

**2025**

**FACULTAD DE INGENERIA DE**

**SISTEMAS COMPUTACIONALES**

**TEMA: EXAMEN T1**

**ALUMNO: ID UPN**

**DOCENTE:**

**ING. MARTIN EDUARDO TORRES RODRIGUEZ**

**CURSO:**

**TECNICAS DE PROGRAMACION ORIE. OBJET.**

**TRUJILLO - 2025**

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

**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

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**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)**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**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)**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**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)**  
Texto

El contenido generado por IA puede ser incorrecto.

**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)**

Texto

El contenido generado por IA puede ser incorrecto.

**COMPILACIÓN: (EVIDENCIA)**

**Texto

El contenido generado por IA puede ser incorrecto.**

**DIAGRAMA UML**

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**Preguntas de análisis:**

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

Se utilizo 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() (sobreescrito).

3. Usar interfaces para definir el contrato Evaluable, que incluya el método evaluar(). 4. Representar las relaciones en un diagrama UML.

**Distribución del proyecto:**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Evaluable.java**

|  |
| --- |
| **package** caso2\_gestionEstudiantes;  **public** **interface** Evaluable {  **void** evaluar();  **double** obtenerPromedio(); }} |

Interfaz Evaluable.java (EVIDENCIA)

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**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 ✗");  }  }    @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)**

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**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)**

Texto

El contenido generado por IA puede ser incorrecto.

**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)**

Texto

El contenido generado por IA puede ser incorrecto.

**Curso.java**

|  |
| --- |
| **package** caso2\_gestionEstudiantes;  **public** **class** Curso {  **private** String nombre;  **private** String codigo;  **private** **int** creditos;  **private** String profesor;  **private** **static** **int** *totalCursos* = 0; // Atributo estático    **public** Curso(String nombre, String codigo, **int** creditos, String profesor) {  **this**.nombre = nombre;  **this**.codigo = codigo;  **this**.creditos = creditos;  **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: " + creditos);  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)**

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**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() +  ": S/. " + 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)**

Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**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)**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**COMPILACIÓN:** (EVIDENCIA)

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

**DIAGRAMA UML**

**Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.**

**Preguntas de análisis:**

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

Se utilizo 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:**

Texto

El contenido generado por IA puede ser incorrecto.

**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)

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**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)**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**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)**

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**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)**

Texto

El contenido generado por IA puede ser incorrecto.

**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** tieneItems() {  **return** !items.isEmpty();  }  } |

**Clase Venta.java (EVIDENCIA)**

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**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("✗ 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)**

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**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("✗ " + 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)** Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**COMPILACIÓN EVIDENCIA**

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto, Carta

El contenido generado por IA puede ser incorrecto.

**DIAGRAMA UML**

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.

**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:**

Imagen de la pantalla de un computador

El contenido generado por IA puede ser incorrecto.

**GIT REPOSITORIO:**

**https://github.com/stevenupnn/ExamenT1POO.git**