teclado Mecánico RGB | Precio: S/. 200.00 | Stock: 7 unidades
Código: PROD1004 | Monitor 24 pulgadas | Precio: S/. 800.00 | Stock: 4 unidades
Código: PROD1005 | Auriculares HyperX | Precio: S/. 120.00 | Stock: 0 unidades
=====
```

```

=== HISTORIAL DE VENTAS ===
VTA0001 - Total: S/. 3000.00
VTA0003 - Total: S/. 950.00
-----
Total vendido: S/. 3950.00
=====

```

```

--- Productos con stock bajo (≤3 unidades) ---
⚠ Auriculares HyperX (S/. 120.00) - Stock: 0

```

```

--- Demostrando búsqueda de productos ---
✓ Producto encontrado: Mouse Logitech G502 (S/. 150.00) - Stock: 7

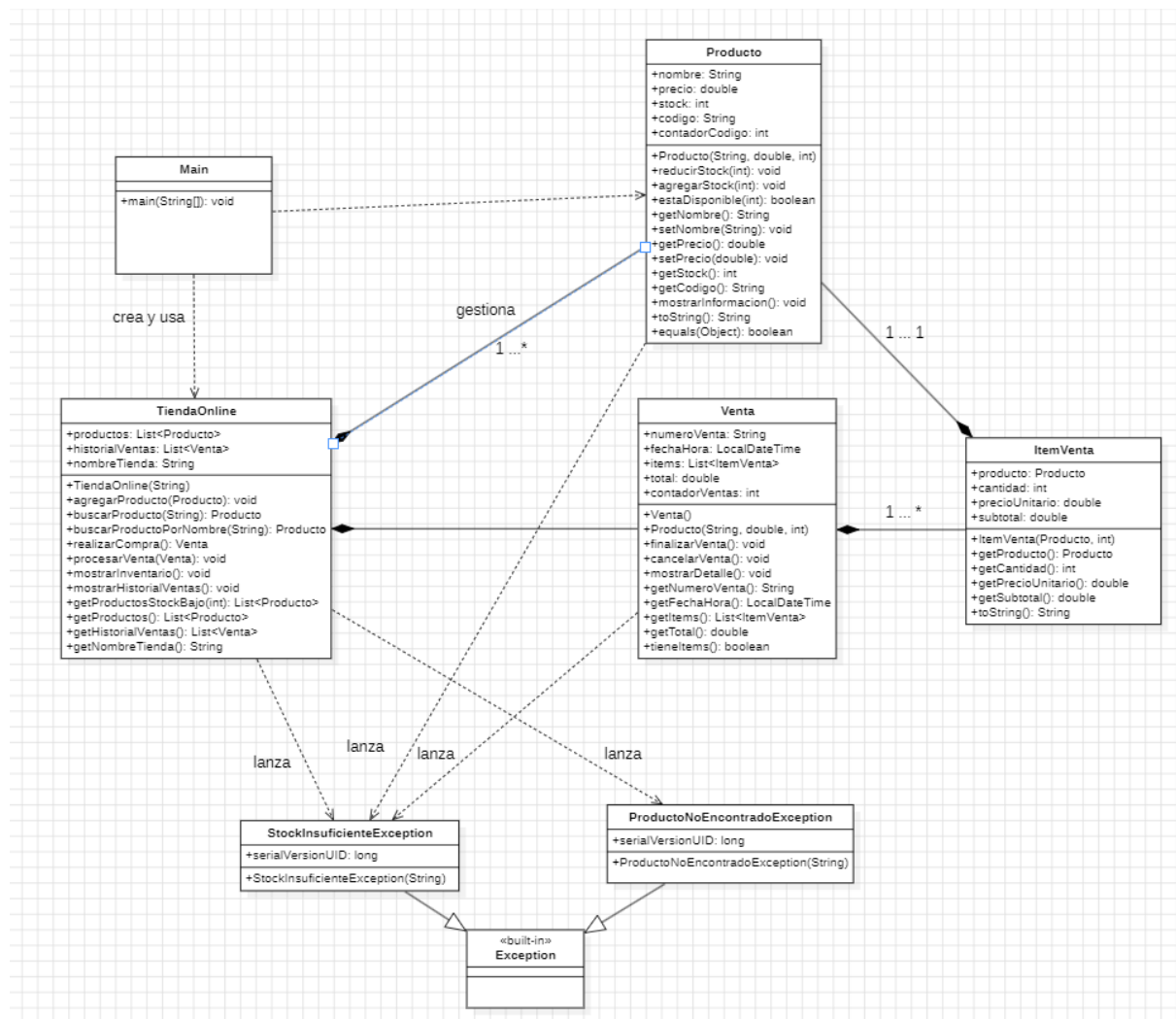
```

```

=== FIN DEL PROGRAMA ===

```

DIAGRAMA UML



Preguntas de análisis:

- ¿Cómo se evidencia el buen uso de las colecciones en este caso?

El buen uso de colecciones se demuestra en varios aspectos del sistema implementado. TiendaOnline utiliza `ArrayList<Producto>` para el inventario y `ArrayList<Venta>` para el historial, permitiendo operaciones eficientes de búsqueda y gestión. La clase `Venta` mantiene una `List<ItemVenta>` que facilita el manejo dinámico de items durante la transacción. El sistema implementa métodos defensivos como `getProductos()` que retorna copias de las colecciones originales, protegiendo la integridad de los datos internos. Además, se incluyen métodos especializados como `getProductosStockBajo()` que filtran y procesan las colecciones según criterios específicos, demostrando un manejo avanzado de las estructuras de datos para diferentes necesidades del negocio.

- ¿Qué excepciones deberías manejar y por qué?

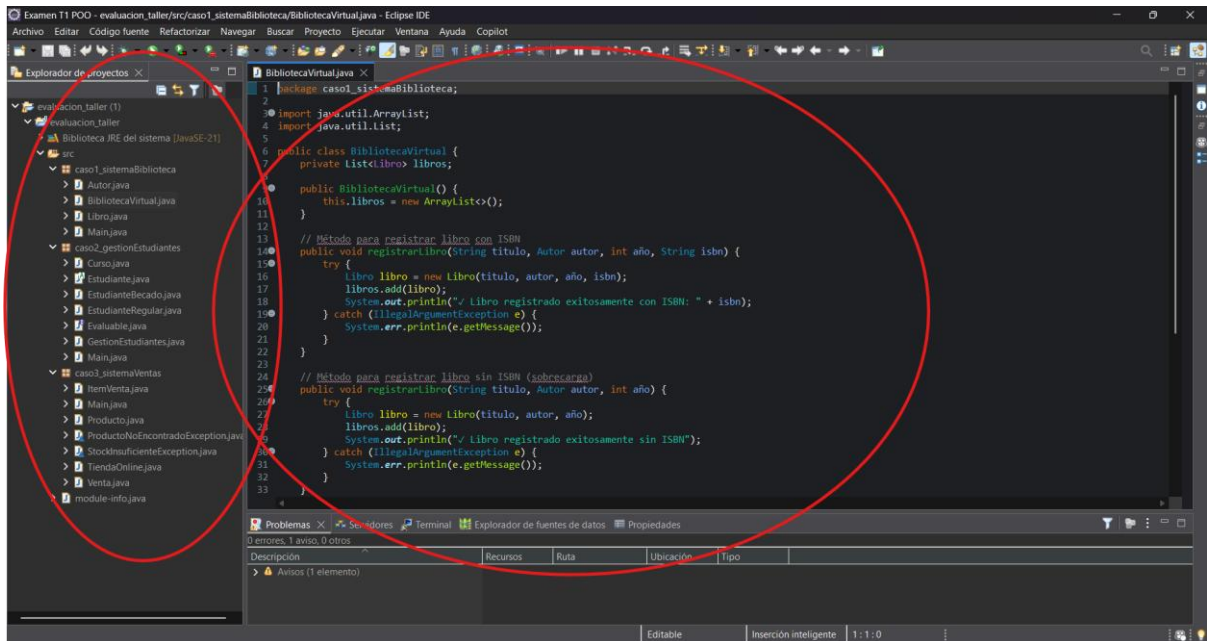
El sistema debe manejar excepciones que protejan la integridad y el correcto funcionamiento. `StockInsuficienteException` es necesaria para evitar ventas mayores al inventario disponible y así garantizar consistencia en el stock. `ProductoNoEncontradoException` permite controlar búsquedas fallidas sin que el sistema se detenga, mostrando un mensaje claro en lugar de generar errores críticos. Estas excepciones aseguran estabilidad, evitan datos inconsistentes y mejoran la experiencia del usuario. Las excepciones son esenciales porque en sistemas comerciales reales, los conflictos de inventario y errores de datos son situaciones frecuentes que deben manejarse sin interrumpir el servicio, proporcionando feedback claro sobre todos los problemas en específicos.

- ¿Cómo se podría aplicar composición en este sistema?

La composición se implementa principalmente en la relación `Venta-ItemVenta`, donde cada venta está "compuesta" por items que no pueden existir independientemente de la venta que los contiene. Cuando una venta se cancela o elimina, automáticamente desaparecen todos sus items asociados, demostrando el ciclo de vida dependiente característico de la composición. Cada `ItemVenta` encapsula información específica de la transacción como precio unitario y subtotal, mientras mantiene una referencia al `Producto` original. Esta estructura permite que el sistema preserve datos históricos de precios al momento de la venta, incluso si el

precio del producto cambia posteriormente. La composición asegura integridad referencial y simplifica la gestión de memoria, ya que los objetos compuestos se eliminan automáticamente con su contenedor.

EVIDENCIA PROGRAMA ECLIPSE:



GIT REPOSITORIO:

<https://github.com/stevenupnn/ExamenT1POO.git>