








Laboratorio 2

Competencias a desarrollar:

- Diseña los componentes de un programa que resuelvan la situación planteada usando un diagrama de clase en UML.
- Implementa un programa que corresponde al diseño elaborado y resuelve la situación planteada.
- Usa arreglos dinámicos de objetos para modelar los componentes de su solución.

Situación a resolver:

Una compañía de desarrollo de software educativo desea implementar una versión digital del clásico juego de memoria con el fin de estimular la concentración y la memoria visual de los usuarios. El sistema debe permitir que dos jugadores compitan por turnos, descubriendo pares de fichas en un tablero representado con arreglos bidimensionales.

El tablero se construye inicialmente con fichas ocultas y cada ficha tiene un símbolo asociado. Para hacer la experiencia más atractiva y sencilla en consola, se sugiere utilizar emojis como símbolos. Por ejemplo, , , , , , , . Cada símbolo debe aparecer exactamente dos veces y colocarse en posiciones aleatorias dentro del tablero al inicio de la partida.

La dinámica del juego es la siguiente:

1. En su turno, un jugador selecciona dos casillas para revelar.
2. Si las fichas descubiertas son iguales, el jugador gana un punto y conserva el turno.
3. Si las fichas no coinciden, las fichas se vuelven a ocultar y el turno pasa al otro jugador.
4. El juego finaliza cuando todas las fichas han sido emparejadas.

Al concluir la partida, el sistema debe mostrar el número de pares obtenidos por cada jugador e indicar quién ganó. Además, debe ofrecer la opción de iniciar una nueva partida. En caso de que los jugadores decidan no continuar, el sistema termina y presenta el resultado final de la sesión.

El programa debe contemplar la validación de jugadas (por ejemplo, impedir que un jugador seleccione la misma casilla dos veces en un turno o que intente destapar una ficha ya emparejada), así como la correcta alternancia de turnos entre los jugadores.

Finalmente, se espera que el diseño del sistema, con clases y métodos claramente definidos, sea lo suficientemente flexible para permitir tableros de distintos tamaños (por ejemplo, 4x4, 6x6) sin alterar la lógica principal del juego.

Entregables:

(30 puntos) Análisis:

Responde las siguientes preguntas como parte de tu análisis:

- **(5 puntos)** Se listan correctamente todos los requisitos funcionales del sistema a desarrollar, punto por punto.

Requisitos Funcionales

1. Creación del tablero: El programa generará un tablero bidimensional con las fichas ocultas, en el cual cada una de estas fichas o símbolos aparecerán dos veces en lugares aleatorios dentro del tablero.
2. Gestión de los turnos: Automáticamente deben alternarse turnos entre dos jugadores, de esta manera, cada uno puede seleccionar dos casillas por turno.
3. Jugadas validas: El programa impedirá la selección de la misma casilla dos veces en un turno, tampoco podrá destapar fichas que ya han sido emparejadas o seleccionar posiciones inválidas.
4. Puntuación: El programa mantendrá un contador de los pares descubiertos exitosamente por cada jugador y otorgarle sus puntos.
5. Comparar fichas: Al seleccionar dos casillas, el sistema comparará sus símbolos y determinará si es un par.
6. Continuidad en el turno: En dado caso un jugador encuentre un par válido, este mantendrá el turno y continuará jugando.
7. Ocultar: Cuando las fichas seleccionadas no coinciden, el programa tendrá que ocultarlas rápidamente antes de cambiar el turno.
8. Fin de juego: Cuando todas las fichas hayan sido emparejadas, el programa finalizará.
9. Reportes: A la hora de finalizar las partidas, el sistema debe de mostrar las puntuaciones correspondientes y decir el ganador.
10. Gestión de sesiones: El programa debe permitir iniciar nuevas partidas y mantener este registro de resultados completo.
11. Tamaño del tablero: Puede haber tableros de distintas dimensiones, de modo que no afecten la lógica utilizada.

- **(5 puntos)** Se describe el propósito de cada una de las clases identificadas. La descripción es lógica y coherente con lo que hace la clase en el sistema que modela y resuelve el problema presentado.

Clases del programa:

1. **Ficha:** Es la representación de una casilla individual dentro del tablero. Esta contiene la información de cada posición del tablero, también su símbolo y el estado en el que se encuentre ya sea oculta, revelada o emparejada.
2. **Tablero:** Gestiona la estructura bidimensional del juego y sus acciones con el conjunto de fichas. Esta clase es la encargada de inicializar aleatoriamente, validar las posiciones y mantener el control del tablero.
3. **Jugador:** Representa a cada participante del juego, manteniendo su información personal y estadísticas. Enfoca los datos relacionados con cada jugador para facilitar la gestión de los turnos y las puntuaciones.
4. **ControladorJuego:** Maneja la lógica principal del juego, coordinando las interacciones entre jugadores, tablero y fichas. Toma en cuenta las reglas del juego y el flujo de partida.
5. **Principal:** Maneja toda la interacción con el usuario usando la consola. Se encarga de recopilar la entrada del usuario, mostrar la información del juego, y presentar los resultados.

Para cada clase:

- **(10 puntos)** se describe el propósito de cada atributo. La descripción es suficiente para comprender la razón de la creación del atributo y su importancia como propiedad de las instancias (o clase).
- **(10 puntos)** Se describe el propósito de la creación de cada método. La descripción es suficiente para comprender qué acciones realizará cada método, qué información requerirá para funcionar (parámetros) y que resultados producirá.

Atributos y métodos de las clases

Ficha

Atributo	Tipo	Descripción
simbolo	String	Emoji que representa el contenido de la ficha.
estaRevelada	boolean	Indica si la ficha está visible.
estaEmparejada	boolean	Indica si la ficha ya fue emparejada.

Método	Parámetros	Retorno	Descripción
Ficha()	String simbolo	-	Constructor, inicializa ficha con símbolo.
revelar()	-	void	Hace visible la ficha.
ocultar()	-	void	Oculto la ficha nuevamente.
emparejar()	-	void	Marca la ficha como emparejada.
getSimbolo()	-	String	Retorna el símbolo de la ficha.
isRevelada()	-	boolean	Retorna si la ficha está revelada.
isEmparejada()	-	boolean	Retorna si la ficha está emparejada.

Tablero

Atributo	Tipo	Descripción
fichas	Ficha[][]	Matriz bidimensional del tablero.
filas	int	Número de filas del tablero.
columnas	int	Número de columnas del tablero.
simbolosDisponibles	String[]	Array de símbolos disponibles para el juego.

Método	Parámetros	Retorno	Descripción
Tablero()	int filas, int columnas	-	Constructor que inicializa tablero.
inicializarFichas()	-	void	Crea y distribuye fichas de forma aleatoria.
seleccionarFicha()	int fila, int columna	Ficha	Revela ficha en la posición especificada.
esSeleccionValida()	int fila, int columna	boolean	Valida si la selección si es permitida.
ocultarFichas()	Ficha ficha1, Ficha ficha2	void	Ocultar dos fichas que no coinciden.
todasEmparejadas()	-	boolean	Verifica si todas las parejas están emparejadas.
mostrarTablero()	-	String	Genera la representación visual del tablero.

Jugador

Atributo	Tipo	Descripción
nombre	String	Nombre del jugador.
paresEncontrados	int	Contador de pares encontrados en la partida.
partidasGanadas	int	Total de partidas ganadas en la sesión.

Método	Parámetros	Retorno	Descripción
Jugador()	String nombre	-	Constructor el cual inicializa al jugador.
incrementarPares()	-	void	Aumenta contador de pares.
incrementarPartidasGanadas()	-	void	Aumenta partidas ganadas.
reiniciarPares()	-	void	Reinicia contador para nueva partida.
getNombre()	-	String	Retorna nombre del jugador.
getParesEncontrados()	-	int	Retorna pares encontrados.
getPartidasGanadas()	-	int	Retorna partidas ganadas.

ControladorJuego

Atributo	Tipo	Descripción
tablero	Tablero	Referencia al tablero activo.
jugadores	Jugador[]	Array con los dos jugadores.
jugadorActual	int	Índice del jugador con turno activo.
juegoTerminado	boolean	Indica si la partida ha finalizado.

Método	Parámetros	Retorno	Descripción
ControladorJuego()	int filas, int columnas, String[] nombres	-	Constructor que inicializa el juego.
jugarTurno()	-	void	Gestiona turno completo de jugador.
procesarSelecciones()	Ficha ficha1, Ficha ficha2	boolean	Procesa par de fichas seleccionadas.
cambiarTurno()	-	void	Cambia al siguiente jugador.
finalizarPartida()	-	void	Determina ganador y actualiza estadísticas.
iniciarNuevaPartida()	-	void	Reinicia para una nueva partida.
obtenerEstadoJuego()	-	String	Retorna la información del estado actual.

Principal

Atributo	Tipo	Descripción
scanner	Scanner	Objeto el cual captura la entrada del usuario.
juego	ControladorJuego	Referencia al juego principal o también conocido como controlador.

Método	Parámetros	Retorno	Descripción
main()	String[] args	void	Punto de entrada del programa
mostrarMenu()	-	void	Muestra opciones disponibles
solicitarDimensiones()	-	int[]	Captura dimensiones del tablero
solicitarNombresJugadores()	-	String[]	Captura nombres de jugadores
solicitarSeleccion()	-	int[]	Pide coordenadas de selección
mostrarResultados()	-	void	Presenta puntuación y estadísticas
confirmarNuevaPartida()	-	boolean	Pregunta si continuar jugando

(30 puntos) Diseño:

Desarrolla un diagrama de clases congruente con las respuestas del análisis que muestre los miembros de cada clase y que establezca una asociación entre ellas:

- **(8 puntos)** La representación de las clases coincide con el estándar del lenguaje UML.
- **(12 puntos)** Están representados cada uno de los componentes de la clase de manera correcta y coinciden con lo descrito en el análisis.
- **(10 puntos)** Las relaciones entre las clases están correctas

(40 puntos) Implementación:

Programa la solución diseñada, recuerda que la clase de interacción con el usuario es la que debe recopilar los datos del usuario y mostrar los resultados de las acciones.

- **(5 puntos)** La cantidad de clases identificadas es suficiente para resolver el problema planteado.
- **(10 puntos)** Uso de programación defensiva y manejo de excepciones.
- **(20 puntos)** El problema se resolvió empleando programación orientada a objetos. Se almacenó la información necesaria creando y llenando listas de objetos y objetos de las clases identificadas. El programa compila y corre.

- **(5 puntos)** El código fuente está suficientemente comentado para comprender qué se hace, incluyendo el encabezado con la información del creador, descripción del programa y crédito a toda fuente de información que haya sido empleada para el desarrollo del programa. Preferiblemente, que también incluya fecha de creación y fecha de última modificación.

Puntos Up (Hasta 15 puntos):

- (+15 puntos) Uso de interfaz gráfica.

Puntos Down (Hasta 100 puntos):

- (-30 puntos) Por cada While true y su correspondiente break.
- (-10 puntos) Por cada mensaje que muestre al usuario (System.out.println) que haga fuera de la clase Principal.
- (-10 puntos) Por cada uso del Scanner que haga fuera de la clase Principal.

Material a entregar en canvas

- Archivo .pdf con el análisis.
- Archivo .jpg o .png que presente el diagrama de clases requerido.
- Archivos .java con el código que implementa las clases y el *driver program* requerido