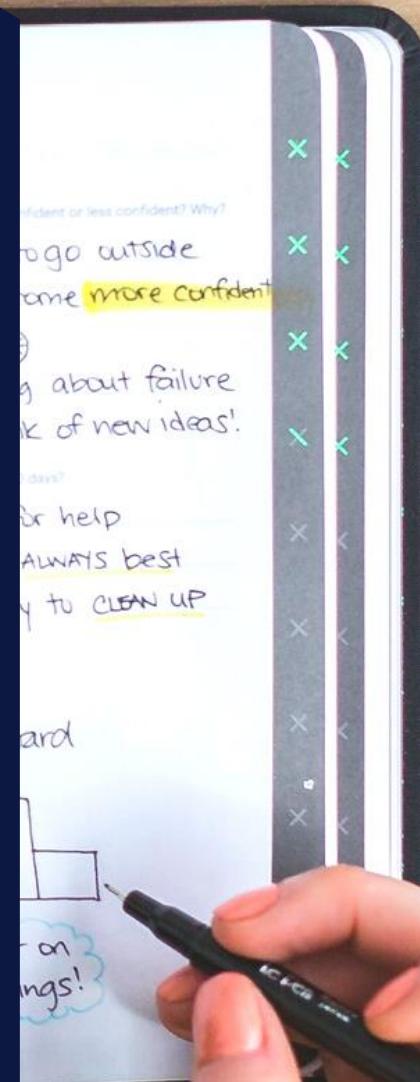




UNIVERSIDAD POLÍTÉCNICA  
**SALESIANA**  
ECUADOR

# EXAMEN PRÁCTICO



# Actividades

## Actividad: Juego de Buscaminas en Consola

El objetivo del examen final para la asignatura de **Programación Orientada a Objetos (POO)** es que los estudiantes desarrollen en equipos de 4 estudiantes una versión en consola del popular juego del **Buscaminas**. Este proyecto permitirá a los estudiantes aplicar conceptos clave de la programación orientada a objetos, manejo de archivos, diseño de software, buenas prácticas de programación, y el uso de control de versiones para la gestión del proyecto.

### Descripción del Juego:

El juego de **Buscaminas** consiste en un tablero de tamaño fijo (10x10 casillas) con una cantidad fija de minas (por ejemplo, 10 minas). El objetivo del juego es que el jugador descubra todas las casillas que no contienen minas. El juego termina cuando el jugador descubre una mina o cuando ha revelado todas las casillas seguras.

El tablero debería ser en consola como el que se muestra a continuación

	1	2	3	4	5	6	7	8	9	10
A										
B										
C										
D										
E										
F										
G										
H										

Caracteres

- **X** = Ubicación de una mina
- **V** = Espacio vacío seleccionado

### Lógica del Juego:

#### 1. Inicialización del Tablero:

- El tablero se representa mediante una matriz de 10x10. Cada casilla puede contener una mina o estar vacía.
- Se deben colocar aleatoriamente 10 minas en el tablero. Las casillas sin minas mostrarán un número que indica cuántas minas hay en las casillas adyacentes (incluyendo diagonales).
- Las casillas no descubiertas se muestran como "cubiertas" hasta que el jugador decide descubrirlas.

#### 2. Interacción del Usuario:

- El jugador introduce coordenadas (por ejemplo, A5) para seleccionar una casilla a descubrir.
- Si el jugador descubre una casilla con una mina, pierde el juego.
- Si la casilla descubierta es vacía (sin minas alrededor), el juego revela automáticamente todas las casillas adyacentes que también están vacías, hasta que encuentre casillas con números.
- El jugador puede marcar casillas que considera que contienen minas. El juego debe permitir esta opción para ayudar al jugador a evitar seleccionar una mina accidentalmente.

#### 3. Condiciones de Victoria/Derrota:

- El juego termina con una derrota si el jugador descubre una casilla que contiene una mina.
- El juego termina con una victoria si el jugador descubre todas las casillas que no contienen minas.
- El tablero debe ser mostrado al jugador después de cada acción para que tenga una visión actualizada del juego.

### Requisitos del Proyecto:

Los estudiantes deben cumplir con los siguientes requisitos para completar el proyecto:

#### 1. Implementación de Conceptos de Programación Orientada a Objetos (POO):

- **Clases y Objetos:** Definir clases para los elementos del juego, como Tablero, Casilla, Jugador, etc. Los objetos de estas clases deben gestionar diferentes partes del juego.
- **Encapsulamiento:** Asegurar que los atributos de las clases estén encapsulados correctamente, accediendo a ellos mediante métodos.
- **Uso de Repositorios GitHub:** Cada equipo debe crear un repositorio en GitHub para gestionar el código fuente del proyecto. **Cada miembro del equipo debe realizar al menos 2 commits en fechas diferentes** para demostrar la colaboración continua.

#### 2. Relación entre Clases:

- **Herencia y Polimorfismo:** Implementar clases base y subclases que hereden comportamientos comunes y utilicen polimorfismo para extender la funcionalidad del juego.
- **Paquetes e Interfaces:** Organizar las clases en paquetes para una mejor gestión del código y utilizar interfaces cuando sea necesario para definir comportamientos comunes.
- **Asociación y Dependencias:** Modelar las asociaciones entre las clases, como la relación entre Tablero y Casilla, y gestionar adecuadamente las dependencias.
- **Diagrama de Clases:** Diseñar un diagrama UML que represente las clases, atributos, métodos, relaciones, y la estructura del programa.
- **Patrón MVC:** Implementar el patrón Modelo-Vista-Controlador (MVC) para separar la lógica de negocio, la presentación y el control de la aplicación.

#### 3. Manejo de Excepciones y Archivos:

- **Excepciones:** Manejar entradas inválidas utilizando excepciones como `InputMismatchException` y `ArrayIndexOutOfBoundsException`.
- **Excepciones Personalizadas:** Crear excepciones propias, como CasillaYaDescubiertaException, para manejar situaciones específicas del juego.
- **Persistencia de datos:** Utilizar archivos de texto o la serialización de objetos para guardar el estado del juego de forma binaria y permitir al jugador continuar desde un estado guardado.

### 4. Código Limpio:

- **Técnica de Código Limpio:** Cada equipo debe elegir y aplicar una técnica de código limpio (como DRY, KISS o YAGNI) para asegurar que el código sea fácil de leer, entender y mantener.
- **Refactorización y Pruebas Unitarias:** Refactorizar el código para mejorar la estructura, eliminar redundancias y aumentar la claridad. Escribir pruebas unitarias para validar la funcionalidad de las clases y métodos.
- **TDD (Desarrollo Guiado por Pruebas):** Aplicar TDD desarrollando primero las pruebas y luego implementando el código que satisface esas pruebas.

### 5. Documentación y Uso de GitHub:

- **Documentación del Proyecto:** Documentar el código adecuadamente, explicando cada clase, método y su propósito. Incluir comentarios para clarificar la lógica del juego.
- **Informe Final:** Redactar un informe que explique los criterios de diseño utilizados, los diagramas del juego, los principios de TDD aplicados, y cómo se implementaron las excepciones, la lógica del juego y el manejo de archivos.
- **Repositorio GitHub:** Crear un repositorio en GitHub para el proyecto, incluyendo un archivo README.md con instrucciones de instalación, uso del juego, y ejemplos de ejecución. Asegurarse de utilizar commits significativos con mensajes claros y demostrar la participación de todos los miembros del equipo.

## Entrega:

Crear un informe indicando los aspectos más importantes implementados en cada etapa y sobre todo el enlace a GitHub del proyecto. Subir el informe como PDF dentro del apartado correspondiente.

## Rúbrica de Evaluación

Criterio de Evaluación	Nivel 1 (Bajo) 10% de los puntos	Nivel 2 (Medio) 50% de los puntos	Nivel 3 (Alto) 100% de los puntos	Puntos
<b>1. Implementación de Conceptos POO</b>	Uso limitado o incorrecto de clases y objetos; encapsulamiento deficiente.  0	Uso adecuado de clases, objetos y encapsulamiento, pero puede mejorarse.  1	Uso excelente de clases, objetos y encapsulamiento, implementado de manera óptima.  2	2
<b>2. Relación entre Clases y Patrón MVC</b>	Relación entre clases inadecuada; pobre implementación del patrón MVC.  0.75	Relación entre clases y uso del patrón MVC es aceptable, con algunas mejoras posibles.  1.5	Relación entre clases bien diseñada; patrón MVC aplicado correctamente, mostrando clara separación de responsabilidades.  3	3
<b>3. Manejo de Excepciones y Archivos</b>	Excepciones personalizadas y manejo de archivos incorrectos o incompletos.  0.75	Excepciones y manejo de archivos implementados correctamente, pero con algunos errores.  1.5	Excepciones y manejo de archivos bien implementados, con un manejo robusto y seguro de errores.  3	3
<b>4. Calidad de Código y Técnica de Código Limpio</b>	Código desordenado, difícil de leer y mantener; no sigue ninguna técnica de código limpio.  0.75	Código generalmente limpio y bien organizado, pero algunas partes pueden mejorarse.  1.5	Código muy limpio, bien organizado, adherente a una técnica de código limpio seleccionada.  3	3

<b>5. Desarrollo de Pruebas Unitarias y TDD</b>	Pruebas unitarias insuficientes o mal diseñadas; no se sigue TDD. 0.75	Pruebas unitarias presentes pero no exhaustivas; TDD parcialmente seguido. 1.5	Pruebas unitarias exhaustivas; TDD aplicado de manera efectiva. 3	3
<b>6. Documentación, GitHub y Participación del Equipo</b>	Documentación pobre o ausente; mala gestión del repositorio; miembros del equipo no contribuyen activamente. 0.75	Documentación adecuada; gestión del repositorio es funcional, pero necesita mejoras; algunos miembros no alcanzan los 2 commits. 1.5	Documentación clara y completa; excelente gestión del repositorio; todos los miembros realizan al menos 2 commits en fechas diferentes. 3	3
<b>7. Informe Final y Criterios de Diseño</b>	Informe ausente o incompleto; falta de detalles sobre criterios de diseño, diagramas y TDD. 0.75	Informe presente pero con detalles insuficientes o poco claros. 1.5	Informe detallado, claro y completo, cubriendo criterios de diseño, diagramas, y TDD exhaustivamente. 3	3
<b>TOTAL</b>				20