UNIVERSIDAD POLITECNICA SALESIANA

FACULTAD DE INGENIERÍA

INTEGRANTES
Paúl Crespo

MATERIA
Programación Orientada a Objetos

DOCENTE
Ing. Gustavo Navas Ruilova

TEMA
Examen Final Práctico

2025

Buscaminas POO

Resumen

El proyecto consiste en el desarrollo de una aplicación de consola de un juego de buscaminas, desarrollado en Java con buenas prácticas de programación obtenidas gracias al curso de Programación Orientada a Objetos, el proyecto esta versionado y subido en GitHub en el siguiente enlace:

https://github.com/PaulCrespoC/POO_ExamenPractico

Fases de Desarrollo

Fase 1: Conceptualización y Diseño Base

- **Objetivo**: Establecer la estructura fundamental del juego Buscaminas
- **Implementación**: Clases básicas con lógica esencial
- **Resultados**:
  - Modelo de datos funcional (Tablero, Casilla, Jugador)
  - Lógica de juego completa
  - Algoritmos de colocación de minas y descubrimiento automático

Fase 2: Aplicación de Patrones de Diseño

- **Objetivo**: Implementar el patrón MVC para separar responsabilidades
- **Implementación**:
  - **Modelo**: Lógica de negocio encapsulada
  - **Vista**: Interfaz de consola independiente
  - **Controlador**: Coordinación entre componentes
- **Resultados**:
  - Separación clara de responsabilidades
  - Código más mantenible y testeable
  - Flexibilidad para cambios futuros

Fase 3: Implementación de Principios SOLID

Single Responsibility Principle (SRP)

- **Aplicación**: Cada clase tiene una única responsabilidad bien definida
- **Ejemplos**:
  - Casilla: Solo maneja el estado de una casilla individual
  - VistaConsola: Solo se encarga de la presentación
  - GestorArchivos: Solo maneja la persistencia de datos

Open/Closed Principle (OCP)

- **Aplicación**: Clases abiertas para extensión, cerradas para modificación
- **Implementación**:
  - Estructura modular que permite agregar nuevas funcionalidades
  - Interfaces que facilitan la extensión sin modificar código existente

Liskov Substitution Principle (LSP)

- **Aplicación**: Polimorfismo correcto en la jerarquía de clases
- **Implementación**: Métodos consistentes que mantienen el comportamiento esperado

Interface Segregation Principle (ISP)

- **Aplicación**: Interfaces específicas y cohesivas
- **Implementación**: Separación de responsabilidades en interfaces focalizadas

Dependency Inversion Principle (DIP)

- **Aplicación**: Dependencia de abstracciones, no de implementaciones concretas
- **Implementación**: Inyección de dependencias y uso de interfaces

Fase 4: Manejo Robusto de Excepciones

- **Implementación**:
  - Excepciones personalizadas (CasillaYaDescubiertaException)
  - Manejo de excepciones estándar (InputMismatchException, ArrayIndexOutOfBoundsException)
  - Validación robusta de entrada del usuario
- **Resultados**:
  - Experiencia de usuario mejorada
  - Sistema más estable y confiable
  - Mensajes de error informativos

Fase 5: Persistencia y Gestión de Datos

- **Implementación**:
  - Serialización de objetos para guardado de partidas
  - Gestión automática de archivos temporales
  - Sistema de limpieza automática (implementando memoria de requisitos)
- **Resultados**:
  - Continuidad de partidas entre sesiones
  - Gestión eficiente del almacenamiento
  - Mantenimiento automático del sistema

Fase 6: Testing y Desarrollo Dirigido por Pruebas (TDD)

- **Implementación**:

  - Suite completa de pruebas unitarias
  - Cobertura de casos edge y escenarios críticos
  - Metodología TDD aplicada consistentemente
- **Resultados**:
  - Alta confiabilidad del código
  - Detección temprana de errores
  - Facilita el mantenimiento y refactoring

Mejores Prácticas Implementadas

Código Limpio

- **Nomenclatura descriptiva**: Variables y métodos con nombres significativos
- **Funciones pequeñas**: Métodos con responsabilidad única
- **Comentarios informativos**: Documentación clara del propósito y funcionamiento
- **Consistencia**: Estilo de código uniforme en todo el proyecto

Encapsulamiento

- **Atributos privados**: Protección de datos internos
- **Métodos de acceso controlado**: Getters y setters apropiados
- **Validación de datos**: Verificación en puntos de entrada

Manejo de Errores

- **Excepciones específicas**: Tipos de error claramente identificados
- **Recuperación grácil**: El sistema continúa funcionando ante errores menores
- **Logging y feedback**: Información clara al usuario sobre el estado del sistema

Arquitectura Resultante

Beneficios de la Arquitectura MVC

1. **Mantenibilidad**: Cambios en una capa no afectan las otras
2. **Testabilidad**: Cada componente puede probarse independientemente
3. **Reutilización**: Componentes pueden reutilizarse en diferentes contextos
4. **Escalabilidad**: Fácil agregar nuevas funcionalidades

Robustez del Sistema

- **Tolerancia a fallos**: Manejo elegante de errores y excepciones
- **Validación exhaustiva**: Verificación de datos en múltiples niveles
- **Persistencia confiable**: Guardado seguro del estado del juego

Extensibilidad

- **Arquitectura modular**: Fácil agregar nuevas características
- **Interfaces bien definidas**: Puntos de extensión claros
- **Principios SOLID**: Base sólida para evolución futura

Métricas de Calidad

Cobertura de Código

- **Pruebas unitarias**: Cobertura completa de funcionalidades críticas
- **Casos de prueba**: Escenarios normales y edge cases
- **Validación**: Verificación de comportamiento esperado

Mantenibilidad

- **Bajo acoplamiento**: Componentes independientes
- **Alta cohesión**: Funcionalidad relacionada agrupada
- **Código autoexplicativo**: Reducción de complejidad cognitiva

Confiabilidad

- **Manejo de errores**: Recuperación grácil ante fallos
- **Validación de entrada**: Protección contra datos inválidos
- **Persistencia robusta**: Guardado confiable del estado

Lecciones Aprendidas

Diseño Orientado a Objetos

- La aplicación consistente de principios POO resulta en código más mantenible
- El diseño modular facilita significativamente el testing y debugging
- La encapsulación adecuada protege la integridad de los datos

Patrones de Diseño

- MVC proporciona una estructura clara y predecible
- La separación de responsabilidades mejora la calidad del código
- Los patrones facilitan la comunicación entre desarrolladores

Desarrollo Dirigido por Pruebas

- TDD fuerza un mejor diseño de interfaces
- Las pruebas actúan como documentación ejecutable
- La detección temprana de errores reduce costos de desarrollo

Conclusión

El proyecto Buscaminas POO representa un ejemplo exitoso de aplicación de principios de ingeniería de software en un contexto educativo. La evolución desde una implementación básica hasta un sistema robusto y profesional demuestra el valor de:

- **Planificación arquitectónica**: Diseño sólido desde el inicio
- **Aplicación de principios**: SOLID, MVC, y código limpio
- **Metodologías de calidad**: TDD y manejo robusto de errores
- **Pensamiento a largo plazo**: Diseño para mantenibilidad y extensibilidad

El resultado es un sistema que no solo cumple con los requisitos funcionales, sino que establece una base sólida para evolución futura y sirve como referencia para el desarrollo de software de calidad.