Juego de memoria

# Análisis del Sistema (30 puntos)

## 1.1 Requisitos funcionales del sistema (5 pts)

Escribir los puntos exactos que debe cumplir tu sistema.

Ejemplo: “El usuario puede crear una nueva solicitud de boletos”, etc.

- El usuario debe ser capaz de ingresar las dimensiones de un tablero de cartas (solo par)

- El usuario debe ser capaz de ingresar el nombre de dos jugadores

- El usuario debe ser capaz de, en su turno, ingresar dos coordenadas de tarjetas (A1), y verificar si consiguió o no un par. Si consiguió un par, se le asigna un punto y estas dos tarjetas se quedan reveladas. De lo contrario, se cambia el turno al del otro jugador.

- El programa debe ser capaz de generar un tablero de tamaño x\*x, siendo que x es un numero par.

- El programa debe ser capaz de llenar el tablero con pares de cartas, de forma que cada tipo de carta se encuentre únicamente 2 veces.

Nota importante: para poder visualizar los emojis de cada tarjeta, es necesario usar la terminal de powershell, y en ella correr los siguientes comandos previo a usar el programa:

* chcp 65001
* [Console]::OutputEncoding = [System.Text.Encoding]::UTF8

## 1.2 Clases necesarias y su propósito (5 pts)

|  |  |
| --- | --- |
| Clase | Propósito |
| Tarjeta | Representa una tarjeta de la cual se busca su pareja. Cuenta con un código numérico (que representa un emoji ASCII) y un estado de si se encuentra revelada. |
| Tablero | Representa el tablero en el cual se encuentran los pares de tarjetas. Cuenta con un tamaño fijo (ingresado por el usuario) |
| Distribuidor de tarjetas | Esta clase ayuda al tablero a distribuir sus pares de tarjetas. Cuenta con un contador de cuantas tarjetas faltan por distribuir (empieza en 2) y el objeto de tarjeta a asignarse. |
| Jugador | Representa uno de los jugadores que juega a la memoria. Cuenta con un nombre (ingresado por el usuario), una cantidad de puntos de la ronda (empieza en 0) y una cantidad de rondas ganadas (empieza en 0). |
| Consola | Es la clase encargada de mostrar los datos del tablero. Cuenta con un tablero de tarjetas. |
| Controlador | Es la clase encargada de relacionar el Tablero con los jugadores. Cuenta con 1 tablero, 2 jugadores y 1 consola. |
| Lista de emojis | Esta clase contiene una lista de códigos Unicode usados para representar los emojis de las tarjetas. No requiere la creación de un objeto, y cuenta con métodos para obtener el código de 1 emoji. |
| Vista | Es la clase encargada de pedir y mostrar datos al usuario. Es la que lleva el flujo de control del programa, y se relaciona únicamente con el controlador. |

## 1.3 Atributos de cada clase (10 pts)

Clase: Tarjeta

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Codigo emoji | String | privada | Es el código ASCII para el emoji de la carta. |
| Pareja encontrada | boolean | privada | Indica si ya se ha encontrado la pareja de esta tarjeta. En caso de haberse encontrado, ya no podrá seleccionarse o dar puntos |

Clase: Tablero

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Tamaño | int | privada | Es el tamaño del tablero. Puesto que el tablero siempre tiene forma cuadrada, no es necesario añadir el dato de la segunda dimensión |
| Celdas | Tarjeta[][] | privada | Es la lista de tarjetas con las que cuenta el tablero. |
| Lista de distribuidores | ArrayList <Distribuidor Tarjeta> | privada | Es la lista de distribuidores de tarjetas que el tablero tiene. Se usa al momento de generar las tarjetas del tablero. |
| RNG | Random | privada | Es un generador de números aleatorios. Se usa al momento de generar las tarjetas del tablero. |

Clase: Distribuidor de tarjetas

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Tarjetas por distribuir | int | privada | Es el número de tarjetas de un tipo que faltan por distribuirse. Suele empezar en 2, pero puede cambiarse |
| Tarjeta a distribuir | Tarjeta | privada | Es el tipo de tarjeta que se asignará si se usa el distribuidor. |

Clase: Jugador

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Nombre | String | privada | Es el nombre del jugador |
| Puntos de ronda | Int | privada | Es la cantidad de puntos que el jugador ha ganado en esta ronda. Cada vez que se obtiene una pareja de tarjetas, el jugador obtiene 1 punto |
| Rondas ganadas | Int | Privada | Es la cantidad de rondas que el jugador ha ganado en una sesión |

Clase: Consola

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Tablero a usar | Tablero | privada | Es el tablero de tarjetas el cual debe visualizarse. |

Clase: Lista de emojis

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Emojis | String[] | privada | Es una lista que contiene códigos Unicode de diferentes emojis. Los códigos siguen el siguiente formato: \uXXXX\uXXXX, en donde cada X es un número de formato hexadecimal. |
| Código de emoji oculto | String | privada | Es el código Unicode del emoji de un cuadrado blanco. Se usa para representar tarjetas sin revelar |
| Emojis sin usar | ArrayList<String> | privada | Es una copia de la lista de emojis. Sin embargo, esta lista se actualiza conforme se usen emojis en el sistema, de forma que no se repita un emoji ya usado. Se relaciona principalmente con los distribuidores de tarjetas. |
| RNG | Random | Privada | Es un generador de números aleatorios. Se usa principalmente al dar el código de un emoji. |

Clase: Controlador

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Tablero a usar | Tablero | privada | Es el tablero de tarjetas que se usarán para jugar. |
| Jugador 1 | Jugador | privada | Es uno de los jugadores que jugará al juego de memoria |
| Jugador 2 | Jugador | Privada | Es otro de los jugadores que jugará al juego de memoria |
| Jugador activo | Jugador | Privada | Indica cual de los dos jugadores tiene el turno actualmente. |

Clase: Vista

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Controlador | Controlador | privada | Es el controlador que se usará para manejar las operaciones del juego de memoria |
| Teclado | Scanner | privada | Es el método con el cual el usuario ingresa sus datos. |

### 1.4 Métodos de cada clase (10 pts)

Clase: Tarjeta

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros: Tipo de dato | Salida: Tipo de dato | Visibilidad | Propósito |
| Ninguno |  |  |  |  |

Clase: Tablero

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros: Tipo de dato | Salida: Tipo de dato | Visibilidad | Propósito |
| Generar tarjetas | Ninguno | Void | privada | Este método genera y asigna diferentes tarjetas a las celdas del tablero. Para ello, genera x cantidad de distribuidores de tarjetas, siendo que . Después de esto, recorre las celdas del tablero, asignándole a cada celda una tarjeta de un distribuidor. |
| Asignar tarjetas | ninguno | Tarjeta | Privada | Este método obtiene la tarjeta de un distribuidor seleccionado al azar de la lista de distribuidores. Después, se revisa si el distribuidor aun puede dar más tarjetas, y en caso de no poder se elimina de la lista. Al final, se regresa la tarjeta obtenida previamente. |
| Evaluar tarjetas | Coordenada X: int  Coordenada Y: int  Puede haber sido revelada: boolean | Tarjeta | privada | Este método revisa si una tarjeta puede usarse o no. Para ello, revisa si las coordenadas se encuentran en el plano del tablero, y revisa si la tarjeta ya se ha revelado o no. Si alguno de estos casos sucede, lanza una excepción adecuada. Si no ocurre ningún caso, regresa la tarjeta de la localización ingresada. |
| Emparejar tarjetas | Coordenada X1: int  Coordenada Y1: int  Coordenada X2: int  Coordenada Y1: int | Boolean | publica | Este método verifica si dos tarjetas del tablero son pares. En caso de ser pares, ambas tarjetas se marcan como reveladas y se regresa true. De lo contrario, se regresa false. |

Clase: Distribuidor de tarjetas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros: Tipo de dato | Salida: Tipo de dato | Visibilidad | Propósito |
| Dar tarjeta | ninguno | Tarjeta | publica | Este método regresa la tarjeta asignada al distribuidor propio, y baja en 1 la cuenta de tarjetas por asignar. |

Clase: Jugador

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros: Tipo de dato | Salida: Tipo de dato | Visibilidad | Propósito |
| Sumar punto | ninguno | void | Publica | Este método suma 1 a la cuenta de puntos de la ronda del jugador. |
| Reiniciar puntos | Ninguno | void | privada | Este método cambia la cuenta de puntos de la ronda del jugador a 0. |
| Resultado ronda | Gano la ronda | Void | Publica | Si el jugador gano la ronda, le suma 1 a su cuenta de rondas ganadas y reinicia sus puntos de ronda. De lo contrario, solamente reinicia sus puntos de ronda. |

Clase: Consola

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros: Tipo de dato | Salida: Tipo de dato | Visibilidad | Propósito |
| Mostrar tablero | Ninguno | Ninguno | Publica | Imprime en la terminal la representación del tablero de tarjetas. Para ello, coloca en cierta posición el emoji que cada tarjeta tiene, y en caso de que esta tarjeta no esté revelada solamente muestra un cuadro negro. |

Clase: Controlador

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros: Tipo de dato | Salida: Tipo de dato | Visibilidad | Propósito |
| Jugar turno | Coordenada X1: int  Coordenada Y1: int  Coordenada X2: int  Coordenada Y2: int | boolean | Publica | Este método recibe las coordenadas de dos tarjetas, y verifica si hacen match. En caso de hacer match, se le suma 1 punto al jugador actual y se regresa true. Si no hacen match, se cambia el jugador actual al siguiente y se regresa false. |
| Imprimir tablero | Ninguno | Ninguno | Publica | Este método llama a la consola, desde donde se imprime el estado actual del tablero. |
| Mostrar información jugador | Ninguno | String | Publica | Este método muestra la cantidad de puntos y la cantidad de rondas que el jugador actual ha ganado. |

Clase: Vista

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros: Tipo de dato | Salida: Tipo de dato | Visibilidad | Propósito |
| Convertir letra a numero | String | int | Publica | Este método convierte una letra a su equivalente numérico, suponiendo que la primera letra del abecedario (A) equivale a 1 |

### 2. Diseño: Diagrama de Clases (30 puntos)

* Asegúrate de mostrar atributos y métodos con visibilidad (+, -).
* Indica relaciones entre clases (asociación, agregación, etc.).
* Incluye el driver program (Main).

<https://lucid.app/lucidchart/e31aa944-7546-41c8-98b2-e0966acbd810/edit?view_items=LrWNfp7cTp.g&invitationId=inv_e3ba9561-1219-4e98-b03a-6558f6dc1359>

### 3. Programa (40 puntos)

En cada archivo `.java`, asegurarse de incluir:

* Las clases necesarias.
* Uso adecuado de objetos.

Menú que debe implementar el driver program:

* 1. Nuevo comprador
  2. Nueva solicitud de boletos
  3. …
  4. …
  5. Salir

GitHub: colocar aquí la URL:

<https://github.com/Oscarro11/Lab2-POO>

# Checklist antes de entregar

Está claro el análisis?

El diagrama tiene los elementos UML correctamente?

Subiste tu código a GitHub con todo lo necesario?