



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Programación Orientada a Objetos

**Segundo Proyecto: Centro de Juegos**

Johan Rodriguez, Karina Urbina, y Maria Solis

Profesor

Jose Leonardo Viquez Acuña

Sede San Carlos

II Semestre 2022

## Segundo Proyecto: Centro de Juegos

### Índice

<b>Introducción</b>	<b>3</b>
<b>Análisis del problema</b>	<b>3</b>
Registro y verificación de los jugadores . . . . .	3
Menú Central de Juegos . . . . .	3
Menú Individual del Juego . . . . .	3
Desarrollo de los juegos del equipo . . . . .	4
Implementación del tercer juego . . . . .	5
Manejo de archivos y excepciones . . . . .	5
<b>Solución del problema</b>	<b>5</b>
Etapa 1: Registro de usuarios . . . . .	5
Etapa 2: Implementación del centro de juegos . . . . .	6
Etapa 3: Menús para la visualización de estadísticas . . . . .	7
Etapa 4: Desarrollo de los juegos del equipo . . . . .	10
Primer juego: . . . . .	10
Segundo juego: . . . . .	11
Etapa 5: Implementación del tercer juego . . . . .	12
Etapa 6: Manejo de archivos y excepciones . . . . .	12
<b>Análisis de resultados</b>	<b>13</b>
<b>Conclusiones</b>	<b>14</b>
<b>Recomendaciones</b>	<b>14</b>
<b>Referencias Bibliográficas</b>	<b>15</b>

## **Introducción**

En este reporte, se busca desarrollar un centro de juegos y sus respectivos juegos utilizando conceptos previamente definidos por el instructor, implementando el lenguaje de programación Java y conceptos previamente analizados en el curso, de esta forma se logra crear una plataforma más estandarizada, permitiendo la interacción con juegos desarrollados por otros grupos de trabajo. Por otro lado, el programa en su totalidad utiliza una interfaz gráfica, de este modo, se logra una interacción más amena entre el usuario y el programa.

A continuación, en el reporte se detallarán los conceptos aplicados para el desarrollo del centro de juegos y sus demás componentes, y el porqué de su aplicación, así como otros detalles referentes a la implementación y generación de juegos para el sistema.

## **Análisis del problema**

Según el problema planteado, se requieren resolver los siguientes puntos:

### **Registro y verificación de los jugadores**

Los usuarios, podrán registrarse en el sistema proporcionando su nombre de usuario y su contraseña, estos datos serán almacenados en una estructura tipo `ArrayList`, a la cual se podrá acceder durante el inicio de sesión cuando el usuario desee entrar a la central de juegos para así validar el usuario y la contraseña.

### **Menú Central de Juegos**

Muestra en pantalla las diferentes opciones de juegos disponibles, así los jugadores pueden seleccionar su juego de preferencia pulsando sobre el botón respectivo.

### **Menú Individual del Juego**

En pantalla se muestra al usuario, los datos necesarios para inicializar el juego escogido, por ejemplo, el signo con el que se desea participar en una partida, etc. También mediante botones, se le brinda al usuario las opciones de ver sus estadísticas personales

y las estadísticas generales.

Las estadísticas personales se almacenan en una lista en la que se registra el puntaje obtenido en cada partida del jugador, además de la fecha y hora de inicio, fecha y hora de finalización, y tiempo que tomó la partida. En la parte superior de la ventana se mostrará también la cantidad de tiempo de juego diario, la cantidad de horas y la cantidad de partidas efectuadas.

Por su parte las estadísticas generales, le muestran al jugador cuales son las diez mejores partidas esto basándose en una comparación de los puntajes obtenidos por todos los jugadores. Se imprime en pantalla el nombre del jugador además de la puntuación obtenida y la fecha y hora en la que se realizó.

Se muestran también los botones respectivos para iniciar la partida o cerrar la ventana, haciendo que el usuario regrese al menú central de juegos.

### **Desarrollo de los juegos del equipo**

Se analiza en primera instancia desarrollar los siguientes juegos:

- Tic Tac Toe: El juego de gato, tradicional, aunque la partida se desarrolla de manera individual entre el usuario y el computador, en el menú individual, se le brinda al usuario la opción de escoger el símbolo con el que desea efectuar la partida, y inicia esta, ganando al completar una línea de 3 espacios con su símbolo de manera vertical, horizontal y diagonalmente
- Memoria: Muestra al usuario en pantalla dieciséis tarjetas volteadas , permitiéndole al usuario escoger dos cartas y comprobar si estas tienen la misma imagen en su reverso, si esto es correcto, suma 20 puntos al marcador del juego, de lo contrario, si las cartas no coinciden , se voltean nuevamente y resta 10 puntos al puntaje del jugador. En este juego también se pretende implementar el patrón de diseño

Singleton.

Ambos juegos permiten al usuario finalizar la partida pulsando el botón de salida, almacenando en el registro la información de la partida y señalando que el estado de esta fue inconcluso, también el juego de memoria permite al usuario seleccionar la opción de reiniciar la partida.

### **Implementación del tercer juego**

Se requiere implementar en el sistema el juego `HiddenNumberGame`, creado por Aaron Gonzáles, David Arguedas y Leiner Alvarado, esto mediante la incorporación del archivo `.class` de su juego.

### **Manejo de archivos y excepciones**

Se requiere implementar excepciones para evitar la caída de la aplicación en las secciones vulnerables, además se deben guardar en archivos los registros de usuario y puntajes, los cuales, volverán a cargarse cada vez que se ejecute el programa.

### **Solución del problema**

Antes de comenzar a dar solución a los puntos planteados en el análisis del problema, se inició por implementar las interfaces `iRegistro`, `iJugador`, `iJuego` y `iCentroJuego`, estas fueron facilitadas por el profesor Leonardo Villalobos. Con estas interfaces se pudo acceder a diferentes funciones como extraer el nombre del jugador, registrar la puntuación, devolver el registro histórico de partidas, asignar la fecha y hora de inicio y finalización de las partidas, etc. Seguidamente se implementaron y resolvieron las siguientes etapas:

#### **Etapas 1: Registro de usuarios**

En la primera etapa se le dará solución a la creación de la estructura para los registros de personas, para esto se crea la clase `Register.java`, la cual cuando el usuario lo solicite, le muestra una nueva ventana, en la cual se le pide al usuario ingresar dos datos, su nombre de usuario y su contraseña, estos son ingresados dentro de los

`TextField` los cuales mediante el método `getText()` capturan los datos ingresados por teclado por el jugador, además de esto le pide al usuario que confirme su contraseña, y al capturar esta, mediante condicionales comprueba que esta confirmación sea igual a la primera contraseña ingresada, si estas no concuerdan, muestra un mensaje de error en la pantalla, de lo contrario, almacena dentro de otra clase llamada `Login`, en una estructura `HashMap`, el nuevo registro y le confirma al usuario que este se a concretado de manera correcta.

Para la validación de usuarios, dentro de la clase `Login`, en el método `actionPerformed()`, se captura el usuario y la contraseña proporcionadas en la ventana de inicio para ingresar a la central de juegos. El método comprueba mediante condicionales `if` que el usuario y la contraseña ingresadas en los `TextField` sea diferentes al nombre de usuario y contraseña de los registros almacenados en el `HashMap` de `users`, si la condicional se cumple, muestra un mensaje en pantalla indicando que las credenciales del usuario son incorrectas. De manera contraria devuelve un `true` en la variable booleana `loginSuccess`, el método `getter` de esta variable es llamado en la clase `Main.java` como confirmación de validación de credenciales e inicialización del juego.

## **Etapas 2: Implementación del centro de juegos**

La creación del Menú Central de Juegos será el problema solucionado en la etapa dos. Dentro de la clase `GameCenter.java`, se define una nueva ventana, la cual implementa las interfaces `iCentroJuego`, y `ActionListener`. Esta clase contiene además tres atributos definidos, el primero es un `ArrayList` de la interfaz `iJugador` y contiene los juegos disponibles. El siguiente es otro `ArrayList` de la interfaz `iRegistro` y contiene los records de partidas registradas, finalmente, el atributo `player` referenciado de la interfaz `iJugador`.

En esta clase se agregan al `ArrayList` de `games` las clases correspondientes a los dos juegos creados (`Tic Tac Toe` y `Memory`). Además del saludo mostrado en la ven-

tana, se muestra el nombre de este gracias a el método `getNombre` del atributo `player`. Se agregan la cantidad de botones correspondientes a cada opción de juego que se encuentre dentro de `games`, esto mediante un ciclo `for` que verifica que, para cada elemento dentro del largo de la lista, se agregue un botón y mediante el método `getNombre`, escribe el nombre del juego en cada uno de los botones creados. Además de esto, crea un botón para que al ser seleccionado muestra el podio de la plataforma.

Gracias a los métodos `getToolTipText()` y `setToolTipText()` le permite al usuario obtener una pequeña descripción del juego que desea seleccionar cuando se posiciona encima del botón correspondiente a este.

Mediante el método `actionPerformed`, y el getter `getActionCommand()` comprueba la selección del botón que haya presionado el usuario, y basándose en esta, muestra en pantalla el menú de inicio del juego seleccionado.

### **Etapas 3: Menús para la visualización de estadísticas**

En cuanto a los menús individuales para cada juego:

- Tic Tac Toe: Dentro del método `iniciarPartida` de la clase `TicTacToe.java`, se define una nueva ventana, la cual lleva por título el nombre del juego, además muestra los botones de:

- `PersonalStatus`: contiene un `HashMap` definido como `playedDays` el cual recibe datos tipo `LocalDate` e `Integer`. Con un ciclo `for`, recorre los registros de juegos en el centro de juegos y mediante el método `getRegistros`, obtiene los, registros de las partidas Tic Tac Toe, luego, de igual manera con base al registro busca los registros cuyo nombre de jugador sea igual al del usuario que está solicitando el listado, esto mediante los métodos `getJugador` y `getNombre` de la interfaz `register` y el método `getNombre` de la interfaz `jugador`, e imprime los datos de inicio y finalización de la partida, con los métodos de la interfaz `register` llamados `getInicio()` y `getFinalizacion()`, también muestra el puntaje

de la partida, la duración de esta y el estado de la misma, por medio de `getPuntaje()` y `getSegundosTotalesPartida()`.

Para registrar los minutos de juego por día en promedio, se basa en la fecha de inicio de los registros y las almacena en `playerDays`, luego hace un calculo de la cantidad de tiempo jugado por día acorde con lo registrado anteriormente, dividiendo las horas entre 3600, los minutos equivalen al modular del total de horas, dividido entre 60, y los segundos al modular del total de horas dividido entre 60; esto se imprime en un `label` colocado en la parte superior del programa en el formato horas, minutos y segundos.

La ventana de `PersonalStatus` permite al usuario abandonar la ventana pulsando sobre el botón inferior que dice `Back`, el cual contiene la función `dispose()` la cual cierra la ventana actual.

– El botón de `GeneralStatus`, permite al usuario visualizar en pantalla las 10 primeras partidas con el mejor puntaje, esto lo hace gracias a una lista referenciada de la interfaz registro, llamada `registers`, la cual corresponde al registro que se almacene en la central del juego correspondiente al juego del `TicTacToe`.

Se hace uso de la interfaz `Comparator` de la implementación de la clase `Util`, para aplicar un ordenamiento mediante el método “`sort`” de los registros, comparando dos registros, y obteniendo el mayor de estos.

Luego mediante un ciclo `for` comprueba que la variable `i` sea menor a 10, e imprime cada registro en pantalla hasta que la condición deje de cumplirse.

De igual forma existe un botón de salida de esta ventana permitiendo al usuario regresar al menú individual del juego.

– En este juego también se le muestra al usuario un mensaje en la ventana que lo incita a escoger el símbolo con el que desea jugar la nueva partida, estos están colocados



sobre botones y corresponden a las letras X y O.

Si el usuario selecciona la X, significa que es el turno del jugador 1 por lo tanto este iniciará la partida

Si el usuario escoge la letra O el primer turno será el del jugador 2, por lo tanto, deberá esperar que la computadora finalice su turno.

– El botón de `Start` en la parte inferior de la pantalla cierra el menú individual del juego e inicializa la partida mediante el método `game()` de la clase `TicTacToe`.

– El botón de `Exit` devuelve al jugador al menú central de juegos

- **Memory Game:** En esta ventana encuentran los botones de `General Stats`, `Personal Stats`, `Start` y `Exit`.

– Tal como se muestra en fig. 1 mostrar las estadísticas personales, se creó un evento que cuando el usuario presione el botón `General Stats` se genere la ventana con los datos de las primeras 10 partidas ordenadas que tengan el mayor puntaje del registro de todos los usuarios. Esto ocurre de la misma manera que el recorrido que se utiliza en el primer juego solo que este caso la lista de registros está referenciada al juego de Memoria.

– En el caso de las estadísticas personales, si el usuario presiona `Personal Stats` se llamará al evento que crea la ventana donde se muestra el historial de partidas que ha realizado el usuario dentro del juego. Para obtener estos datos se recorre el registro del centro de juegos, se compara el usuario actual con los demás usuarios y se almacena la hora de inicio y finalización con los puntos obtenidos de cada partida, si fue finalizada con éxito. Además, se guarda el promedio de tiempo diario en el juego, que se obtiene dividiendo la cantidad de días por la cantidad de horas y también se guarda la cantidad de partidas realizadas. Exactamente igual al botón con la misma función del primer juego

pero su recorrido se basa en los registros del juego memoria, en la fig. 2 y fig. 2 se muestra el código utilizado.

– Los botones de `Start` y `Exit` cumplen las mismas funciones de inicio de partida y cierre de ventana actual que en el primer juego.

#### **Etapla 4: Desarrollo de los juegos del equipo**

##### ***Primer juego:***

Para el desarrollo de este juego no se tomó como referencia ningún ejemplo, se creó con conocimientos previamente adquiridos por lo integrantes del grupo de trabajo, el juego consiste en una implementación básica del famoso juego TicTacToe, en este caso si el jugador escogió la X antes de iniciar el juego este iniciará primero, en el caso contrario primero será el turno de la CPU, a continuación se detallan los pasos realizados para el desarrollo de este juego.

**Paso 1.** En el primero paso se planteó la interfaz de usuario, en este caso se optó por una interfaz simple, con botones grandes y poco contenido, para que no exista confusión mientras el usuario juega la partida, cada botón al presionarse muestra con una fuente de tamaño grande el símbolo previamente escogido por el jugador.

**Paso 2.** Seguidamente, se implementan los métodos definidos en la interfaz `iJuego`, de esta forma es posible tener acceso a los datos necesarios para identificar el juego e iniciar y finalizar las partidas. De este modo, la partida es iniciada con el método `iniciarPartida`.

**Paso 3.** Por otro lado, se implementa el método `computerTurn`, donde se utiliza un algoritmo que genera números random, de esta forma el CPU es capaz de posicionarse sobre los botones que se encuentran sin presionar.

**Paso 4.** Por último, existe el método `calcWin()` el cual se encarga de verificar que alguno de los dos jugadores haya ganado, si este método detecta que uno de los jugadores ganó, o existe un empate (todos los botones fueron presionados) entonces

el juego termina y se llama al método `terminarPartida`, para que registre el tiempo de finalización y destruya el frame de juego, seguidamente se añade el registro de juego al centro de juegos, utilizando el método proporcionado por la interfaz `iCentroJuego`.

### ***Segundo juego:***

Para la elaboración del segundo juego se tomó como referencia parte del código creado por (Galindo, s. f.) y (Chinchilla, s. f.) donde cada uno desarrollan un juego de memoria similar. Tomando como base estos ejemplos, se procedió a fusionar partes de ambos códigos para obtener un `Memory Game` adaptado a los requerimientos del proyecto.

Se crearon dos archivos llamados `jMemory` y `Memory`, donde el primero cuenta con la estructura y el funcionamiento del juego que se le va a mostrar al usuario y el segundo, incorpora la estructura del menú principal del juego y lo conecta con el centro de juegos.

A continuación, se detalla un poco más el proceso de desarrollo:

**Paso 1.** En el primer paso para el desarrollo de este juego se inició por personalizar la interfaz gráfica de la base del juego, cambiando la dimensión y las imágenes de las cartas y agregando otros elementos a la ventana de la partida.

**Paso 2.** Seguidamente, se procedió a implementar las interfaces brindadas por el profesor a la base del `Memory Game` y se incorporó el patrón de diseño `Singleton` para que solo exista una única instancia del juego y así evitar que se creen repetidas. En la fig. 4 se observa el método público creado para acceder al constructor privado de la interfaz gráfica del juego y así lograr crear la única instancia que se va a utilizar.

**Paso 3.** Se incluye al juego la ventana principal del mismo, cuya descripción de elaboración se encuentra en definida en la Etapa 3 de la solución del problema del presente proyecto.

**Paso 4.** Se incorporó el método de `terminarPartida`, donde se guarda la fecha y hora en la que se finaliza la partida, se elimina la instancia y se verifica si la partida fue finalizada completamente o se terminó antes de completar el juego. A partir de esto

se guarda el puntaje y se registra en el historial del jugador. Además, se almacena la información en el registro del centro de juegos, ver fig. 5.

**Paso 5.** Para este último paso se implementa la clase `JMemory` cual contiene métodos como el `setCards()` que declara los 16 botones correspondientes a las 16 cartas del juego, las cuales inicialmente hacen uso del método `setDisabledIcon()` para ocultar la imagen de su reverso. Luego, contiene el método `btnEnable()` el cual se encarga de saber si una carta se ha volteado.

Al presionar la primera carta, mediante una condicional, se visualiza el icono que esta tiene, si al presionar la segunda carta, el icono es igual, se suman 20 puntos al marcador, de lo contrario si mediante el método `compare()` se demuestra que el icono de las cartas es diferente, esto gracias al método `getDescription()`, se le suma 10 puntos.

Los mejores puntajes están calculados en base a los jugadores que hayan obtenido la menor cantidad de puntos, ya que entre más intentos fallidos tengan, el puntaje será mucho mayor.

### **Etapas 5: Implementación del tercer juego**

Para la implementación del tercer juego se debía utilizar uno de los juegos elaborados por los otros grupos de trabajo, en esta ocasión se optó por el juego `HiddenNumberGame`, elaborado por los compañeros Leiner Alvarado, Aaron Gonzalez y David Arguedas. En este caso se debe de implementar el archivo `.class` correspondiente al juego elaborado por los compañeros, por lo tanto, se incluyó en el paquete de `juego3`, posterior a esto mediante las interfaces de las cuales disponen los juegos, se logra iniciar la partida y recuperar el nombre y descripción correspondientes al juego, de esta forma dando por finalizada la implementación del tercer juego.

### **Etapas 6: Manejo de archivos y excepciones**

En cuanto al manejo de archivos, se implementaron archivos con extensión `.txt`, donde se almacenan los usuarios y los registros correspondientes al centro de juegos,

en este caso los usuarios son almacenados en el archivos `users.txt`, que se crea en el directorio desde el cual se ejecuta el programa, y posteriormente el archivo `records.txt`, el cual almacena los registros historicos de partidas jugadas por los usuarios.

Por otra parte, el manejo de excepciones se implementó de varias maneras, se usaron condicionales para verificar datos y opciones, también la estructura `try catch` para manejar excepciones correspondientes a entradas erroneas y excepciones similares, en general el desarrollo del programa emplea manejo de excepciones para que el usuario pueda disponer de la mayor cantidad de diversión sin caídas o errores del programa. Ejemplos del manejo de excepciones se pueden visualizar en fig. 6, fig. 7, fig. 8 y fig. 9.

<b>Análisis de resultados</b>		
Tarea/Requirimiento	Estado	Observaciones
Registro y verificación de los jugadores	Completado	Se logra registrar y verificar correctamente.
Menú central de juegos	Completado	Implementado correctamente.
Menú principal de cada juego	Completado	Su ejecución es correcta tal y como se planteó.
Desarrollo de los juegos del equipo	Completado	Sin observaciones, todo completo.
Implementación del tercer juego	Completado	El tercer juego fue escogido, implementado y ejecutado de la manera correcta.

Tarea/Requerimiento	Estado	Observaciones
Manejo de archivos y excepciones	Completado	Manejo de excepciones y archivos completo, permitiendo mediante las excepciones evitar la caída del sistema, y recuperando los registros de jugadores gracias a los archivos.

### Conclusiones

Al concluir el presente proyecto se ha logrado desarrollar el sistema propuesto por el profesor, el cual, a pesar de ser complejo por la implementación de diferentes interfaces y juegos, permitió a los integrantes del equipo comprender de mejor manera la funcionalidad de las interfaces, la implementación de GUI y el manejo de excepciones para llevar a cabo de manera exitosa el proyecto. Finalmente, las extensas pruebas y el tiempo que se dedicó al proyecto permitieron que el proceso de desarrollo se concluyera con éxito y dejara como resultado una plataforma de juegos donde los usuarios pueden divertirse y competir contra otros jugadores o incluso contra ellos mismos, para así mejorar su rendimiento, fomentando la productividad y resiliencia a través del juego.

### Recomendaciones

- Distribuir adecuadamente las tareas del proyecto.
- Administrar mejor el tiempo de dedicación al trabajo.
- El programa podría en el futuro implementar el sistema para permitir juegos con multijugadores.
- Definir de manera más clara los requerimientos del proyecto.

### **Referencias Bibliográficas**

Chinchilla, Cinthya. s. f. «Juego de Memoria (JAVA NetBeans)». [https://www.youtube.com/watch?v=JtT3\\_eo](https://www.youtube.com/watch?v=JtT3_eo) .

Galindo, Darwin. s. f. «Como hacer un juego simple en java & Netbeans (Encuentra par)». <https://www.youtube.com/watch?v=naWUqN0VoaQ>.

```

1  final JButton generalStatsButton = new JButton("General Stats");
2  generalStatsButton.addActionListener(new ActionListener() {
3      public void actionPerformed(ActionEvent e) {
4          JFrame statsFrame = new JFrame("General Stats");
5          statsFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
6          statsFrame.setLayout(new FlowLayout());
7          statsFrame.setResizable(false);
8          statsFrame.setSize(500, 550);
9
10         // Add a text area to show the stats
11         JTextArea stats = new JTextArea();
12         stats.setPreferredSize(new Dimension(450, 450));
13         stats.setEditable(false);
14
15         // Show only the 10 best scores of centroJuegos.getRegistros(Memory.this)
16         List<iRegistro> registers = centroJuegos.getRegistros(Memory.this);
17         registers.sort(new Comparator<iRegistro>() {
18             public int compare(iRegistro r1, iRegistro r2) {
19                 return r2.getPuntaje() - r1.getPuntaje();
20             }
21         });
22         for (int i = 0; i < 10; i++) {
23             if (i < registers.size()) {
24                 iRegistro register = registers.get(i);
25                 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");
26                 stats.append("Username: " + register.getJugador().getNombre() + "\n" +
27                     "Score: " + register.getPuntaje() + "\n" +
28                     "Date: " + register.getInicio().format(formatter) + "\n\n");
29             }
30         }
31
32         statsFrame.add(stats);
33
34         JButton exitButton = new JButton("Back");
35         exitButton.setPreferredSize(new Dimension(450, 50));
36         exitButton.setFont(new Font("Arial", Font.BOLD, 20));
37         exitButton.addActionListener(new ActionListener() {
38             public void actionPerformed(ActionEvent e) {
39                 statsFrame.dispose();
40             }
41         });
42         statsFrame.add(exitButton);
43
44         statsFrame.setVisible(true);
45
46     });

```

Figure 1

*Eventos del botón General Stats en el juego Memory Game*



```

1  final JButton statsButton = new JButton("Personal Stats");
2      statsButton.addActionListener(new ActionListener() {
3          public void actionPerformed(ActionEvent e) {
4
5              JFrame statsFrame = new JFrame("Personal Stats");
6              statsFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
7              statsFrame.setLayout(new FlowLayout());
8              statsFrame.setResizable(false);
9              statsFrame.setSize(500, 650);
10
11              // Add a text area to show the stats
12              JTextArea stats = new JTextArea();
13              stats.setPreferredSize(new Dimension(450, 450));
14              stats.setEditable(false);
15
16              HashMap<LocalDate, Integer> playedDays = new HashMap<LocalDate, Integer>();
17              int totalGames = 0;
18              int totalHours = 0;
19
20              for (iRegistro register: centroJuegos.getRegistros(Memory.this)) {
21                  if (register.getJugador().getNombre().equals(jugador.getNombre())) {
22                      DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");
23                      stats.append("Start time: " + register.getInicio().format(formatter) + "\n" +
24                          "End time: " + register.getFinalizacion().format(formatter) + "\n" +
25                          "Score: " + register.getPuntaje() + "\n" +
26                          "Total time: " + register.getSegundosTotalesPartida() + " seconds" + "\n" +
27                          "Finished: " + register.getEstadoFinalizado() + "\n\n");
28                      totalGames++;
29                      totalHours += register.getSegundosTotalesPartida();
30
31                      if (playedDays.containsKey(register.getInicio().toLocalDate())) {
32                          playedDays.put(register.getInicio().toLocalDate(), playedDays.get(register.getInicio().toLocalDate()) + 1);
33                      } else {
34                          playedDays.put(register.getInicio().toLocalDate(), 1);
35                      }
36                  }
37              }

```

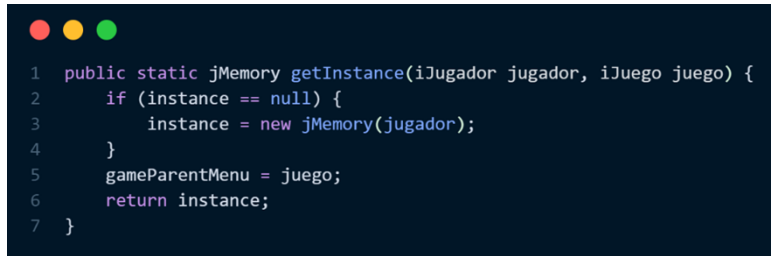
**Figure 2**

*Eventos del botón "Personal Stats"*

```
1 // Show the average time played per day according to the total HashMap
2 int totalDays = 0;
3 for (LocalDate date: playedDays.keySet()) {
4     totalDays++;
5 }
6 JLabel averageTime;
7 if (totalDays != 0) {
8     averageTime = new JLabel("Average time played per day: " + totalHours / totalDays + " seconds");
9 } else {
10     averageTime = new JLabel("Average time played per day: 0 seconds");
11 }
12 averageTime.setPreferredSize(new Dimension(450, 25));
13 statsFrame.add(averageTime);
14
15 JLabel totalGamesLabel = new JLabel("Total games: " + totalGames);
16 totalGamesLabel.setPreferredSize(new Dimension(450, 25));
17 statsFrame.add(totalGamesLabel);
```

**Figure 3**

*Código para calcular el promedio del tiempo en el juego y el total de partidas.*



```
1 public static jMemory getInstance(iJugador jugador, iJuego juego) {  
2     if (instance == null) {  
3         instance = new jMemory(jugador);  
4     }  
5     gameParentMenu = juego;  
6     return instance;  
7 }
```

**Figure 4**

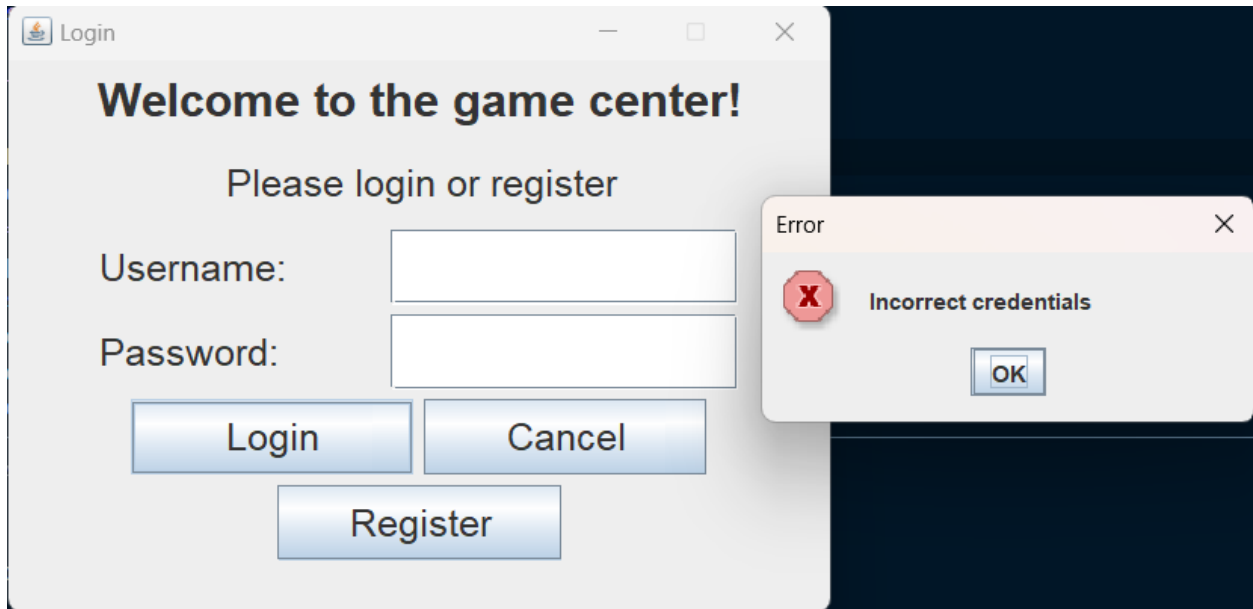
*Método que retorna una instancia de la clase*



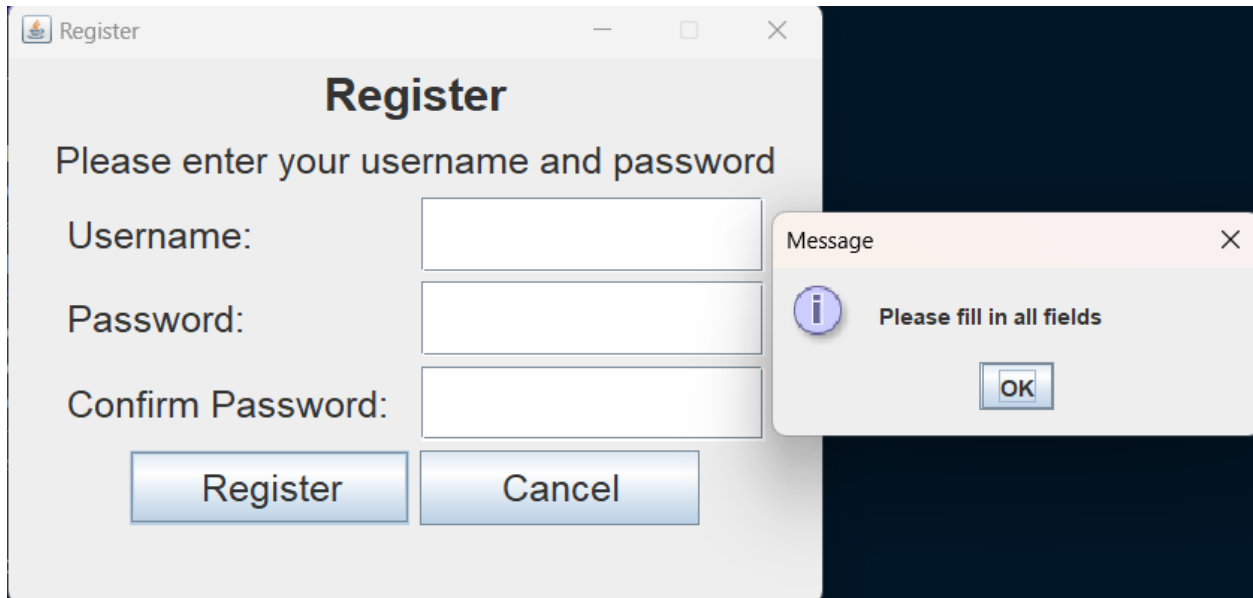
```
1 public void terminarPartida() {  
2     register.setFinalizacion(LocalDateTime.now());  
3  
4     instance.dispose();  
5  
6     if (instance.getIsFinished()) {  
7         int score = instance.getScore();  
8         register.setFinished(true);  
9         register.setScore(score);  
10        player.registrarPuntaje(score, this);  
11    } else {  
12        register.setFinished(false);  
13        register.setScore(0);  
14        player.registrarPuntaje(0, this);  
15    }  
16  
17    center.addRegistro(register);  
18 }
```

**Figure 5**

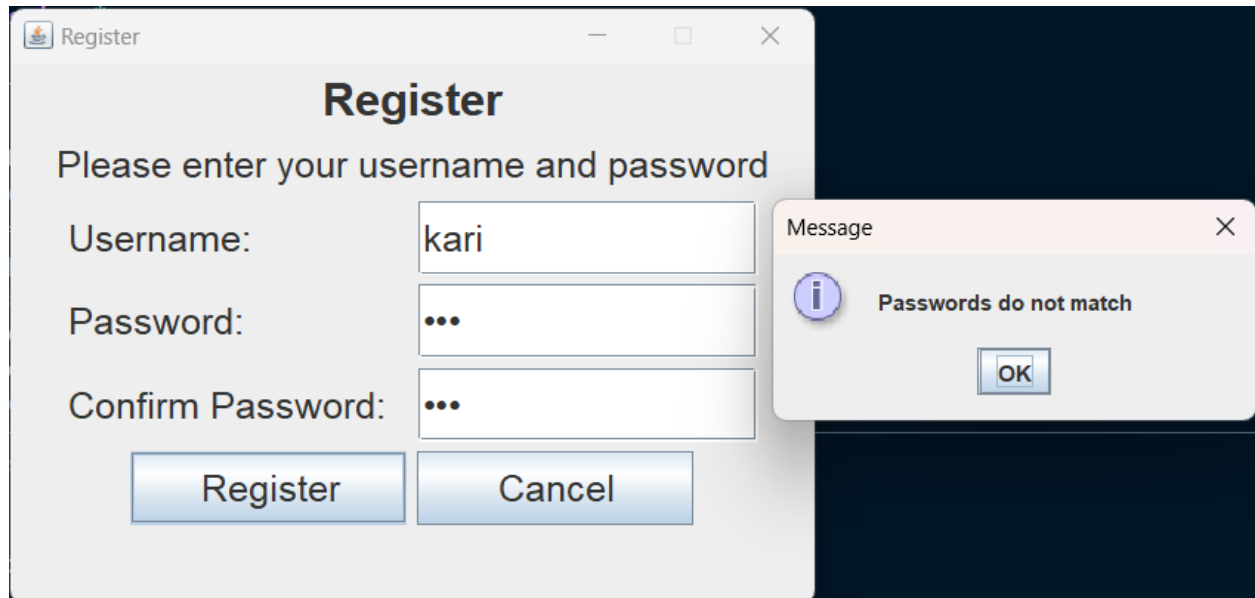
*Método que guarda la partida finalizada*

**Figure 6**

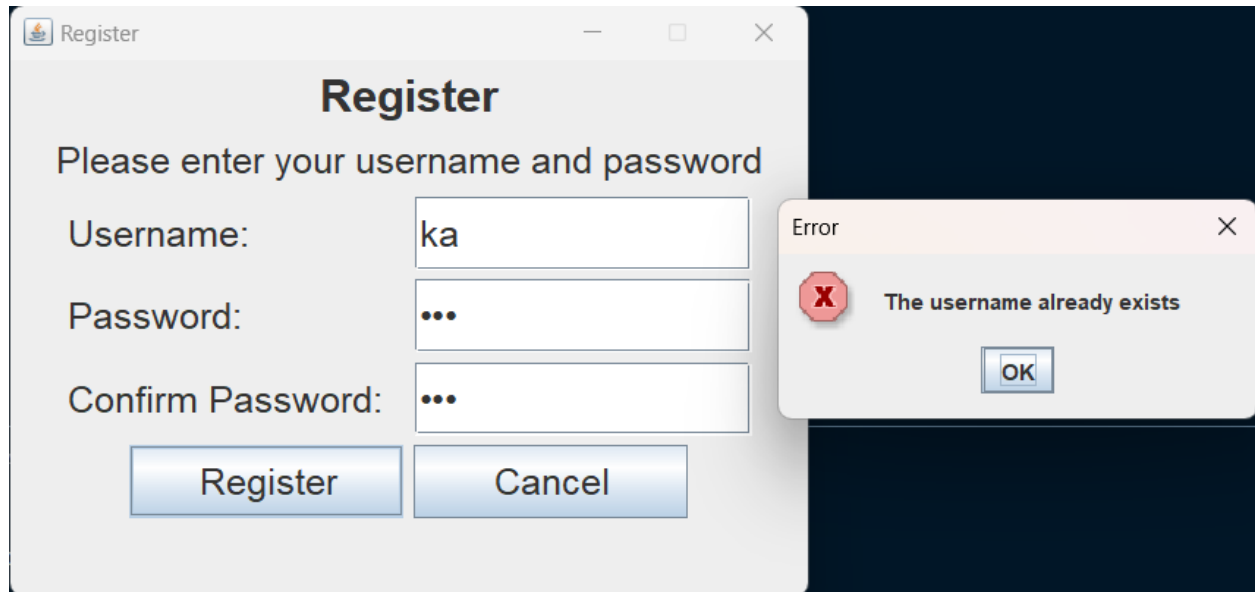
*Credenciales incorrectas*

**Figure 7**

*Espacios sin llenar*

**Figure 8**

*Las contraseñas no coinciden*

**Figure 9**

*Registro de usuario existente*