**Tarea online 09**

**Título de la tarea:**Creando aplicaciones gráficas de escritorio: Juego de Memoria.

**Unidad:**09

**Ciclo formativo y módulo:**DAM/DAW, Programación.

**Curso académico:**2023/24

**¿Qué contenidos o resultados de aprendizaje trabajaremos?**

RA 5: Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.

En esta unidad se introducen las bases para el desarrollo de interfaces gráficos.

El enfoque gráfico de las aplicaciones va a permitir que sean más intuitivas y amigables para el usuario.

Para comenzar a desarrollar este tipo de aplicaciones deben conocerse los controles básicos:

* contenedores
* cuadros de texto
* casilas de verificación
* botones de radio
* botones
* menus
* listas
* ventanas emergentes de diálogo
* imágenes

Conocer su funcionamiento básico y sus eventos asociados permitirá abordar el uso de otros más avanzados.

También es fundamental haber entendido la gestión de los eventos.

# 1.- Descripción de la tarea

# Caso práctico

[PikiSuperStar (Freepik)](https://www.freepik.es/vector-gratis/tarjetas-juego-memoria-dibujadas-mano_37189017.htm). Juego de Memoria ([CC BY](http://creativecommons.org/licenses/?lang=es))

**Juan** se ha planteado un pequeño reto. Esta vez no se trata de un trabajo para la empresa donde trabaja, sino un pequeño desafío que se ha propuesto.

Ha encontrado un **juego de memoria** (típico juego de encontrar parejas en un tablero de cartas) implementado en modo texto, con entradas desde el teclado y salida a través de la consola. Ha pensado que sería una buena idea adaptar ese programa a una aplicación con interfaz gráfica sencilla, de forma que se pueda jugar de una forma mucho más visual.

**Ana**, una amante de los desafíos informáticos, se une a **Juan** en su proyecto de crear una versión de un juego de memoria en JavaFX. Juntos, se embarcan en esta aventura de aprendizaje y desarrollo.

Deciden comenzar por el diseño de la interfaz gráfica. **Ana**, con su habilidad para entender los conceptos de diseño de usuario, sugiere una disposición limpia y atractiva para el tablero de juego. Proponen una cuadrícula de cartas boca abajo, cada una representando un elemento del juego de memoria.

**Juan**, con su experiencia en Java y JavaFX, se encarga de la implementación técnica. Utiliza las herramientas proporcionadas por JavaFX para crear la ventana de la aplicación y la estructura de la cuadrícula de cartas. Define las interacciones necesarias para voltear las cartas y compararlas cuando el jugador selecciona dos.

Conforme avanzan en el desarrollo, enfrentan desafíos técnicos. Por ejemplo, deben asegurarse de que las cartas se barajen de manera aleatoria al inicio del juego y de que la lógica de comparación de parejas funcione correctamente. Además, incorporan efectos visuales para dar una retroalimentación clara al jugador sobre su progreso en el juego.

Una vez que terminan el desarrollo inicial, prueban la aplicación juntos. Descubren que funciona bien, pero deciden realizar algunas mejoras adicionales, como agregar sonidos para aumentar la experiencia del jugador y crear un sistema de puntuación para incentivar la competencia amistosa.

#### ¿Qué te pedimos que hagas?

Las actividades a realizar se centrarán en la **creación de una aplicación gráfica de escritorio consistente en un "juego de memoria"**. Para ello, deberás **adaptar a JavaFX el juego "MemoriaConsola"** que se proporciona en el ***Apartado 2.- Información de interés***.Además, en el apartado **Introducción - Proyecto de consola** puedes encontrar más información sobre el código de ese proyecto.

Deberás hacer las adaptaciones necesarias, entre ellas:

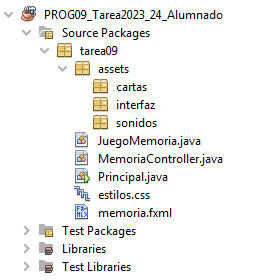
* El juego debe **mostrarse en un entorno gráfico**.
* Los **controles del juego** se realizarán con el **ratón**.
* Se utilizará un **tablero** de dimensiones **4x4** (16 casillas) que esconderán **8 parejas elegidas**de una lista de personajes mayor (para que siempre no se repitan los mismos personajes).
* El programa contemplará la posibilidad de **iniciar una partida nueva**y **salir de la aplicación**.
* Se recomienda utilizar una **representación gráfica** de las cartas (se ofrecen varias lista de personajes, pero se pueden usar otras imágenes).
* Se incluirá un **temporizador** que contabilice el **total de segundos** que dura la partida.
* Se incluirá un **contador de intentos** que se incrementará cada vez que se trate de descubrir una pareja.
* Al descubrir una pareja válida, sus cartas **desaparecerán** del tablero.
* Al descubrir todas las parejas, se debe mostrar un**mensaje de finalización** de la partida.

Para el desarrollo de la aplicación puedes usar los siguientes controles:

* Cuadros y etiquetas de texto.
* Botones con texto y botones con gráficos.
* Fondo de la zona de juego.
* Uso de imágenes para las cartas.
* Ventanas emergentes para mensajes del juego.
* Estilos CSS de JavaFX para personalizar algún/os aspectos de los distintos elementos (textos, botones, cartas, etc).
* Efectos de sonido.

Aunque os hemos preparado una propuesta de diseño visual del juego, podéis **usar vuestra creatividad para desarrollar otra propuesta** tanto de interfaz de juego como de personajes. La propuesta que se realice debe ser **coherente** en cuanto a imágenes, fondos, colores, sonidos, etc.

##### Estructura del proyecto base

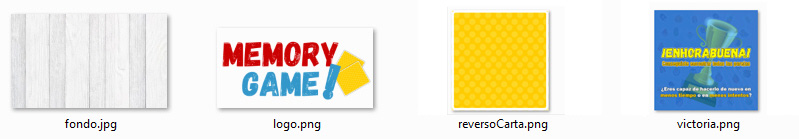
Estructura del proyecto

En el apartado **2.- Información de interés** encontrarás el proyecto base **PROG09\_Tarea2023\_24\_Alumnado** que debes descargar para comenzar a trabajar. La estructura del proyecto se muestra en la imagen de la derecha. Como puedes ver, contendrá varios archivos y paquetes (carpetas) que pasamos a comentar a continuación:

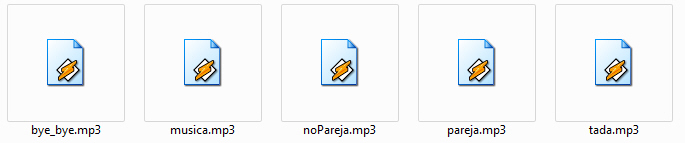
* **Archivo JuegoMemoria.java:**es la clase que implementa la lógica del juego (inicialización y gestión del tablero, comprobación de parejas, etc).
* **Archivo MemoriaController.java:** es la clase encargada de gestionar la relación entre la interfaz y el funcionamiento del juego. Permite enlazar los distintos elementos de la interfaz con la funcionalidad necesaria, desarrollada en este archivo.
* **Archivo Principal.java:** Es el archivo lanzador del juego. Define la escena principal y la inicia.
* **Archivo memoria.fxml:** Este archivo define la **interfaz gráfica de la aplicación**. Se podría editar directamente (es un archivo XML) pero es mucho más cómodo desarrollarla utilizando el programa **Scene Builder**. En el proyecto base, únicamente dispondrá de un contenedor principal de tipo AnchorPane al que se le añadirán todos los elementos de interfaz que se quieran utilizar.
* **Archivo estilos.css:** Este archivo permite crear estilos JavaFX CSS para los distintos elementos de la interfaz. Para que los estilos se puedan aplicar correctamente este archivo debe estar enlazado al AnchorPane principal (se puede enlazar desde Scene Builder a través de la propiedad Stylesheets).
* **Paquete assets:** contendrá todos los recursos gráficos que utilizará la aplicación: imágenes de las cartas, elementos de interfaz y recursos de audio. Los recursos se organizarán en 3 subpaquetes:
  + **Subpaquete cartas:** contendrá las imágenes de las distintas cartas. Puedes crear dentro varias subcarpetas si quieres plantear distintos juegos temáticos de parejas (por ejemplo: personas, escudos, plantas, etc). Puedes descargas algunos **paquetes de imágenes para las cartas** que se ofrecen a continuación o utilizar un conjunto de imágenes para las cartas de tu elección:

|  |  |  |
| --- | --- | --- |
| Assets Personas | Assets Escudos | Assets Plantas |
| **Asset Personas** [**[ DESCARGAR ]**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/personas.zip) | **Asset Escudos de fútbol** [**[ DESCARGAR ]**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/escudos.zip) | **Asset Plantas** [**[ DESCARGAR ]**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/plantas.zip) |

* + **Subpaquete interfaz:** contendrá elementos relacionados con la interfaz del juego, por ejemplo: imagen para el fondo del escenario, imagen para el reverso de la carta, logotipo con el nombre del juego, imagen para la pantalla de victoria, etc.

**Recursos gráficos para la interfaz**  
[**[ DESCARGAR ]**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/interfaz.zip)

* + **Subpaquete sonidos:** en este paquete se pueden colocar los recursos de audio que utilice el juego. Algunos ejemplos podrían ser: música de fondo, efecto de sonido para pareja válida, efecto de sonido para pareja no válida, música o efecto de sonido para la pantalla de victoria, etc.

**Recursos de audio**[**[ DESCARGAR ]**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/sonidos.zip)

Ten en cuenta que, aunque se aportan recursos gráficos y sonidos de muestra, sería muy interesante que plantearas **tu propia versión del juego** (con la temática que desees). De esta forma, tendrías una **trabajo personalizado que incluir en tu portfolio personal**.

# 1.1.- Ejercicio 1: Creando la interfaz básica de la aplicación

El primer paso que podemos realizar para comenzar a crear nuestro juego con JavaFX es **diseñar la interfaz de nuestra aplicación**. Para ello, puedes utilizar la aplicación gráfica ***Scene Builder***, la cual nos facilitará mucho conseguir un diseño vistoso y agradable.

##### Diseño de la interfaz utilizando Scene BuilderElementos disponibles

Puedes añadir distintos elementos (botones, paneles, imágenes, contenedores, etc) seleccionando cada uno de ellos en la lista de elementos disponibles y arrastrándolos a tu interfaz.

Se propone la definición de, al menos, los siguientes elementos visuales:

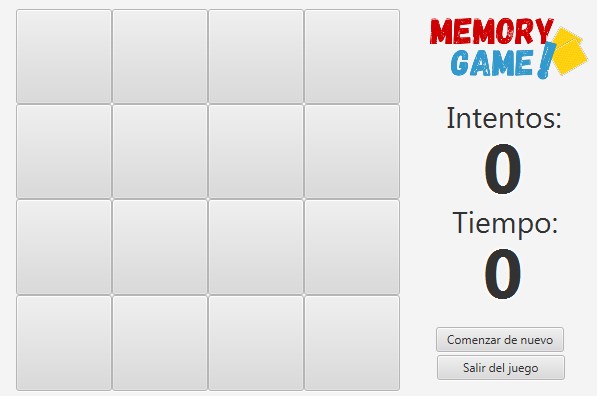
* **Fondo gráfico** para el escenario.
* Tablero de juego con **16 cartas** (puedes utilizar la estructura que prefieras).
* **Imagen gráfica** con el título del juego (por ejemplo Memory Game!, puedes diseñar una imagen propia con el título de tu juego).
* **Textos fijos**.
* **Textos dinámicos** (para el contador de tiempo y el temporizador).
* Botones de acción para **comenzar una partida** o **salir del juego**.

En el proyecto base se aportan **distintos recursos gráficos**. Puedes utilizarlos para construir la interfaz o bien diseñar otros **recursos propios** para dotar a tu juego de un aspecto **más personal** o diseñarlo dentro de una **temática concreta**.

# Ejercicio Resuelto

Principio del formulario

Una **propuesta de diseño de la interfaz** podría ser la siguiente (puedes basarte en esta propuesta o plantear un diseño personalizado a tu gusto).

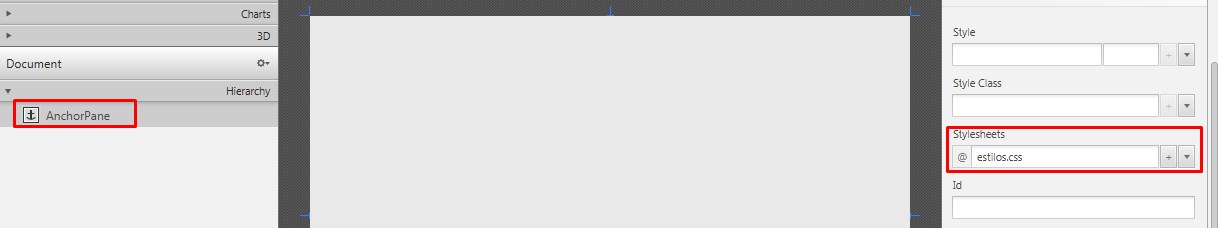
Propuesta de Interfaz básica del juego Memory Game!

Final del formulario

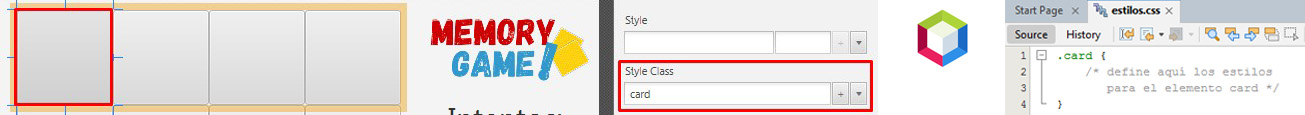
# 1.2.- Ejercicio 2: Mejorando con estilos el aspecto de la interfaz diseñada

Una vez, hayas definido la interfaz principal del juego, puedes aplicar algunos estilos visuales para mejorar el aspecto de los componentes.

Aunque desde la propia interfaz de Scene Builder se pueden incluir algunas **personalizaciones sobre cada elemento** actuando sobre su panel **Properties**, en nuestro caso **aplicaremos estas personalizaciones a través de una hoja de estilos CSS** (**estilos.css**) que, como puedes ver en la imagen, ya está vinculada al panel principal (AnchorPane) en el proyecto base.

Hoja de estilos (estilos.css) vinculada al panel principal (AnchorPane)

Para aplicar estilos a un elemento podemos asignarle una clase CSS del mismo modo que se hace en HTML. Una vez vinculada la clase al elemento, podremos escribir las reglas CSS directamente en el fichero **estilos.css** en NetBeans. En la siguiente imagen, se muestra **cómo se ha asociado la clase .card a una de las cartas del tablero** y cómo establecer la **vinculación CSS** y **escribir las reglas de estilo para esta clase**.

Estilos JavaFX CSS para la clase .card que da estilo personalizado a las cartas del juego

**Importante:** Incluye en tu fichero **estilos.css**, al menos, **15 reglas de estilo JavaFX CSS** (tipos de letra, colores, tamaños de letra, fondos de color o de imagen, efectos de borde o sombreado, etc) para aplicar a los elementos de tu diseño. Algunos ejemplos donde podrás aplicar estilos: botones, cartas, textos, imágenes, panel principal, etc.

# Debes conocer

Recuerda que todas las reglas JavaFX CSS deben incluir el prefijo **-fx-** característico. Por ejemplo: -fx-font-size, -fx-font-family, entre otras.

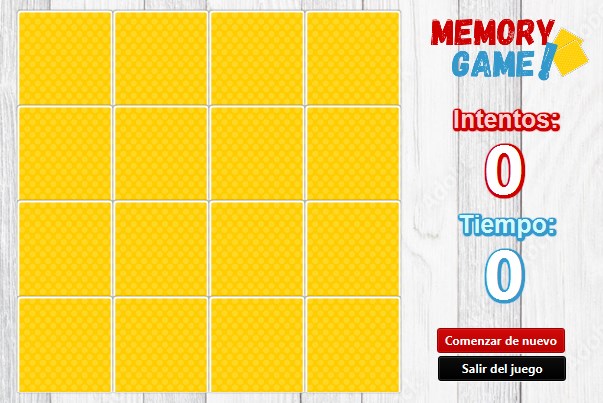
Te recomendamos consultar la **referencia oficial de JavaFX CSS** para conocer qué reglas de estilo se pueden aplicar a los distintos elementos que utilices en tu diseño y elegir las que más te gusten:

[**https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html**](https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html)

# Ejercicio Resuelto

Principio del formulario

Un ejemplo de resultado que podríamos obtener tras diseñar nuestra interfaz y personalizarla con estilos JavaFX CSS podría ser el siguiente:

Interfaz con estilos JavaFX CSS personalizados

Por supuesto, esto es solo una propuesta. Puedes plantear tu propio diseño. **¡Muestra tu creatividad!**

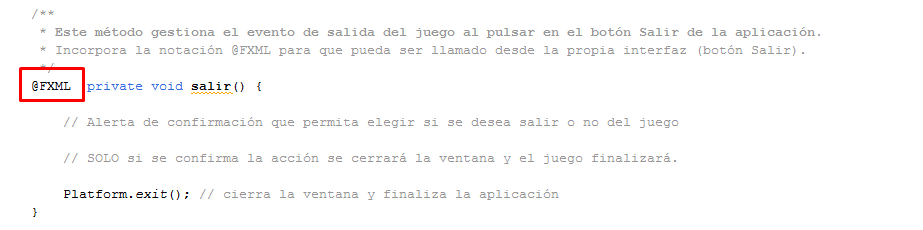
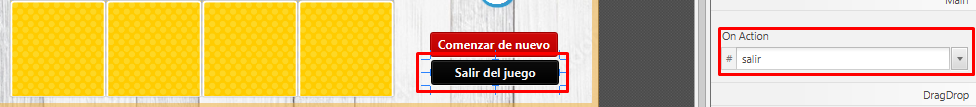
Final del formulario

# 1.3.- Ejercicio 3: Vinculando los elementos de la interfaz con la clase MemoriaController

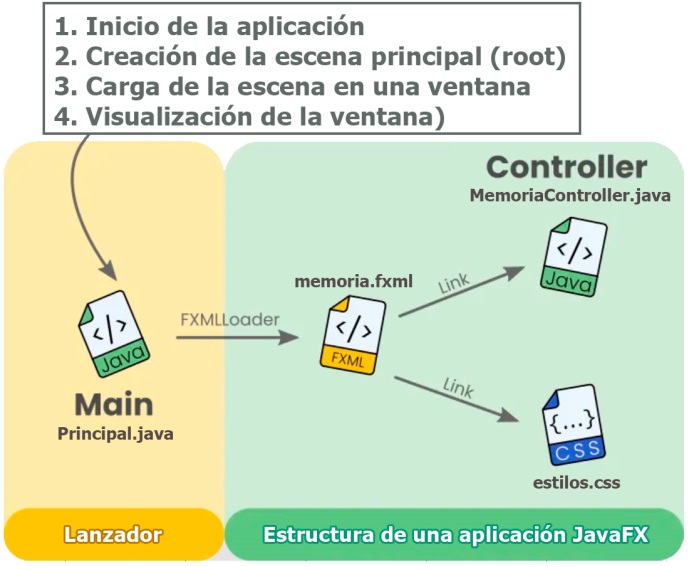
La clase **MemoriaController**será una de las más importantes del proyecto ya que relacionará la vista (interfaz) con la lógica de la aplicación (métodos desarrollados para implementar la lógica del juego).

##### Vinculación de elementos de la interfaz con variables y métodos de la clase MemoriaController

Para conseguir esta vinculación entre los elementos de la interfaz y las variables y métodos definidos en la clase controladora haremos lo siguiente:

* Primeramente, en la clase MemoriaController crearemos distintos identificadores para nombrar a cada uno de los elementos de la interfaz con los que queramos interactuar. Por ejemplo: cada una de las cartas, el contador de tiempo o de intentos o los botones Salir/Comenzar. Un ejemplo podría ser el siguiente:  
  FXML para intentos y tiempo  
  **IMPORTANTE: Presta atención a la notación @FXML antes de las variables. Esta notación es la que permite enlazar elementos del controlador con la interfaz**
* Del mismo modo, podemos definir los métodos que se llamarán desde nuestra interfaz como respuesta a los eventos que ocurran en ella (observa que, de nuevo, se utiliza la **notación @FXML** para poder vincular este método con algún evento de la interfaz). Un ejemplo podría ser el método que asociaremos al botón para Salir de la aplicación:  
  
* Asignar los **identificadores** a aquellos elementos que queramos utilizar **desde el código (MemoriaController) de la aplicación**. Para hacer esto debemos señalar cada elemento y en el panel **Code** e indicar en el apartado **fx:id** el mismo **identificador único que hemos creado en la clase MemoriaController**para que quede asociado a dicho elemento.  
  
* Además, si al interactuar con el elemento se debe realizar la llamada a algún método, debemos indicarlo también en el panel **Code** indicando el **nombre del método a lanzar** y el **tipo de evento que debe ocurrir** para que se lance la llamada a ese método. Habitualmente usaremos el evento **On Action** que corresponde al clic de ratón sobre el elemento pero tenemos una gran cantidad de eventos disponibles tanto de ratón como de teclado. Por ejemplo:  
    
  

**1.4.- Ejercicio 4: Programando la lógica de la aplicación**

[](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/estructuaJavaFX.jpg)*Estructura de un proyecto JavaFX (pulsa en la imagen para ampliar)*

Una vez ya hayamos definido y vinculado los elementos de la interfaz es el momento de iniciar la programación de la lógica de la aplicación que permita implementar un juego de buscar parejas.  
  
La lógica de funcionamiento del juego estará implementada en la clase **MemoriaController**, que se comunicará con la vista de la aplicación (**memoria.fxml**) y gestionará a los eventos que se envíen desde la misma.

Los métodos que se propone desarrollar para conseguir la funcionalidad del juego se detallan a continuación:

* Método **public void initialize(URL url, ResourceBundle resourceBundle)** : este método es **llamado automáticamente** cuando se inicializa un controlador asociado a un archivo FXML (en este caso, cuando se carga el archivo de interfaz **memoria.fxml**).  
    
  Su misión es configurar todos los aspectos necesarios para **comenzar una nueva partida**: inicializar la instancia de juego que controlará la partida (instancia de la clase **JuegoMemoria**), gestionar la lista de cartas, configurar el contador de tiempo, poner los intentos a 0, etc.

**Debes conocer**

[Imagen de pngtree.com](https://pngtree.com/freepng/3d-icon-of-vintage-sand-clock-time-and-deadline-business-concept_7265511.html)

El contador de tiempo se puede gestionar con una instancia de la clase **javafx.animation.Timeline**. Esta clase permite **realizar acciones cada determinado tiempo,**definido por una duración recibida como parámetro.

Puedes obtener información sobre esta clase en la API de JavaFX:  [**https://docs.oracle.com/javase/8/javafx/api/javafx/animation/Timeline.html**](https://docs.oracle.com/javase/8/javafx/api/javafx/animation/Timeline.html)

También puedes consultar el siguiente ejemplo de uso de la clase Timeline:

**Reloj despertador con JavaFX**: [**https://www.youtube.com/watch?v=zJZ-ogqin2o**](https://www.youtube.com/watch?v=zJZ-ogqin2o)

* Método **@FXML private void reiniciarJuego(ActionEvent event)** : este método será **llamado cada vez que el usuario pulse en la interfaz el botón Comenzar nuevo juego**. Se encargará de finalizar la partida actual (detener el contador de tiempo, la música, volver a hacer visibles las casillas de las parejas encontradas en la partida anterior, etc.)  
    
  Por último, para ahorrar código y no volver a redefinir aspectos ya definidos, llamará al método **initialize** visto anteriormente que terminará de configurar los aspectos de la nueva partida que comenzará a continuación.  
    
    
  
* Método **@FXML private void mostrarContenidoCasilla(ActionEvent event)** : este método será de realizar una de las acciones principales de este juego, la cual consiste en desvelar la imagen que se esconde tras la carta pulsada. Debe implementar la lógica necesaria para saber si se trata la **primera carta descubierta en el intento actual** (no hay posibilidad de pareja aún) o se trata ya de la **segunda carta descubierta en el intento actual** (en ese caso deberá comprobarse si se ha encontrado una pareja o no).  
    
  Asimismo, deberá gestionar correctamente otros aspectos como el **conteo de intentos**, el **efecto de sonido** a reproducir en caso de pareja correcta o incorrecta o hacer la llamada a la **finalización del intento** cuando haya pasado el tiempo necesario (el desarrollo de este método se explica a continuación).

**Una pista: Considera que descubrir una carta consiste en asignarle la imagen que debería tener, mientras que ocultarla consistiría en "anular" esa imagen**

**IMPORTANTE:** se recomienda implementar un **método para finalizar el intento** (descrito a continuación) ya que no se deberían ocultar las cartas descubiertas en este mismo método (de hacerlo aquí, la segunda carta se ocultaría nada más ser descubierta, por lo que no daría tiempo a ver su contenido). En su lugar, es mejor utilizar un **Timeline** similar al utilizado en el contador de tiempo de forma que una vez se haya descubierto la segunda carta, **se disponga de, al menos, 1.5 segundos para visualizar el intento realizado**.

* Método **private void finalizarIntento()** : este método será el responsable de finalizar el intento actual. Este método implementará la lógica necesaria para **hacer desaparecer las cartas elegidas** (si se ha descubierto una pareja) o, si no ha sido así, **volver a ocultar su contenido** manteniendo ambas cartas en el tablero.  
    
  Además, verificará si el juego ha finalizado (se ha descubierto la última pareja disponible). En ese caso, se detendrá el contador de tiempo, se mostrará una **pantalla para indicar que el juego ha finalizado** y sonará una música o efecto de sonido para indicar la **victoria del jugador**.
* Método **@FXML private void salir(ActionEvent event)** : este método será cuando el usuario pulse en la interfaz el botón **Salir del juego**. Debe mostrar una ventana de aviso para preguntar al usuario **si desea salir de la aplicación o no**. Si el usuario confirma la acción, la aplicación se cerrará. En caso contrario, la aplicación continuará normalmente.

**Ejercicio Resuelto**

Principio del formulario

Tras completar el desarrollo de la aplicación este podría ser su aspecto final:

*Memory Game! Aplicación terminada. Fran Jiménez (elaboración propia).* ([CC BY](http://creativecommons.org/licenses/?lang=es))

**Nota:** en la aplicación que se muestra en el vídeo se ha implementado la posibilidad de disponer de **3 tipos de juegos distintos** (escudos de fútbol, personas y plantas) que se elige aleatoriamente al comenzar la partida. Aunque esa variabilidad hace más atractivo el juego, **no es una característica que deba desarrollarse obligatoriamente**.

Final del formulario

**2.- Información de interés**

**Recursos necesarios y recomendaciones**

El proyecto **MemoriaConsola** que se entrega consiste en un sencillo juego de memoria en el que se crea un tablero de 16 posiciones y se eligen aleatoriamente 8 valores que, por parejas, serán introducidos también de forma aleatoria en el tablero. El jugador deberá ir descubriendo cada una de las posiciones tratando de localizar las parejas que hay en el tablero.

Tu tarea consiste en **adaptar el juego**para que se pueda jugar a través de una **interfaz gráfica** e incorpore elementos multimedia (imágenes, sonidos, fondos, etc) que lo hagan mucho más atractivo.

**Recursos**

* Ordenador personal.
* Entorno de desarrollo NetBeans.
* JDK y JRE de Java SE.
* SDK de JavaFX:<https://gluonhq.com/products/javafx/>
* Aplicación SceneBuilder para el diseño de la interfaz: <https://gluonhq.com/products/scene-builder/>

**Proyecto base y otros recursos necesarios para realizar la tarea**

* Proyecto de juego "**MemoriaConsola**" **(para ver el funcionamiento del juego a través de consola, NO hay que desarrollar nada aquí)**:
  + [Descargar el proyecto **MemoriaConsola.zip**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/MemoriaConsola.zip)
* Para la realización de la tarea **debes descargar y trabajar con el siguiente proyecto >>** [**PROG09\_Tarea2023\_24\_Alumnado.zip**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/PROG09_Tarea2023_24_Alumnado.zip)
* Recursos que puedes utilizar para el desarrollo del juego (también puedes buscar y utilizar recursos propios):
  + Imágenes para las cartas:   [**Juego de Escudos de fútbol**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/escudos.zip)     |     [**Juego de Personas**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/personas.zip)     |     [**Juego de Plantas**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/plantas.zip)
  + Recursos gráficos para la interfaz: [**Recursos gráficos para la Interfaz**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/interfaz.zip)
  + Música y efectos de audio: [**Sonidos**](https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/sonidos.zip)

**Recomendaciones**

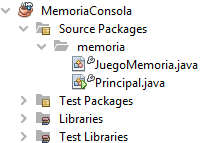
* Realiza en primer lugar un diseño de la interfaz de la aplicación.
* Decide qué elementos gráficos tendrá y el cometido de cada elemento, teniendo en cuenta los que se indican en el enunciado.
* Define claramente qué funcionalidad tendrá cada opción o elemento disponible en la interfaz.
* Haz una interfaz amena para el usuario y fácil de usar (puedes guiarte por la propuesta que te realizamos o aplicarle un "toque personal").
* Realiza la codificación de las distintas funcionalidades del juego.
* Revisa que toda la funcionalidad básica sea accesible a través de la interfaz.
* Prueba la aplicación y asegúrate de que el funcionamiento de la misma es adecuado.

**Para saber más**

Una vez realizada la tarea, el envío se realizará a través de la plataforma. Comprime la carpeta del proyecto NetBeans en un fichero .zip y nómbralo siguiendo las siguientes pautas:

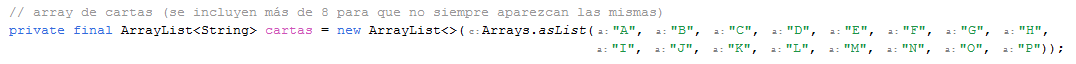
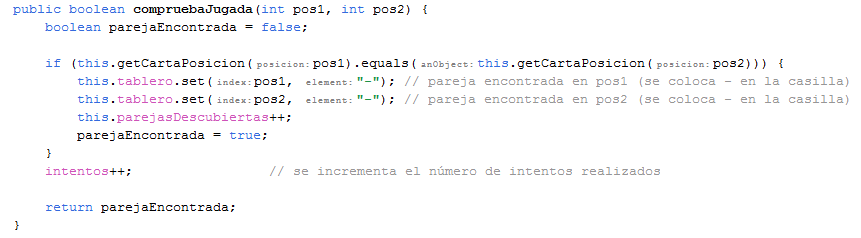
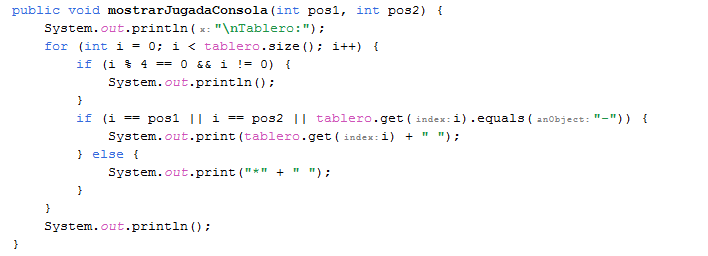
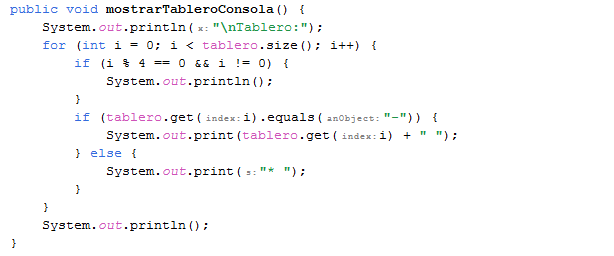
**Apellido1\_Apellido2\_Nombre\_PROG\_Tarea09**

**Anexo I - Explicación del proyecto MemoriaConsola**

El proyecto **MemoriaConsola** implementa de forma muy básica el clásico juego de encontrar parejas de cartas iguales en un tablero 4x4.

La **entrada** de información se realiza **desde el teclado eligiendo** en cada turno dos cartas indicando su posición (1-16) en el tablero de juego. Una vez elegida la posición de dos cartas distintas, se comparará su valor y si contienen el mismo se marcarán en el tablero como parejas identificadas (utilizando el símbolo "-"). La elección de cartas continuará hasta que se hayan encontrado todas las parejas del tablero, momento en el que finalizará el juego.

La estructura del proyecto contiene únicamente dos archivos:

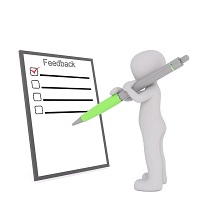
1. **JuegoMemoria.java** : Contiene la lógica necesaria para llevar el control de la partida. Consta de varios atributos y métodos:
   * **Atributos**:
     + **cartas (ArrayList<String>):** Contiene una lista de cartas de las que se elegirán 8 parejas que participarán en cada partida. Es interesante que esta lista de cartas sea superior al número de parejas elegidas para que no aparezcan siempre los mismos valores.  
       
     + **tablero** (**ArrayList<String>**): Contiene la lista de cartas seleccionadas en esta partida.
     + **intentos** (**int**): Almacena el número de intentos realizados en la partida actual.
     + **parejasDescubiertas** (**int**): Almacena el número de parejas descubiertas hasta el momento en la partida actual.
     + **parejasTotal** (**int**): Almacena el número de parejas en total.
   * **Métodos de la clase:**
     + **public void iniciarJuego():** Permite crear una nueva partida, definiendo un nuevo tablero de 16 posiciones. Se utilizarán 8 cartas aleatorias elegidas desde el ArrayList **cartas**. Las 8 cartas **se introducirán por duplicado** (8 parejas, 16 cartas en total) en el ArrayList **tablero** y se barajarán con el método **shuffle** para que aparezcan desordenadas.  
         
       https://www.juntadeandalucia.es/educacion/gestionafp/datos/tareas/DAW/PRO_8712926/2023-24/DAW_PRO_9_2023-24_Individual__859347/imagen.png
     + **public boolean compruebaJugada(int pos1, int pos2)**: Compara si los valores las cartas situadas en las posiciones pos1 y pos2 coinciden o no. Si coinciden se actualizará el tablero para indicar que las posiciones han sido descubiertas y se devolverá **true**, se devolverá **false** en caso contrario.  
         
       
     + **public String getCartaPosicion(int posicion):** Permite obtener el valor de la carta situada en la posición indicada.
     + **public boolean compruebaFin():** Permite saber si el juego ha finalizado (si el número de parejas descubiertas coincide con el número de parejas total) o no.  
         
       compruebaFin
     + **public int getTamanoTablero():** Permite obtener la cantidad de posiciones que tiene el tablero de juego.
     + **public void mostrarJugadaConsola(int pos1, int pos2):** Imprime a través de la consola el tablero mostrando en las posiciones recibidas como parámetro el valor de la carta que hay en esa posición. Para el resto de posiciones se mostrará el carácter  (\*) si la carta no ha sido descubierta o (-) si la carta ya ha sido descubierta (pareja encontrada). Este método **NO se utilizará en la implementación gráfica** del juego ya que la salida debe realizarse a través de un entorno gráfico.  
         
       
     + **public void mostrarTableroConsola():** Imprime a través de la consola la situación actual del tablero. Se mostrará el carácter  (\*) si la carta de cada posición no ha sido descubierta o (-) si la carta de cada posición ya ha sido descubierta (pareja encontrada). Este método **NO se utilizará en la implementación gráfica** del juego ya que la salida debe realizarse a través de un entorno gráfico.  
         
       
2. **Principal.java** : Contiene el programa main que implementa la lógica de juego para una interfaz en consola.
   * Creación de una instancia de la clase **JuegoMemoria** que almacenará la lógica de la partida.
   * Se utiliza un bucle **while** que continuará repitiéndose mientras la partida no finalice.
     + Se comienza mostrando la situación actual del tablero.
     + Por teclado, se pide al usuario la posición de la primera casilla a descubrir (se utiliza control de las excepciones **InputMismatchException** e **IndexOutOfBoundsException**) y se muestra la jugada realizada (se indica dos veces la misma posición ya que el método **mostrarJugadaConsola** necesita dos parámetros).
     + Por teclado, se pide al usuario la posición de la segunda casilla a descubrir (se utiliza control de las excepciones **InputMismatchException** e **IndexOutOfBoundsException**) y se muestra la jugada realizada (ahora sí se indican las dos posiciones introducidas por el usuario).
     + Se comprueba que las posiciones no sean la misma
     + Se realiza una llamada al método **compruebaJugada** para saber si se ha encontrado una pareja o no.
   * Al descubrir todas las parejas el juego finaliza con un mensaje.

En un entorno gráfico el fichero **Principal.java** **NO será necesario**, deberás implementar la lógica de juego desde el **Controller** de JavaFX.

Puedes descargar el proyecto **MemoriaConsola** desde el apartado 2.- Información de interés.

**3.- Evaluación de la tarea**

**Criterios de evaluación implicados**

[Peggy\_Marco](https://pixabay.com/es/illustrations/fax-hombre-blanco-modelo-3d-aislado-1889009/) ([Pixabay License](http://creativecommons.org/publicdomain/zero/1.0/deed.es" \o "Pixabay License" \t "_blank))

Del RA 5: Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.

* f) Se han utilizado las herramientas del entorno de desarrollo para crear interfaces gráficos de usuario simples.
* g) Se han programado controladores de eventos.
* h) Se han escrito programas que utilicen interfaces gráficos para la entrada y salida de información.

**¿Cómo valoramos y puntuamos tu tarea?**

Aquí tienes algunas cuestiones generales que podrán penalizarte en la evaluación de los ejercicios de la tarea. Tenlos muy en cuenta al resolver los ejercicios y comprobar su funcionamiento:

* **La aplicación debe compilar.** Cualquier funcionalidad pedida en el enunciado debe poderse probar y ha de funcionar correctamente de acuerdo a las especificaciones del enunciado. **Si la aplicación contiene errores de sintaxis y ni siquiera compila, el ejercicio podría considerarse nulo y su puntuación podría llegar a ser 0.**
* **La ejecución de la aplicación debe ser adecuada sin "romperse" en ningún caso**. Si eso sucede es porque la implementación realizada no ha sido correcta de acuerdo a los requisitos indicados en el enunciado.
* **Se utilizan estructuras de salto incondicional para el control del flujo**, o bien herramientas como **return**, **System.exit()** o cualquier otro mecanismo que no sea el fin de la ejecución natural del programa.
* **Se declaran identificadores que** **no cumplen con el convenio sobre "asignación de nombres a identificadores"** establecido para el lenguaje Java para constantes, variables, métodos, clases, etc.
* **Se declaran identificadores que no tienen nombres significativos o descriptivos** que representen de alguna manera la información que están almacenando para que el código quede lo más claro, legible y autodocumentado posible.
* **No se observa una corrección ortográfica y gramatical,** así como la coherencia en las expresiones lingüísticas, tanto en los comentarios en el código como en los textos de los mensajes que aparezcan en pantalla para pedir información de entrada al usuario, mostrar resultados de salida o indicar situaciones de error. Deben evitarse mensajes inapropiados y/o erróneos, descontextualizados o insuficientes.
* **El código no está correctamente indentado**. Es fundamental para poder observar e intuir rápidamente, y de forma visual, la estructura de los programas. Recuerda que NetBeans puede hacer esto por ti (selecciona el código a "formatear" o "embellecer" y **pulsa Alt+Mayús.+F**).

En la siguiente tabla puedes observar con detalle los elementos de rúbrica o puntos de control que se van a evaluar en esta tarea. Para cada elemento se indican los criterios de evaluación asociados así como el peso que tendrá en la calificación total de la tarea y los aspectos que se tendrán en cuenta para la corrección. Esta calificación total se calculará mediante la corrección de cada uno de esos elementos de rúbrica y se normalizará sobre una escala de 0 a 10 teniendo en cuenta la puntuación o peso de cada elemento:

|  |  |
| --- | --- |
| **Rúbrica de la tarea** | |
| **Punto de Control 1: Se ha trabajado con Java FX Scene Builder para crear la interfaz solicitada en el enunciado, y dicha interfaz es coherente (similar a la mostrada en los ejemplos o personalizada a gusto del alumno/a).**  RA5 CEf: Se han utilizado las herramientas del entorno de desarrollo para crear interfaces gráficos de usuario simples. | **16,66%** |
| **Punto de Control 2: Se ha creado la clase controladora de eventos asociados a la interfaz, generando en cada caso la llamada a los métodos oportunos para controlar los eventos generados por la interfaz presentando la funcionalidad deseada, vinculando de esta forma ambas partes del código.**  RA5 CEg: Se han programado controladores de eventos. | **16,67%** |
| **Punto de Control 3: Se han utilizado interfaces gráficos para gestionar la entrada/salida de información (utilización del juego y de sus opciones) obteniendo los resultados esperados en cada caso.**    RA5 CEh: Se han escrito programas que utilicen interfaces gráficos para la entrada y salida de información. | **16,67%** |
| **Punto de Control 4: Los ejercicios desarrollados funcionan correctamente de acuerdo a las especificaciones y limitaciones requeridas en la tarea.**    RA5 CEf: Se han utilizado las herramientas del entorno de desarrollo para crear interfaces gráficos de usuario simples.  RA5 CEg: Se han programado controladores de eventos.  RA5 CEh: Se han escrito programas que utilicen interfaces gráficos para la entrada y salida de información. | **50%** |