

ACADEMÍA JAVA XIDERAL MONTERREY, N.L

SEMANA 1 - EJERCICIO 2: SINGLETON

**Realizado por:
Luis Miguel Sánchez Flores**

PROBLEMÁTICA:

Más allá de completar la historia principal y de disfrutar de los modos tanto *offline* como en línea, los videojuegos también pueden ofrecer una serie de desafíos secundarios que ponen a prueba las habilidades de los jugadores.

Una vez que el jugador haya cumplido con las condiciones, el propio juego recompensa al jugador en forma de **logros** que los usuarios pueden exhibir con orgullo para demostrar su experiencia en determinados modos del videojuego, proporcionando una sensación de orgullo y logro que impulsa a los jugadores a completar todos y cada uno de los logros posibles, valorando el maestría del jugador en el juego y ganándose el reconocimiento de sus compañeros.

Para garantizar que todos los jugadores tengan una experiencia justa al recibir sus logros, y que se comuniquen con el mismo sistema cada vez que completen determinados retos, **los desarrolladores deben establecer un sistema centralizado y adecuado que proporcione un punto de acceso global a los logros para cada uno de los jugadores, ¿pero como?.**



SOLUCIÓN:

Un punto de acceso centralizado y global puede lograrse mediante el uso del patrón de diseño **Singleton**, que asegura que una clase tenga una, y **sólo una instancia que esté disponible de forma global durante todo el ciclo de vida de la aplicación**, evitando su modificación desde otras partes del código, garantizando que los demás componentes de la aplicación trabajen con el mismo objeto y se eviten datos duplicados u otros conflictos. El patrón de diseño Singleton **promueve la consistencia y la fiabilidad**, así como puede reducir el uso de recursos

Para este caso, el Singleton viene en la forma de la clase pública **SistemaLogros**, siguiendo la estructura del patrón de diseño **al privar tanto la instancia estática del sistema como el constructor de la clase**, dejando sólo el método estático `getInstance()` que devuelve la única instancia.

```
1 package singleton;
2
3 public class SistemaLogros {
4
5     // Inicializar la única instancia estática de SistemaLogros:
6     private static SistemaLogros sistemaLogros = new SistemaLogros();
7
8     // Declarar un arreglo de logros que puede ganar el jugador:
9     private Logro[] logros;
10
11     // CONSTRUCTOR PRIVADO:
12     private SistemaLogros() {
13
14         // Definir los títulos de los logros:
15         String[] titulos = {
16             "100% Completado",
17             "El mejor de todos",
18             "Mega-Combo!",
19             "De poco a poco...",
20             "Perfeccionista",
21             "Por los pelos!"
22         };
23
24         // Inicializar el arreglo de logros:
25         logros = new Logro[titulos.length];
26
27         // Para cada uno de los títulos definidos anteriormente,
28         // agregarlas al arreglo con un ID único:
29         for(int i = 0; i < titulos.length; i++)
30         {
```

```
36
37     // Función para recuperar el número de logros disponibles en el juego:
38     int getNumLogros() {
39         return logros.length;
40     }
41
42     // Función que retorna la única instancia de la clase:
43     public static SistemaLogros getInstance()
44     {
45         return sistemaLogros;
46     }
47
48
49     // Función para recompensar un logro al jugador correspondiente, de acuerdo con ID:
50     void anadirLogro(Jugador jugador, int logroID)
51     {
52         System.out.println("El usuario " + jugador.getUsuario() +
53             " ganó un logro! : " + logros[logroID].getTitulo() );
54     }
55 }
56
57 }
58
```

SistemaLogros también incluye un array de objetos **Logro**, que sólo contiene un ID y un nombre para cada uno de los logros.

```
1 package singleton;
2
3
4 // Clase de Logro
5 public class Logro {
6
7     private int ID;
8     private String titulo;
9
10    public Logro(int ID, String titulo) {
11        this.ID = ID;
12        this.titulo = titulo;
13    }
14
15    public String getTitulo() {
16        return titulo;
17    }
18
19    public int getID() {
20        return ID;
21    }
22
23 }
```

De forma similar, se define una clase simple de Jugador, el cual contiene un atributo de usuario, su respectivo getter y setter y se sobrescribe la función toString() heredada de Object para mostrar el usuario del jugador:

```
1 package singleton;
2
3 // Clase Jugador
4 public class Jugador {
5
6     private String usuario;
7
8     public Jugador(String usuario) {
9         this.usuario = usuario;
10    }
11
12    public String getUsuario() {
13        return usuario;
14    }
15
16    public void setUsuario(String usuario) {
17        this.usuario = usuario;
18    }
19
20    @Override
21    public String toString() {
22        return "JUGADOR : " + usuario;
23    }
24
25
26
27 }
28
```

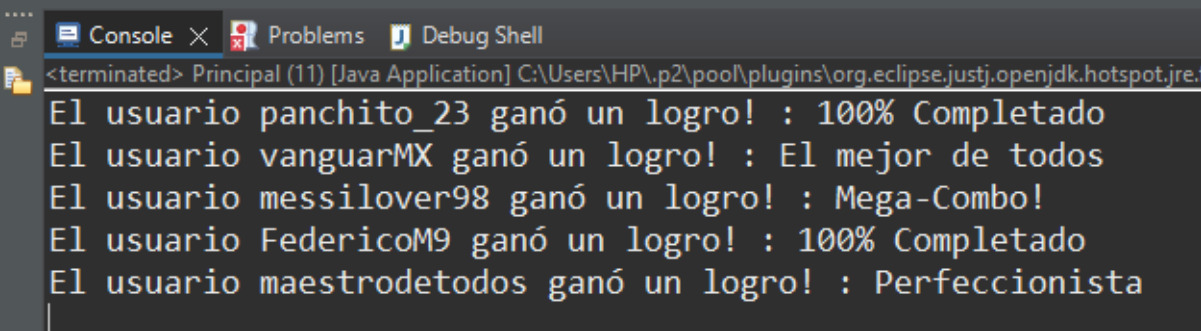
Finalmente, en la clase **Principal** dentro del método **main**, se comprueba el funcionamiento del Singleton mediante una simulación de una partida del juego, primero generando una lista de jugadores y un objeto Random que escogerá de forma aleatoria los logros que se van a ganar los jugadores:

```
11 public class Principal {
12
13     public static void main(String[] args) throws InterruptedException {
14         // Creación de jugadores
15
16         Jugador[] jugadores = {
17             new Jugador("panchito_23"),
18             new Jugador("vanguarMX"),
19             new Jugador("messilover98"),
20             new Jugador("FedericoM9"),
21             new Jugador("maestrodetodos"),
22         };
23
24
25         // Creación de objeto Random:
26         Random rand = new Random();
27
```

Se desarrolló un ciclo for en el que se llama la única instancia de SistemaLogros para agregar un logro aleatorio al jugador correspondiente. También se aprovecha el objeto Random para pausar el programa durante un periodo de tiempo aleatorio como parte de la simulación de la partida:

```
28
29     // Para cada uno de los jugadores de la lista:
30     for(Jugador jugador : jugadores)
31     {
32
33         // Pausar el programa durante un tiempo aleatorio (max 5 seg) para simular una partida del juego:
34         Thread.sleep(rand.nextInt(5000));
35
36         // Se recupera la única instancia de SistemaLogros:
37         SistemaLogros sistemaLogros = SistemaLogros.getInstance();
38
39         // Se añade un logro aleatorio al jugador:
40         sistemaLogros.anadirLogro(jugador,
41             rand.nextInt(sistemaLogros.getNumLogros()));
42     }
43
44
```

La salida del programa muestra que cada uno de los cinco jugadores de la lista recibió un logro aleatorio utilizando la misma instancia de SistemaLogros, simplificando la comunicación entre los jugadores y el sistema y asegurando que los logros se gestionan de manera justa y uniforme.



The screenshot shows the Eclipse IDE's console window. The title bar includes 'Console', 'Problems', and 'Debug Shell'. The console output displays five achievement messages, each on a new line. The messages are: 'El usuario panchito_23 ganó un logro! : 100% Completado', 'El usuario vanguarMX ganó un logro! : El mejor de todos', 'El usuario messilover98 ganó un logro! : Mega-Combo!', 'El usuario FedericoM9 ganó un logro! : 100% Completado', and 'El usuario maestrodetodos ganó un logro! : Perfeccionista'. The text is white on a dark background.

```
<terminated> Principal (11) [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.i  
El usuario panchito_23 ganó un logro! : 100% Completado  
El usuario vanguarMX ganó un logro! : El mejor de todos  
El usuario messilover98 ganó un logro! : Mega-Combo!  
El usuario FedericoM9 ganó un logro! : 100% Completado  
El usuario maestrodetodos ganó un logro! : Perfeccionista
```