Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Reporte Final del Proyecto de Integración

Licenciatura en Ingeniería en Computación

Modalidad de Proyecto Tecnológico

Juego para apoyar la comprensión del paradigma de programación orientada a objetos

Alumno:

Rodríguez Nogal Ricardo

Matrícula:

206211627

Asesores:

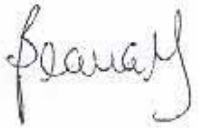
Dra. María Lizbeth Gallardo López
Dra. Beatriz Adriana González Beltrán

Trimestre 2014 otoño

9 de enero de 2015

Yo, María Lizbeth Gallardo López, *declaro que aprobé el contenido del presente Reporte de Proyecto de Integración* y *doy mi autorización* para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Yo, Beatriz Adriana González Beltrán, *declaro que aprobé el contenido del presente Reporte de Proyecto de Integración* y *doy mi autorización* para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Yo, Ricardo Rodríguez Nogal, *doy mi autorización* a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Resumen

El paradigma orientado a objetos permite resolver problemas mediante una apropiada definición y clasificación de objetos, así como de las relaciones que guardan entre ellos. Para algunos estudiantes de ingeniería de la UAM-A resulta difícil realizar las abstracciones necesarias en la solución a un problema a través del paradigma orientado a objetos. Por otra parte, se ha comprobado que los juegos ayudan a estimular las capacidades de razonamiento, memoria y comprensión.

En este proyecto se desarrolló e implementó un juego para apoyar la comprensión del paradigma de programación orientada a objetos. En particular, es una aplicación de escritorio que integra tres juegos: **sopa de instrucciones**, **encuentra el código** y **memoria**.

Para el desarrollo de este proyecto, se utilizó el proceso unificado, el cual consta de iteraciones dentro de las cuales se encuentran las etapas de análisis, diseño, implementación y pruebas. En cada iteración se avanza en una de las etapas mencionadas y se afinan los detalles que sean necesarios en las etapas anteriores.

Para el presente proyecto, se implementaron los módulos de juego, continuando con los módulos de gestión de usuarios y finalmente con la implementación del sistema de puntuaciones. En cada implementación se realizaron pruebas individuales a cada módulo y al final a todo el sistema en conjunto.

Tabla de contenido

Índice de figuras

1 Introducción

El paradigma orientado a objetos "es un método de implementación en el cuál los programas están organizados como grupos cooperativos de objetos, cada uno de los cuales representa una instancia de alguna clase, y estas clases, todas son miembros de una jerarquía de clases unidas vía relaciones de herencia" [1].

El paradigma orientado a objetos permite resolver problemas mediante una apropiada definición y clasificación de objetos, así como de las relaciones que guardan entre ellos. Un objeto es un ejemplar de una clase donde se definen sus atributos (propiedades) y sus métodos (comportamiento); cada objeto tiene una identidad que lo distingue de otros objetos de la misma clase.

En la solución a un problema, empleando este paradigma, se puede emplear el Lenguaje de Modelado Unificado (UML), el cual está compuesto por elementos estructurales, elementos de comportamiento, elementos de agrupación, elementos de anotación y diagramas que muestran de una manera gráfica un sistema desde diferentes perspectivas [2].

Para algunos estudiantes de ingeniería de la UAM-A resulta difícil realizar las abstracciones necesarias en la solución a un problema a través del paradigma orientado a objetos.

Por otra parte, se ha comprobado que los juegos ayudan a estimular las capacidades de razonamiento, memoria y comprensión [3]. Para apoyar a la comprensión del paradigma orientado a objetos se propuso realizar una aplicación como un juego.

2 Antecedentes

2.1 Referencias externas

Tesis

Diseño de *software* educativo para la enseñanza de la programación orientada a objetos basado en la taxonomía de *Bloom*

Este trabajo presenta la planeación y realización de contenidos de *software* educativo basándose en la taxonomía de *Bloom* en los niveles de: conocimiento, compresión y aplicación. Los alumnos primero captan la información (nivel de conocimiento), la interiorizan (nivel de comprensión) y después la aplican ante nuevas situaciones (nivel de aplicación) [4]. *Bajo* esta taxonomía, en este proyecto se aborda únicamente el nivel de

comprensión, dado que no se pretende enseñar el paradigma de POO, sino poner en práctica los conceptos principales abordados en clase.

Artículos

Enseñando y aprendiendo Programación Orientada a Objetos en los primeros cursos de Programación: la experiencia en la Universidad ORT Uruguay

En este artículo se describe el problema de la enseñanza del paradigma orientado a objetos, y de cómo se abordó en la Universidad ORT Uruguay [5]. Se implementaron notas de curso, se preparó material audiovisual y *software* que involucraban conceptos básicos de POO. En clase, los alumnos realizaban prácticas, llevándose los ejercicios a casa junto con tareas para reforzar lo aprendido en clase. En nuestro proyecto, los juegos que se proponen son una alternativa para que los alumnos reafirmen los conocimientos adquiridos en clase.

Software

Aprendiendo POO a través de la cultura indígena venezolana

Este software es un videojuego que aborda los conceptos básicos de POO a través de la presentación de objetos tridimensionales que muestran sus características (atributos) y las acciones que pueden realizarse sobre ellos (métodos), proponiendo distintos retos en los cuales se deben identificar los atributos del objeto, así como sus métodos, a saber: escalar, rotar, crear y cambiar de color. El *software* también tiene un clasificador de objetos para agruparlos de acuerdo a la clase a la que pertenecen [6]. Nuestro proyecto también se enfoca en los conceptos básicos de POO pero a través de tres juegos diferentes en los cuales se aplican estos conceptos para la resolución de problemas básicos que ayuden a la comprensión de POO.

2.2 Referencias Internas

Sistema Tutor *Web* para el Aprendizaje de Programación Orientada a Objetos

Este proyecto presenta un sistema para la enseñanza de la programación orientada a objetos, proponiendo ejercicios de opción múltiple sobre conceptos; además, cuenta con un *chat* que permite a los alumnos mantenerse en contacto con algún asesor; finalmente, cuenta con un registro de material electrónico de apoyo, tales como: archivos *pdf* y videos [7]. En nuestro proyecto de integración se proponen tres juegos que apoyen en el aprendizaje del paradigma orientado a objetos.

Aplicación para el apoyo a la enseñanza de la UEA Métodos Numéricos

Este proyecto presenta una aplicación para apoyar a la enseñanza de la UEA Métodos numéricos, implementando los algoritmos que se abordan en el curso, mostrando paso a paso la ejecución del algoritmo y sus resultados parciales y totales [8]. Nuestro proyecto de integración no pretende mostrar la ejecución de algoritmos, sino ofrece un medio para ejercitar los principales conceptos teórico-prácticos del paradigma orientado a objetos a través de un juego.

Aplicación *Android* para la práctica de verbos compuestos del idioma inglés

Este proyecto presenta una aplicación Android para dispositivos móviles que apoya en el aprendizaje de los verbos compuestos del idioma inglés. Consta de tres juegos donde se involucran imágenes y oraciones, además de proponer tres niveles: principiantes, intermedios y avanzados [9]. Nuestro proyecto de integración también cuenta con tres juegos y tres niveles, pero no se propuso para dispositivos móviles porque los códigos y los diagramas que se construyeron no pueden ser visualizados cómodamente en una pantalla pequeña.

3 Justificación

En la actualidad la programación orientada a objetos ha cobrado gran relevancia en la implementación de aplicaciones que usamos cotidianamente. Para los alumnos de ingeniería de la UAM-A, es relevante comprender este paradigma para ampliar su formación académica, y satisfacer una demanda del mercado laboral.

Con el juego que se planteó desarrollar en este proyecto, se espera apoyar a los estudiantes de la UAM-A, en la comprensión del paradigma orientado a objetos. De acuerdo con *Dewey* [6], una de las formas de reforzar los conocimientos adquiridos es llevándolos a la práctica de una manera lúdica. Nosotros abordamos el paradigma a través de tres juegos. Estos juegos están enfocados en relacionar diagramas de clases con su representación en código; relacionar conceptos del paradigma con su respectiva representación en *UML*, y la estructuración de un programa orientado a objetos, a partir de líneas de código en desorden.

4 Objetivos

4.1 Objetivo general

Desarrollar e implementar una aplicación para apoyar a los estudiantes en la comprensión del paradigma de programación orientada a objetos.

4.2 Objetivos específicos

- Diseñar e implementar el juego **Sopa de instrucciones**.
- Diseñar e implementar el juego **Encuentra el código**.
- Diseñar e implementar el juego **Memoria**.
- Integrar los juegos en una aplicación y establecer la secuencia de los juegos.
- Diseñar e implementar un módulo para la gestión de los usuarios.
- Diseñar e implementar un módulo que permita elegir opciones del juego.
- Diseñar e implementar una base de datos que contenga la información de cada juego; así como la información que permita determinar el avance del usuario durante el juego.

5 Marco teórico

Paradigma de programación orientada a objetos

En este paradigma de programación se combinan en un sólo módulo todos los datos y funciones que operan sobre estos datos. Los módulos se denominan objetos que se organizan en un conjunto finito. Estos objetos se pueden llamar unos a otros a través del paso de mensajes. Sus principales características son: abstracción, encapsulamiento de datos, ocultación de datos, herencia y polimorfismo [10].

La **abstracción** es la propiedad de los objetos que consiste en tener en cuenta sólo los aspectos más importantes desde un punto de vista predeterminado y no tomar en cuenta los aspectos restantes.

El **encapsulamiento de datos** consiste en agrupar datos y operaciones relacionadas bajo la misma unidad de programación.

La **ocultación de datos** permite separar el aspecto de un componente, definido por su interfaz con el exterior, de sus detalles internos de implementación.

La **herencia** permite definir nuevas clases a partir de otras clases ya existentes de modo que presentan las mismas características y comportamiento de éstas, así como otras adicionales.

El **polimorfismo** es la propiedad de que una función o un operador actúen de modo diferente en función del objeto sobre el que se aplican.

6 Desarrollo del sistema

6.1 Metodología empleada en el desarrollo del proyecto

La metodología utilizada en este proyecto fue el Proceso Unificado (PU) [11].

En la primera iteración se diseñó e implementó la base de datos, el módulo de selección de juego y el **juego de memoria**.

En la segunda iteración, se diseñó e implementó el **juego sopa de instrucciones** y el módulo de gestión de usuario.

En la tercera iteración se realizaron las pruebas tanto del módulo de gestión de usuario como del **juego de memoria**.

En la cuarta iteración se diseñó e implementó tanto el **juego encuentra el código** como el módulo que permite seleccionar el nivel de cada juego y se realizaron las pruebas correspondientes. Además se diseñó e implementó el módulo de puntuaciones.

6.2 Diseño del sistema

Esta sección incluye el diagrama de casos de uso, un conjunto de casos de uso de texto, el diagrama de clases, el diagrama entidad-relación y el diagrama de la arquitectura del sistema.

6.2.1 Diagrama de casos de uso

El diagrama de casos de uso tiene dos casos de uso esenciales que consiste en gestionar a un jugador y jugar (ver Figura 1).

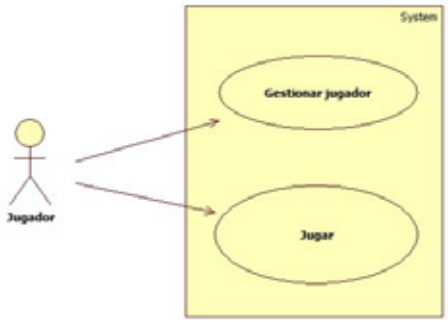


Figura 1: Diagrama de casos de uso principales.

El caso de uso gestionar jugador tiene como subcasos de uso registrar jugador e iniciar sesión (ver Figura 2).

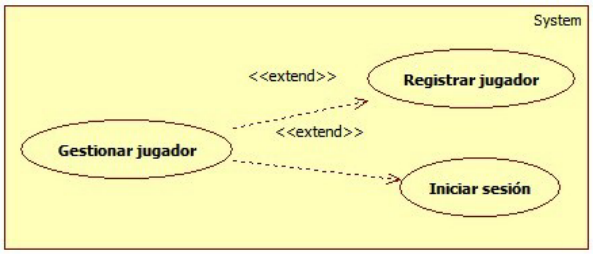


Figura 2: Diagrama de caso de uso gestionar jugador.

6.2.2 Casos de uso de texto

En este apartado se presentan los casos de uso registrar jugador, iniciar sesión y jugar.

6.2.2.1 *Caso de uso: Registrar jugador*

Actor principal: Jugador.

Intereses del personal involucrado.

El jugador desea registrarse en el sistema

Precondiciones: Que el jugador no esté registrado.

Garantías de éxito: El nuevo jugador se encuentra registrado para poder jugar.

Escenario Principal:

1. El jugador enciende el sistema.

2. El jugador solicita un registro para añadir sus datos.
3. El jugador introduce el nombre, apellido paterno, apellido materno, matrícula, el correo electrónico, un nombre de usuario y una contraseña.
4. El sistema registra los datos.
5. El sistema finaliza la operación.

Extensiones:

3ª. El jugador identifica que el nombre, apellido paterno, apellido materno, matrícula, el correo electrónico, nombre de usuario y/o contraseña son incorrectos.

a. El jugador modificar el nombre, apellido paterno, apellido materno, matrícula, el correo electrónico, nombre de usuario y/o contraseña

b. El jugador cancela el registro.

c. El jugador inicia un nuevo registro.

Frecuencia: Media.

6.2.2.2 *Caso de uso: Iniciar sesión*

Actor principal: Jugador.

Intereses del personal involucrado:

El jugador desea ingresar al sistema para poder jugar.

Precondiciones: El jugador debe estar registrado en el sistema.

Garantías de éxito: El jugador puede jugar en el sistema.

Escenario Principal:

1. El Jugador enciende el sistema.
2. El Jugador ingresa su nombre de usuario y contraseña.
3. El sistema verifica que el nombre de usuario y contraseña sean válidas.
4. El sistema da la bienvenida al jugador.
5. El sistema muestra la pantalla de jugar.

Extensiones:

4ª. El sistema no deja autentificar al jugador.

a. El jugador verifica el nombre de usuario y/o contraseña

i. El sistema reintenta la autentificación.

Frecuencia: Media.

6.2.2.3 *Caso de uso: Jugar*

Actor principal: Jugador.

Intereses del personal involucrado:

El jugador desea jugar.

Precondiciones: El jugador debe haber iniciado sesión en el sistema.

Garantías de éxito: El jugador quiere jugar en el sistema.

Escenario Principal:

1. El jugador escoge un juego.
2. El jugador selecciona la dificultad del juego.
3. El jugador selecciona el nivel del juego.
4. El sistema inicia el juego.
5. El jugador juega al juego seleccionado con la dificultad y nivel seleccionados.
6. El sistema muestra las puntuaciones.
7. El sistema regresa a la selección de nivel.

Extensiones:

1-3. El sistema no deja elegir el juego.

a. Verificar que el sistema funcione correctamente.

ii. reiniciar el sistema.

iii. reintentar la selección de juego.

Frecuencia: Continua.

6.2.3 Diagramas de clases

La Figura 3 muestra el diagrama de paquetes del sistema. Existen cuatro paquetes: `com.ricardo.pi`, `com.ricardo.pi.actor`, `com.ricardo.pi.common` y `com.ricardo.pi.pantallas`. Esta división obedece a la necesidad de mantener las clases del juego similares agrupadas para mantener una organización de las mismas y facilitar su búsqueda y manipulación al momento del desarrollo de la aplicación.

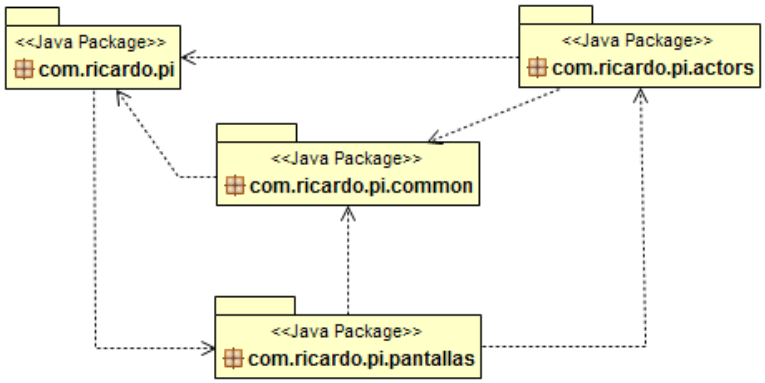


Figura 3: Diagrama de paquetes del sistema.

La Figura 4 muestra la clase principal `POOGame` que permite iniciar la aplicación. Esta clase hereda a la clase *Game* del *framework libGDX* [15] y define las pantallas a usar que están contenidas en el paquete `com.pi.ricardo.pantallas`, así como los recursos audiovisuales y los tipos de fuentes contenidos en el juego. La clase `POOGame` contiene una instancia de *AssetManager* que permite gestionar los recursos audiovisuales dentro del juego.

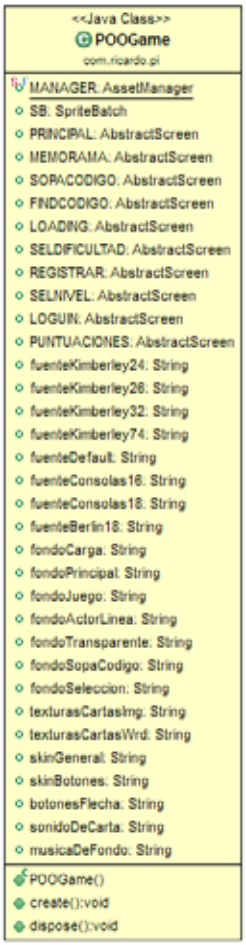


Figura 4: Clase principal POOGame

La Figura 5 nos detalla el paquete `com.ricardo.pi.actors`. En este paquete se encuentran las clases que heredan de la clase *Actor* del *framework libGDX*. Estas clases se incluyen en los escenarios contenidos en las pantallas del paquete `com.ricardo.pi.pantallas`. La clase `ActorCarta` pertenece al **juego de memoria**, la clase `LineaCodigo` pertenece al **juego sopa de instrucciones** y las clases `ActorItem`, `ActorEjercicio` y `ActorRespuesta` pertenecen al **juego encuentra el código**.

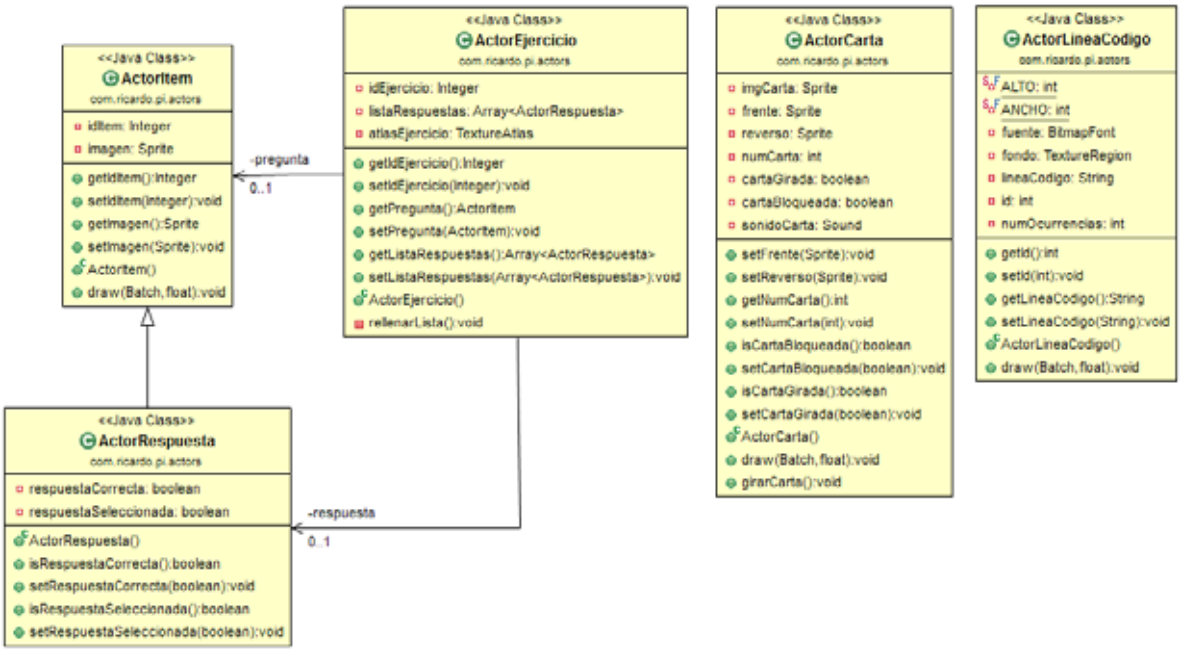


Figura 5: Diagrama de clases del paquete com.ricardo.pi.actors

La Figura 6 detalla el paquete `com.ricardo.pi.common`. Este contiene aquellas clases que son comunes a las clases contenidas en los paquetes ya mencionados y se encargan de leer archivos, crear temporizadores, gestionar usuarios, gestionar puntuaciones, manipular la base de datos, validar datos introducidos en formularios y mostrar cuadros de diálogo dentro de cada pantalla de juego.

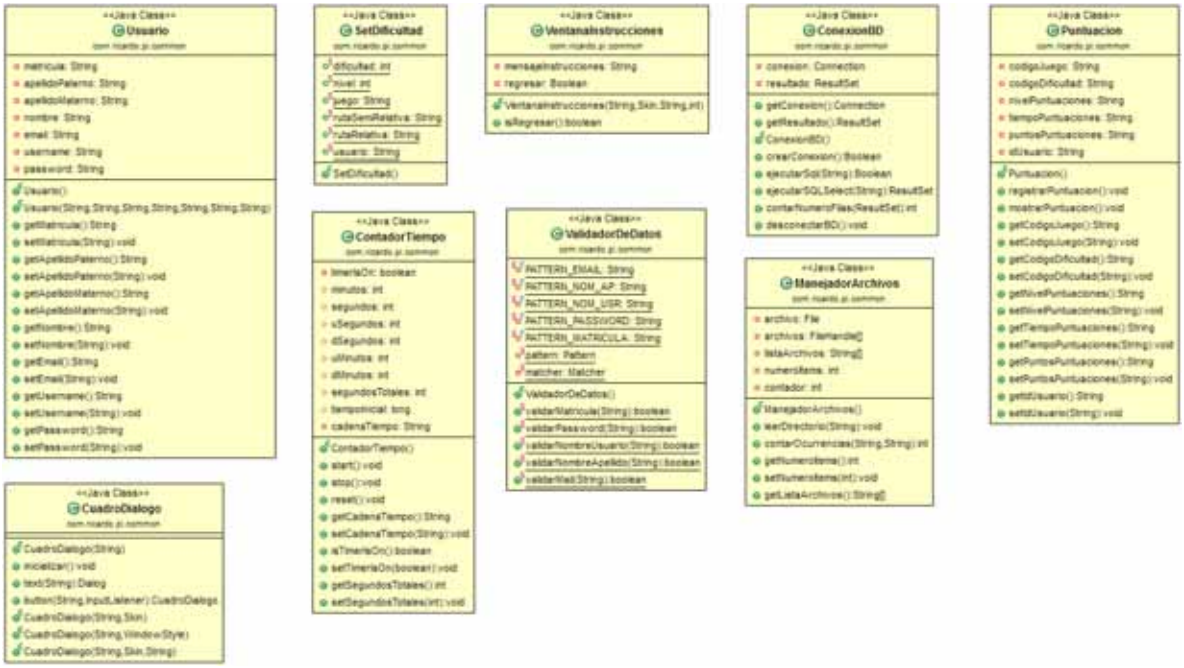


Figura 6: Diagrama de clases del paquete com.ricardo.pi.common

Por último, el paquete `com.ricardo.pi.pantallas` mostrado en la Figura 7 contiene una clase abstracta que implementa a la interfaz *Screen* del *framework libGDX* y de la cual se heredan las pantallas o vistas que conforman el juego. En este paquete se encuentra la pantalla de autentificación y registro del jugador las cuales forman parte del caso de uso Gestionar jugador (ver Figura 2). Además se encuentran las pantallas de selección de dificultad, selección de nivel y la pantalla principal del juego, como también se encuentran las pantallas memorama, sopa de código y encuentra el código, anexando la pantalla de puntuaciones que en conjunto forman el caso de uso Jugar.

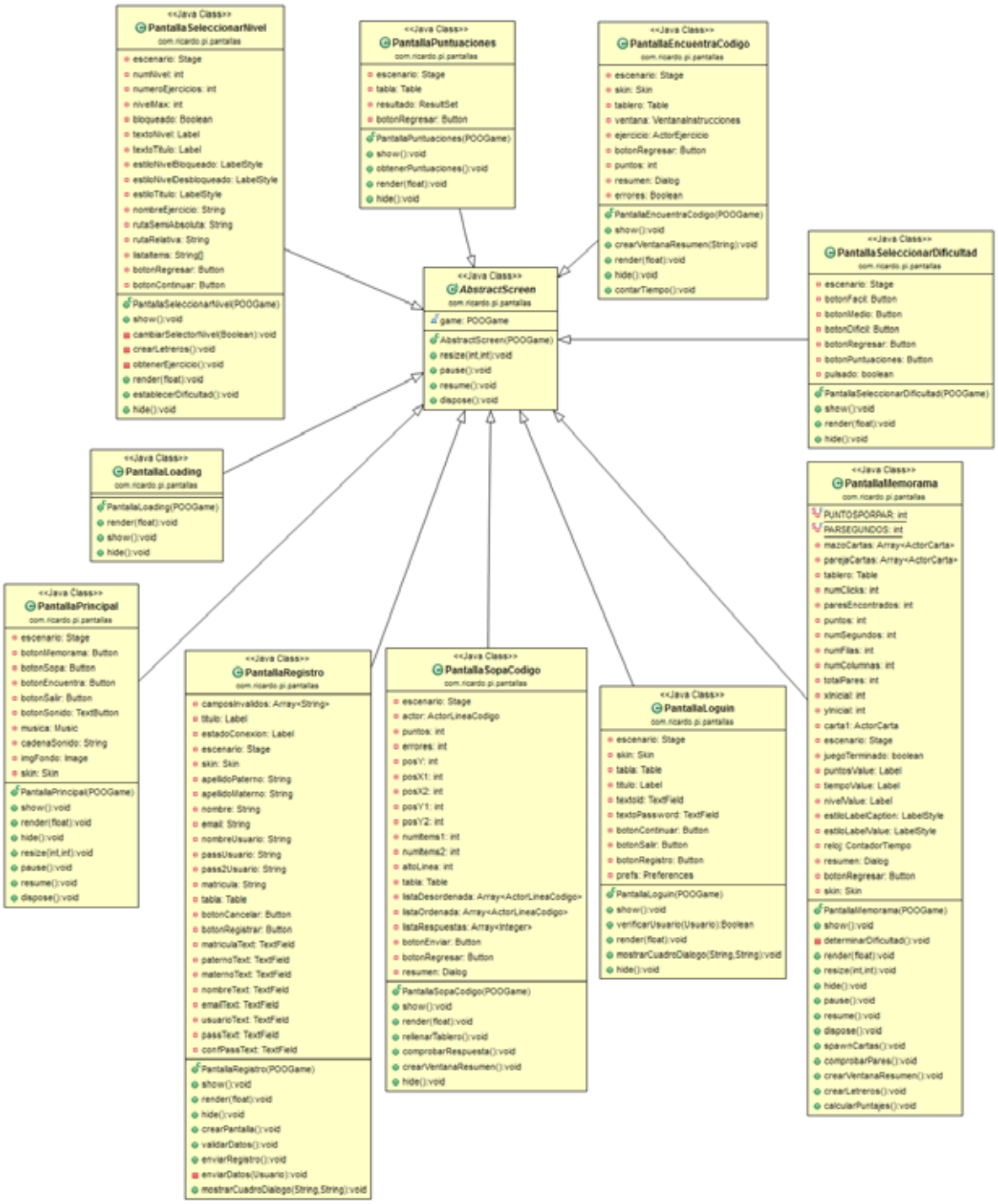


Figura 7: Diagrama de clases del paquete com.ricardo.pi.pantallas

6.2.4 Arquitectura del sistema

La Figura 8 muestra el diagrama de arquitectura del sistema. El sistema muestra una arquitectura jerárquica heredada del *framework* utilizado para el desarrollo de este juego (*libGDX*). El sistema se encuentra conformado por un escenario el cual puede contener actores que gozan de un comportamiento y atributos independientes. Ambos, escenario y actores responden a eventos que son recibidos a través de un *listener* que se encuentra ligado a un dispositivo de entrada. Los comportamientos del escenario y los actores son enviados al módulo de *render* que maneja la salida por pantalla de la computadora.

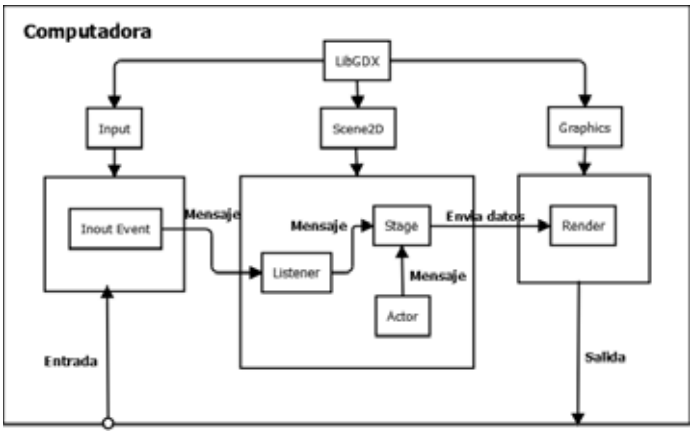


Figura 8: Diagrama de arquitectura del sistema.

6.2.5 Estructura de la base de datos

La base de datos representada en el diagrama de la Figura 9 consta de cuatro tablas: la tabla usuario, en donde se almacenan los datos concernientes al jugador, la tabla juego que funge como un catálogo el cual contiene la *id* del juego en el sistema, un código de identificación de tres caracteres y la descripción del juego.

La tabla dificultad es un catálogo que contiene el *id* de la dificultad, un código de tres caracteres y la descripción de la dificultad.

Finalmente, la tabla puntuaciones se encuentra compuesta con los campos de *id* de usuario, *id* de juego, *id* de dificultad, así como el número de nivel en el que se registra la puntuación, contiene los valores de puntuación y tiempo obtenidos en el juego.

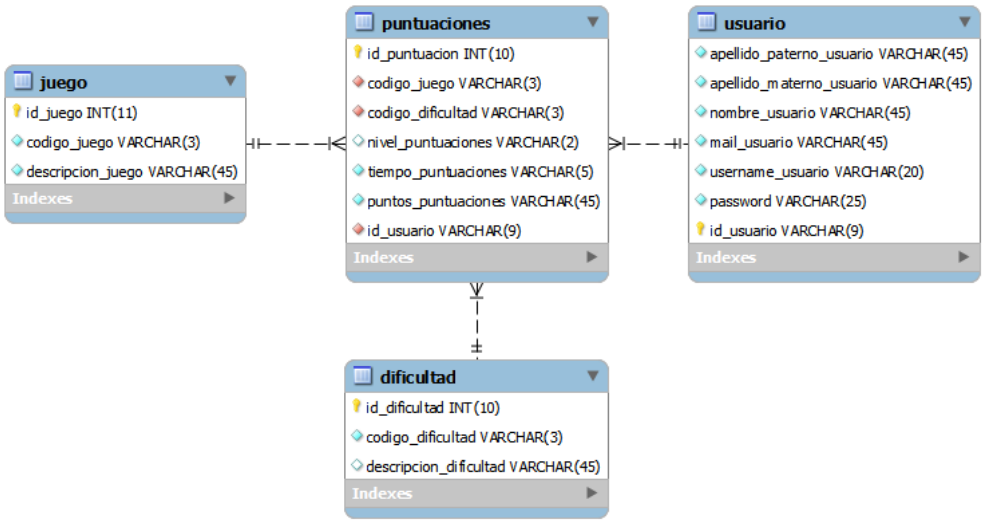


Figura 9: Diagrama Entidad-Relación de la base de datos.

6.3 Uso del juego

A continuación se describen los pasos para ejecutar y utilizar el juego.

1. Ingresa en la carpeta de la aplicación y ejecutarla haciendo doble click sobre el ícono (Figura 10).

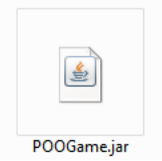


Figura 10: Ícono de la aplicación.

2. Aparece la pantalla de autenticación de jugador (Figura 11) si se cuenta con un nombre de usuario y contraseña ingresarlos y seguir al paso 4.

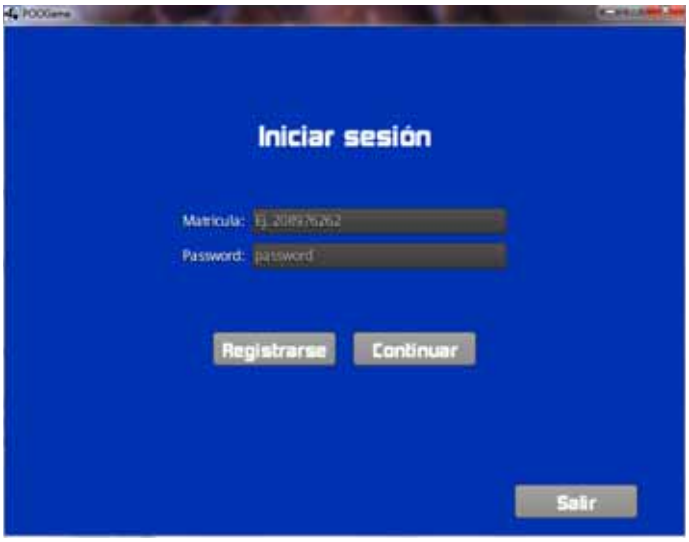


Figura 11: Ventana de autenticación de jugador.

3. En la pantalla de registro de jugador (Figura 12), introducir los datos requeridos. Todos los campos son requeridos. Los campos de nombre, apellidos sólo pueden contener caracteres alfabéticos; el campo de matrícula sólo puede contener dígitos y tiene una longitud máxima de 9; la contraseña (*password*) debe contener al menos una letra mayúscula, un dígito y una longitud mínima de 4 caracteres.

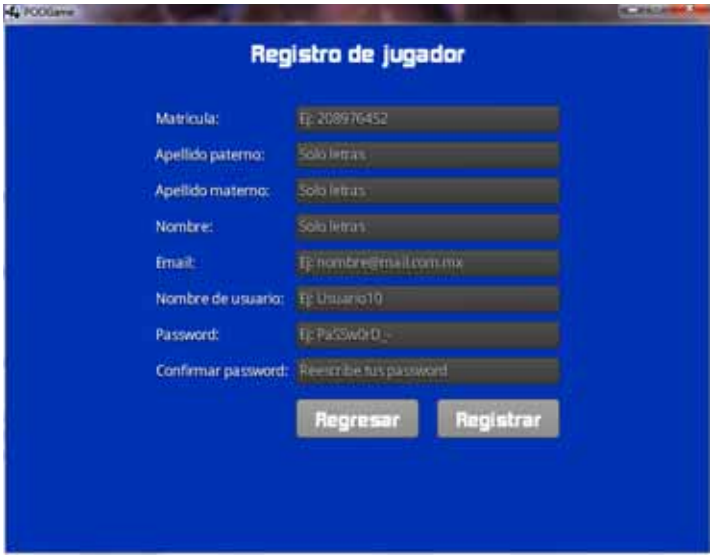


Figura 12: Ventana de registro de jugador.

4. Una vez autenticado el jugador, en la ventana principal (Figura 13) del juego, se puede elegir jugar uno de los tres juegos y apagar o encender la música.

Figura 13: Ventana principal del juego.

5. Para cada juego, se puede elegir de entre tres dificultades posibles: **fácil**, **medio** y **difícil**, esto mediante la ventana de Elegir dificultad (Figura 14).

Figura 14: Ventana de elección de dificultad.

Dentro de esta misma ventana, también es posible consultar la tabla de puntuaciones (Figura 15) para el juego seleccionado.

Figura 15: Ventana de puntuaciones por juego.

6. Para el **juego de Memoria** (Figura 16), el objetivo es encontrar todos los pares en el menor tiempo posible. Se tiene un tiempo límite para cada dificultad y se recibe una bonificación por terminar dentro del tiempo límite.

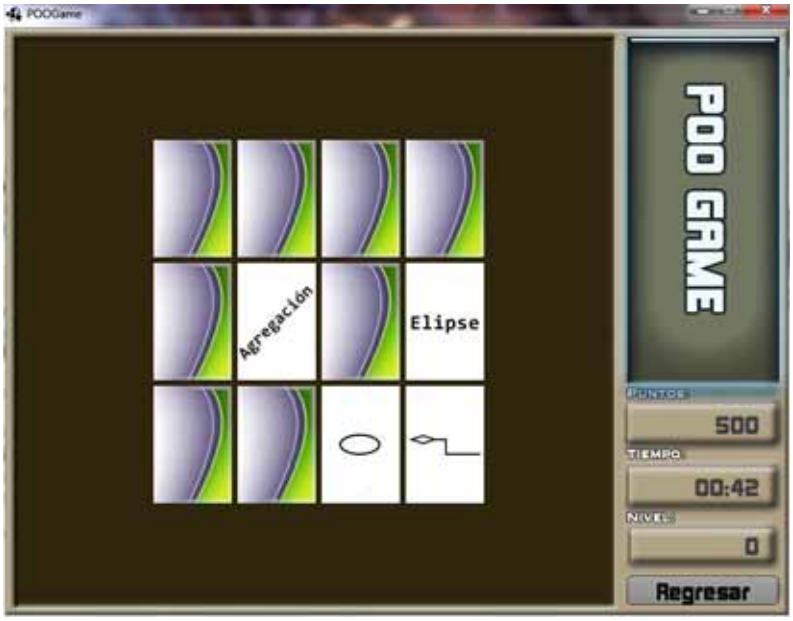


Figura 16: Ventana de juego de Memoria.

Los juegos de **Encuentra el código** y **Sopa de código** cuentan con una ventana de selección de nivel (Figura 17), los cuales se van desbloqueando a medida que se juega, es decir, para jugar un nivel, se debe desbloquear el anterior inmediato al

nivel en cuestión. Los niveles bloqueados se muestran en otro color (Figura 18) y no se puede acceder a ellos.

Figura 17: Ventana de selección de nivel (desbloqueado).

Figura 18: Ventana de selección de nivel (bloqueado).

En el juego **Sopa de instrucciones** (Figura 19), el objetivo es ordenar un fragmento desordenado de código en Java y enviarlo al sistema para su evaluación. El sistema determina el número de errores que se tuvo y en caso de no existir errores, la puntuación es enviada y se refleja en la tabla de puntuaciones.

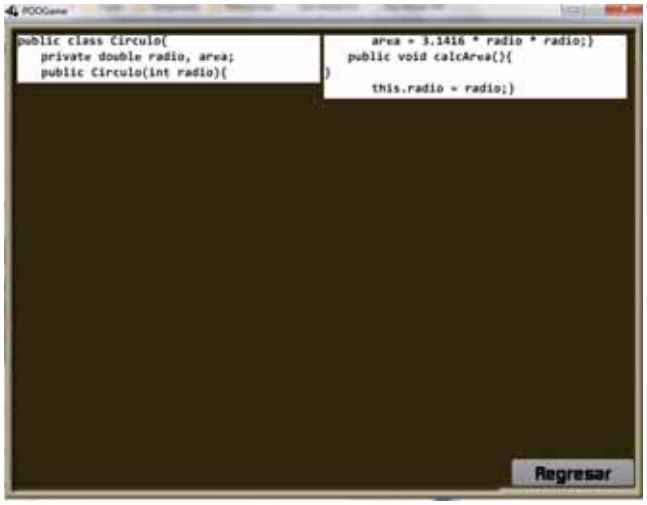


Figura 19: Ventana del juego Sopa de instrucciones.

El objetivo del juego Encuentra el código (Figura 20), es elegir de entre tres fragmentos de código en Java, aquel que sea equivalente a un diagrama UML mostrado en la parte superior. Al elegirlo, el sistema califica si la respuesta es correcta o no.

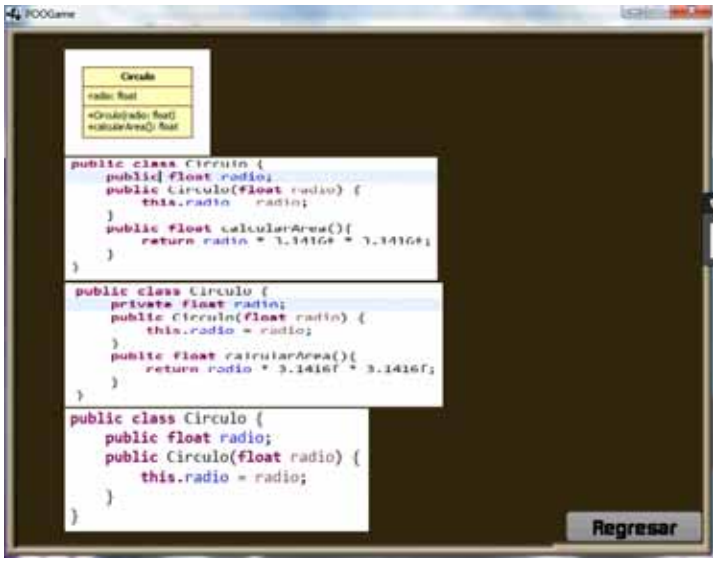


Figura 20: ventana del juego Encuentra el código.

6.4 *Hardware* y *software* necesario

En esta sección se describe el hardware y software necesarios para el desarrollo de la aplicación, la ejecución de la aplicación y la instalación de la misma para el usuario final.

6.4.1 Tecnología para el desarrollo de la aplicación

- *Framework* de desarrollo del juego: libGDX 1.0 [12].
- *SDK*: Java EE JDK 1.6.0. [13]
- Gestor de base de datos: MySQL 5.6. [14]
- *Framework* de traducción *HTML5*: *Google Development Kit*(*GWT*) 2.6.1 [15].
- *IDE*: *Android Developer Tools* (*ADT*) *Bundle* [16].

6.4.2 Tecnología para la instalación y puesta en marcha de la aplicación

6.4.2.1 *Requisitos para la instalación*

- Servidor de base de datos: MySQL 5.
- Computadora con 1 MB de RAM mínimo, con velocidad de 1.6GHz mínimo.

6.4.2.2 *Requisitos para la ejecución*

- Computadora con 1 MB de RAM mínimo, con velocidad de 1.6GHz mínimo.
- Java JRE 7.

7 Resultados

Se realizó e implementó el juego **memoria** que consta de tres dificultades, a saber: **fácil**, **media** y **difícil**. Cada nivel de difícultad consta de 8, 9 y 12 pares, respectivamente.

Se realizó el diseño e implementación de la base de datos en un servidor local y se realizaron pruebas de conectividad y manipulación de datos resultando satisfactorias.

Se realizó el juego **sopa de instrucciones** utilizando archivos de texto que contienen un listado de código de programa en java a manera de prueba y esta fue exitosa.

Se realizó el juego **encuentra el código** utilizando un mapa de imágenes para cada ejercicio. El mapa de imágenes contiene una región que muestra un diagrama de clases y otras tres regiones que muestran distintos códigos de programación que pueden corresponder al diagrama mostrado.

Se realizó el módulo que permite seleccionar los juegos, realizando pruebas seleccionando los juegos y resultando satisfactorias.

Se realizó el módulo que permite seleccionar los niveles para los juegos de **sopa de instrucciones** y **encuentra el código**.

Se realizó el módulo de registro de jugador. Se realizaron validaciones de los datos por medio de expresiones regulares, comprobando que los datos introducidos sean válidos o los campos no se encuentren vacíos.

Se realizó el módulo de autenticación de jugador que permite al jugador autenticarse en el sistema para poder jugar y guardar sus avances, así como las puntuaciones obtenidas.

Se realizó el módulo de puntuaciones que permite registrar en la base de datos las puntuaciones obtenidas por los usuarios en cada juego, así como mostrar las puntuaciones para cada juego ordenadas de manera descendente.

8 Análisis y discusión de resultados

El presente proyecto está realizado para un jugador a la vez y consta de tres juegos: **memoria**, **sopa de instrucciones** y **encuentra el código**. Los juegos mencionados cuentan con tres dificultades cada uno: **fácil**, **medio** y **difícil**. En el caso de los juegos de **encuentra el código** y **sopa de instrucciones**, se cuenta en cada dificultad con 5 niveles ascendentes a los cuales se puede acceder únicamente si se ha completado el nivel que le antecede. Para el juego de **memoria**, sólo se cuenta con tres niveles de dificultad.

En el sistema se realizaron pruebas de validación de datos en la base de datos, utilizando expresiones regulares para este fin. Se realizaron pruebas de lectura de directorios y archivos para el juego de sopa de instrucciones manejando las excepciones necesarias en caso de no encontrar los archivos o directorios.

Se realizaron pruebas del correcto funcionamiento del sistema donde tres usuarios diferentes fueron registrados en la base de datos, y donde ellos lograron autentificarse con éxito. Cada uno de los jugadores probaron los tres juegos y sus puntuaciones se enviaron correctamente a la base de datos, finalmente los avances se guardaron para cada usuario registrado.

9 Conclusiones

Los objetivos de diseñar e implementar los juegos se cumplieron por completo. Sin embargo, inicialmente se tenía como objetivo la integración de los juegos en una aplicación web pero se decidió hacer una aplicación de escritorio dado que no fue posible utilizar el *framework* para realizar conexiones asíncronas El módulo que permite gestionar a los usuarios está completo así como el módulo que permite elegir las opciones del juego.

10 Perspectivas del proyecto

Este proyecto puede mejorarse agregando más juegos. Por ejemplo, un juego que contenga ejercicios de sintaxis. Otra mejora puede presentarse en el diseño visual. Además de poder extenderse para plataformas Android.

Este proyecto se diseñó como aplicación de escritorio, una mejora sería que se pudiera implementar como una aplicación web.

Referencias

[1] Greiff W. R. Paradigma vs Metodología; El Caso de la POO (Parte II). Soluciones Avanzadas. Ene-Feb 1994. pp. 31-39.

[2] I. Jacobson, “El lenguaje unificado de modelado, ” Editorial Addison Wesley, 2002.

[3] J. Dewey “Democracy an education. An introduction to the philosophy of education, ” Editorial The Macmillan company, 2004.

[4] I. H. Losada, “Diseño de software educativo para la enseñanza de la programación orientada a objetos basado en la taxonomía de Bloom,” M.S. tesis, Dept. Lenguajes y sistemas informáticos, Ing, Universidad Rey Juan Carlos, Madrid, España, 2012

[5] I. Kerenky, "Enseñando y Aprendiendo Programación Orientada a Objetos en los primeros cursos de Programación: la experiencia en la Universidad ORT Uruguay," presentado en Universidad ORT Uruguay, Cuareim, Uruguay, 2005

[6] J. Cisneros, R. Collazo (2011), “Aprendiendo POO a través de la cultura indígena venezolana,” [Archivo], Disponible en http://dl.dropbox.com/u/18045712/videojuego_aprendiendo_poo.zip

[7] J. J. Perea Sánchez, " Sistema Tutor Web para el Aprendizaje de Programación Orientada a Objetos ", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2011.

[8] M. Tapia Téllez, " Aplicación para el apoyo a la enseñanza de la UEA Métodos Numéricos", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2010.

[9] J.C. Castillo García, "Aplicación Android para la práctica de verbos compuestos del idioma inglés", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2013.

[10] L. J. Aguilar, “Fundamentos de programación: Introducción a la ciencia de la computación y a la programación,” 3ra edición, McGraw-Hill, 2003.

[11] C. Larman, “Desarrollo iterativo y el proceso unificado,” en *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado*, 2da edición, Pearson Education, 2002, pp. 14.

[12] libGDX, “libGDX 1.0: wiki” [Online], Disponible en: https://github.com/libgdx/libgdx/wiki

[13] Java EE 7 SDK, “Java EE Downloads” [Online], Disponible en http://www.oracle.com/technetwork/java/javaee/downloads/index.html.

[14] MySQL Community Server, “MySQL Downloads” [Online], Disponible en http://dev.mysql.com/downloads/mysql/.

[15] Google Development Kit (GWT), “GWT homepage” [Online], Disponible en: http://www.gwtproject.org/

[16] Android Developer Tools (ADT), “ADT homepage” [Online], Disponible en: http://developer.android.com/tools/help/adt.html.

Anexo A: Listado del API del juego

En este anexo se presenta el listado del API del juego para apoyar a la comprensión del paradigma orientado a objetos. Para generar el API se utilizó *javadoc*.

Package com.ricardo.pi

Class Summary

POOGame

Clase principal de juego que define las pantallas a usar y carga todos los recursos que se usarán durante el juego.

---

com.ricardo.pi

Class POOGame

```
java.lang.Object
    |
    +--com.badlogic.gdx.Game
         |
         +--com.ricardo.pi.POOGame
```

All Implemented Interfaces:
com.badlogic.gdx.ApplicationListener

< Fields > < Constructors > < Methods >

---

public class **POOGame**
extends com.badlogic.gdx.Game

Clase principal de juego que define las pantallas a usar y carga todos los recursos que se usarán durante el juego.

Author:
Ricardo Rodríguez Nogal

Fields

FINDCODIGO

`public AbstractScreen FINDCODIGO`

LOADING

`public AbstractScreen LOADING`

LOGUIN

`public AbstractScreen LOGUIN`

MANAGER

public static final com.badlogic.gdx.assets.AssetManager **MANAGER**

MEMORAMA

public AbstractScreen **MEMORAMA**

PRINCIPAL

public AbstractScreen **PRINCIPAL**

PUNTUACIONES

public AbstractScreen **PUNTUACIONES**

REGISTRAR

public AbstractScreen **REGISTRAR**

SB

public com.badlogic.gdx.graphics.g2d.SpriteBatch **SB**

SELDIFICULTAD

public AbstractScreen **SELDIFICULTAD**

SELNIVEL

public AbstractScreen **SELNIVEL**

SOPACODIGO

public AbstractScreen **SOPACODIGO**

---

botonesFlecha

public java.lang.String **botonesFlecha**

---

fondoActorLinea

public java.lang.String **fondoActorLinea**

---

fondoCarga

public java.lang.String **fondoCarga**

---

fondoJuego

public java.lang.String **fondoJuego**

---

fondoPrincipal

public java.lang.String **fondoPrincipal**

---

fondoSeleccion

public java.lang.String **fondoSeleccion**

---

fondoSopaCodigo

public java.lang.String **fondoSopaCodigo**

---

fondoTransparente

public java.lang.String **fondoTransparente**

---

fuenteBerlin18

public java.lang.String **fuenteBerlin18**

---

fuenteConsolas16

public java.lang.String **fuenteConsolas16**

---

fuenteConsolas18

public java.lang.String **fuenteConsolas18**

---

fuenteDefault

public java.lang.String **fuenteDefault**

---

fuenteKimberley24

public java.lang.String **fuenteKimberley24**

---

fuenteKimberley26

public java.lang.String **fuenteKimberley26**

---

fuenteKimberley32

public java.lang.String **fuenteKimberley32**

---

fuenteKimberley74

public java.lang.String **fuenteKimberley74**

---

musicaDeFondo

public java.lang.String **musicaDeFondo**

---

skinBotones

public java.lang.String **skinBotones**

---

skinGeneral

public java.lang.String **skinGeneral**

---

sonidoDeCarta

```
public java.lang.String sonidoDeCarta
```

---

texturasCartasImg

```
public java.lang.String texturasCartasImg
```

---

texturasCartasWrd

```
public java.lang.String texturasCartasWrd
```

Constructors

POOGame

```
public  POOGame()
```

Constructor principal que inicializa las distintas pantallas del juego.

Methods

create

```
public void create()
```

---

dispose

```
public void dispose()
```

Overrides:

dispose in class com.badlogic.gdx.Game

Package com.ricardo.pi.actors

Class Summary

ActorCarta

Clase que define una carta del memorama dentro del juego *Memoria*

ActorEjercicio

Clase que implementa un ejrcicio de tipo opción múltiple para el juego *Encuentra el código.*

ActorItem

Clase que contiene la definicón de un item invariablemente que sea una pregunta o respuesta para el juego *Encuentra el código*

ActorLineaCodigo

Clase que define una línea de código para el juego *Sopa de instrucciones*

ActorRespuesta

Clase que implementa los atributos y comportamientos para de una respuesta para el juego *Encuentra el código*.

---

com.ricardo.pi.actors

Class ActorCarta

```
java.lang.Object
    |
    +--com.badlogic.gdx.scenes.scene2d.Actor
        |
        +--com.ricardo.pi.actors.ActorCarta
```

< Constructors > < Methods >

---

public class **ActorCarta**
extends com.badlogic.gdx.scenes.scene2d.Actor

Clase que define una carta del memorama dentro del juego *Memoria*

Author:

Ricardo Rodríguez Nogal

Constructors

ActorCarta

```
public  ActorCarta()
```

Constructor que inicializa los atributos de la carta.

Methods

draw

```
public void draw(com.badlogic.gdx.graphics.g2d.Batch batch,
                 float parentAlpha)
```

Overrides:

draw in class com.badlogic.gdx.scenes.scene2d.Actor

---

getNumCarta

```
public int getNumCarta()
```

Regresa el número del identificador de la carta en el juego de *Memoria*

Returns:

numCarta

---

girarCarta

```
public void girarCarta()
```

Gira la carta si esta no se encuentra bloqueada..

---

isCartaBloqueada

```
public boolean isCartaBloqueada()
```

Returns:

Devuelve True si la carta se encuentra bloqueda o false en caso contrario.

---

isCartaGirada

```
public boolean isCartaGirada()
```

Returns:

regresa true si la carta ha sido volteada o false en caso contrario.

---

setCartaBloqueada

`public void` **`setCartaBloqueada`**`(boolean cartaBloqueada)`

Bloquea la carta para que no se pueda seleccionar.

Parameters:

cartaBloqueada - true para bloquearla, false para desbloquearla.

---

setCartaGirada

`public void` **`setCartaGirada`**`(boolean cartaGirada)`

Pone la carta en un estado girado para que no sepueda volver a girar mientras se encuentra en uso

Parameters:

cartaGirada -

---

setFrente

`public void` **`setFrente`**`(com.badlogic.gdx.graphics.g2d.Sprite frente)`

Establece la imagen que se mostrará al frente de la carta en el juego de *Memoria*.

Parameters:

frente - sprite que contiene la imagen del frente de la carta.

---

setNumCarta

`public void` **`setNumCarta`**`(int numCarta)`

Establece el número de carta para ser usado en el juego de *Memoria*

Parameters:

numCarta - número de la carta.

---

setReverso

`public void` **`setReverso`**`(com.badlogic.gdx.graphics.g2d.Sprite reverso)`

Establce la imagen que se mostrará al reverso de la carta en el juego de *Memoria*.

Parameters:

reverso - sprite que contiene la imagen del reverso de la carta.

---

com.ricardo.pi.actors

Class ActorEjercicio

```
java.lang.Object
    |
    +--com.badlogic.gdx.scenes.scene2d.Actor
        |
        +--com.ricardo.pi.actors.ActorEjercicio
```

< Constructors > < Methods >

---

public class **ActorEjercicio**
extends com.badlogic.gdx.scenes.scene2d.Actor

Clase que implementa un ejrcicio de tipo opción múltiple para el juego *Encuentra el código. el ejercicio consta de una pregunta y tres respuestas.*

Author:

Ricardo Rodriguez Nogal

Constructors

ActorEjercicio

`public` **`ActorEjercicio`**`()`

Constructor principal.

Methods

getIdEjercicio

`public java.lang.Integer` **`getIdEjercicio`**`()`

Returns:

Devuelve el identificador del ejercicio.

---

getListaRespuestas

`public com.badlogic.gdx.utils.Array` **`getListaRespuestas`**`()`

Regresa la lista de respuestas que conforman el ejercicio.

Returns:

listaRespuestas.

getPregunta

`public ActorItem getPregunta()`

Returns:

Devuelve la pregunta como un actor en el escenario.

---

setIdEjercicio

`public void setIdEjercicio(java.lang.Integer idEjercicio)`

Establece el identificador del ejercicio.

Parameters:

idEjercicio -

---

setListaRespuestas

`public void setListaRespuestas(com.badlogic.gdx.utils.Array listaRespuestas)`

Establece la lista de respuesta que conforman el ejercicio.

Parameters:

listaRespuestas -

---

setPregunta

`public void setPregunta(ActorItem pregunta)`

Establece el valor de la pregunta como un actor.

Parameters:

pregunta -

com.ricardo.pi.actors

Class ActorItem

```
java.lang.Object
    |
    +--com.badlogic.gdx.scenes.scene2d.Actor
        |
        +--com.ricardo.pi.actors.ActorItem
```

Direct Known Subclasses:
ActorRespuesta

< Constructors > < Methods >

---

public class **ActorItem**
extends com.badlogic.gdx.scenes.scene2d.Actor

Clase que contiene la definicón de un item invariablemente que sea una pregunta o respuesta

para el juego *Encuentra el código*

Author:
Ricardo Rodríguez Nogal

Constructors

ActorItem

```
public  ActorItem()
```

Constructor de la clase.

Methods

draw

```
public void draw(com.badlogic.gdx.graphics.g2d.Batch batch,
                 float parentAlpha)
```

Overrides:

draw in class com.badlogic.gdx.scenes.scene2d.Actor

---

getIdItem

```
public java.lang.Integer getIdItem()
```

Returns:

El identificador del item.

---

getImagen

```
public com.badlogic.gdx.graphics.g2d.Sprite getImagen()
```

Returns:

Regresa un Sprite con el contenido del item.

---

setIdItem

```
public void setIdItem(java.lang.Integer idItem)
```

Establece el identificado del item.

Parameters:

idItem -

---

setImagen

```
public void setImagen(com.badlogic.gdx.graphics.g2d.Sprite imagen)
```

Establece el contenid0o de la ímagen dentro del item.

Parameters:

imagen -

---

com.ricardo.pi.actors

Class ActorLineaCodigo

```
java.lang.Object
    |
    +--com.badlogic.gdx.scenes.scene2d.Actor
        |
        +--com.ricardo.pi.actors.ActorLineaCodigo
```

< Constructors > < Methods >

---

public class **ActorLineaCodigo**

extends com.badlogic.gdx.scenes.scene2d.Actor

Clase que define una línea de código para el juego *Sopa de instrucciones*

Author:

Ricardo Rodríguez Nogal

Constructors

ActorLineaCodigo

```
public  ActorLineaCodigo()
```

Methods

draw

```
public void draw(com.badlogic.gdx.graphics.g2d.Batch batch,
                 float parentAlpha)
```

Overrides:

draw in class com.badlogic.gdx.scenes.scene2d.Actor

---

getId

```
public int getId()
```

Returns:

regresa el identificado de la línea de código.

---

getLineaCodigo

```
public java.lang.String getLineaCodigo()
```

Returns:

Regresa el contenido de la línea de código.

---

setId

`public void **setId**(int id)`

Asigna el identificador de la línea de código.

Parameters:

id - int identificador de la línea de código.

---

setLineaCodigo

`public void **setLineaCodigo**(java.lang.String lineaCodigo)`

Asigna una línea de código a un String

Parameters:

lineaCodigo - String línea de código

---

com.ricardo.pi.actors

Class ActorRespuesta

```
java.lang.Object
    |
    +--com.badlogic.gdx.scenes.scene2d.Actor
        |
        +--ActorItem
            |
            +--com.ricardo.pi.actors.ActorRespuesta
```

< Constructors > < Methods >

---

public class **ActorRespuesta**
extends ActorItem

Clase que implementa los atributos y comportamientos para de una respuesta para el juego *Encuentra el código*.

Author:

Ricardo Rodríguez Nogal.

Constructors

ActorRespuesta

`public **ActorRespuesta**()`

Methods

isRespuestaCorrecta

`public boolean` **`isRespuestaCorrecta`**`()`

Verifica que el valor de *respuestaCorrecta* sea true.

Returns:

true si es correcto **false** en caso contrario.

---

isRespuestaSeleccionada

`public boolean` **`isRespuestaSeleccionada`**`()`

Verifica si la respuesta ha sido seleccionada.

Returns:

true si es correcto **false** en caso contrario.

---

setRespuestaCorrecta

`public void` **`setRespuestaCorrecta`**`(boolean respuestaCorrecta)`

stablece el valor de *respuestaCorrecta*.

Parameters:

respuestaCorrecta -

---

setRespuestaSeleccionada

`public void` **`setRespuestaSeleccionada`**`(boolean respuestaSeleccionada)`

Establece el valor de *respuestaSeleccionada*

Parameters:

respuestaSeleccionada -

Package com.ricardo.pi.common

Class Summary

ConexionBD

Clase que se encarga de hacer un enlace con la base de datos.

ContadorTiempo

Clase que controla el timer para los distintos juegos de POO Game

CuadroDialogo

Crea un cuadro de diálogo personalizado para usar en el juego.

ManejadorArchivos

Clase encargada de manipular los recursos externos de la aplicación

Puntuacion

Se encarga de manejar las puntuaciones del juego

SetDificultad

Se encarga de almacenar los datos de juego de una manera persistente durante todo el juego.

Usuario

Clase que contiene los datos del usuario para poder identificarlo.

ValidadorDeDatos

VentanaInstrucciones

Crea una ventana modal donde se muestran instrucciones de juego..

---

com.ricardo.pi.common

Class ConexionBD

```
java.lang.Object
    |
    +--com.ricardo.pi.common.ConexionBD
```

< Constructors > < Methods >

---

public class **ConexionBD**
extends java.lang.Object

Clase que se encarga de hacer un enlace con la base de datos.

Author:

Ricardo Rodríguez Nogal

Constructors

ConexionBD

```
public  ConexionBD()
```

Constructor que se encarga de inicializar los atributos de la clase.

Methods

contarNumeroFilas

```
public int contarNumeroFilas(java.sql.ResultSet resultado)
```

Cuenta el número de filas en el ResultSet.

Parameters:

resultado - Resultset que contiene el resultado de una consulta a la base de datos.

Returns:

número de filas en una consulta.

---

crearConexion

```
public java.lang.Boolean crearConexion()
```

Establece una conexión con la base de datos.

Returns:

true si se estableció la conexión, **false** en caso contrario.

---

desconectarBD

```
public void desconectarBD()
```

Desconecta de la base de datos.

---

ejecutarSQLSelect

```
public java.sql.ResultSet ejecutarSQLSelect(java.lang.String sql)
```

Ejecuta una consulta a la base de datos.

Parameters:

sql - String que contiene la consulta a la base de datos.

Returns:

un objeto ResultSet con el resultado de la consulta a la base de datos.

---

ejecutarSql

```
public java.lang.Boolean ejecutarSql(java.lang.String sql)
```

Ejecuta una operación de escritura a la base de datos.

Parameters:

sql - String que contiene la consulta a la base de datos.

Returns:

true si se ejecutó la operación, **false** en caso contrario.

---

getConexion

```
public java.sql.Connection getConexion()
```

Returns:

Regresa un objeto de conexión a la base de de datos.

---

getResultado

```
public java.sql.ResultSet getResultado()
```

Returns:

Regresa un ResultSet con el resultado de la consuta a la base de datos.

com.ricardo.pi.common

Class ContadorTiempo

```
java.lang.Object
    |
    +--com.ricardo.pi.common.ContadorTiempo
```

< Constructors > < Methods >

---

public class **ContadorTiempo**
extends java.lang.Object

Clase que controla el timer para los distintos juegos de POO Game

Author:

RRicardo Rodríguez Nogal

Constructors

ContadorTiempo

`public` **`ContadorTiempo`**`()`

Constructor que inicializa los atributos de la clase

Methods

getCadenaTiempo

`public java.lang.String` **`getCadenaTiempo`**`()`

Returns:

regresa la cadena con formato "MM:SS".

---

getSegundosTotales

`public int` **`getSegundosTotales`**`()`

Returns:

regresa los segundos transcurridos desde la última vez que el timer se puso en marcha.

---

isTimerIsOn

`public boolean **isTimerIsOn**()`

Returns:

true si se ha iniciado el timer, false en caso contrario.

---

reset

`public void **reset**()`

Se encarga de poner a cers el timer.

---

setCadenaTiempo

`public void **setCadenaTiempo**(java.lang.String cadenaTiempo)`

Establece el valor en la cadena con formato "MM:SS".

Parameters:

cadenaTiempo -

---

setSegundosTotales

`public void **setSegundosTotales**(int segundosTotales)`

Establece los segundos totales del timer

Parameters:

segundosTotales - contiene el núemro de segundos transcurridos.

---

setTimerIsOn

`public void **setTimerIsOn**(boolean timerIsOn)`

Activa o desactiva la bandera para iniciar el timer.

Parameters:

timerIsOn - Bandera de indica si el timer está en marcha o no.

---

start

```
public void start()
```

Se encarga de iniciar el contador de tiempo

---

stop

```
public void stop()
```

Se encarga de detener el timer.

---

com.ricardo.pi.common

Class CuadroDialogo

```
java.lang.Object
    |
    +--com.badlogic.gdx.scenes.scene2d.Actor
        |
        +--com.badlogic.gdx.scenes.scene2d.Group
            |
            +--com.badlogic.gdx.scenes.scene2d.ui.WidgetGroup
                |
                +--com.badlogic.gdx.scenes.scene2d.ui.Table
                    |
                    +--com.badlogic.gdx.scenes.scene2d.ui.Window
                        |
                        +--com.badlogic.gdx.scenes.scene2d.ui.Dialog
                            |
                            +--com.ricardo.pi.common.CuadroDialogo
```

All Implemented Interfaces:

com.badlogic.gdx.scenes.scene2d.utils.Cullable, com.badlogic.gdx.scenes.scene2d.utils.Layout

---

< Constructors > < Methods >

---

public class **CuadroDialogo**
extends com.badlogic.gdx.scenes.scene2d.ui.Dialog

Crea un cuadro de diálogo personalizado para usar en el juego.

Author:

Ricardo Rodríguez Nogal

Constructors

CuadroDialogo

```
public  CuadroDialogo(java.lang.String title)
```

Constructor que crea al cuadro de diálogo.

Parameters:

title - Título del cuadro de diálogo.

CuadroDialogo

```
public  CuadroDialogo(java.lang.String title,
                      com.badlogic.gdx.scenes.scene2d.ui.Skin skin)
```

Constructor que crea al cuadro de diálogo.

Parameters:

title - Título del cuadro de diálogo.
skin - Skin a usar por el cuadro de diálogo.

CuadroDialogo

```
public  CuadroDialogo(java.lang.String title,
                      com.badlogic.gdx.scenes.scene2d.ui.Skin skin,
                      java.lang.String windowStyleName)
```

Parameters:

title - Título del cuadro de diálogo.
skin - Skin a usar por el cuadro de diálogo.
windowStyleName - Estilo de ventana a usar por el cuadro de diálogo.

CuadroDialogo

```
public  CuadroDialogo(java.lang.String title,
                      com.badlogic.gdx.scenes.scene2d.ui.Window.WindowStyle
windowStyle)
```

Parameters:

title - Título del cuadro de diálogo.
windowStyle - Estilo de ventana a usar por el cuadro de diálogo.

Methods

button

```
public CuadroDialogo button(java.lang.String buttonText,
                            com.badlogic.gdx.scenes.scene2d.InputListener
listener)
```

Agrega un TextButton a la tabla de borones del cuadro de diálogo

Parameters:

listener - El InputListener que será añadido al botón
buttonText - El texto que contendrá el botón

---

inicializar

```
public void inicializar()
```

Define los márgenes, tamaños, posiciones y atributos del cuadro de diálogo.

---

text

```
public com.badlogic.gdx.scenes.scene2d.ui.Dialog text(java.lang.String text)
```

Overrides:

text in class com.badlogic.gdx.scenes.scene2d.ui.Dialog

---

com.ricardo.pi.common

Class ManejadorArchivos

```
java.lang.Object
    |
    +--com.ricardo.pi.common.ManejadorArchivos
```

< Constructors > < Methods >

---

public class **ManejadorArchivos**
extends java.lang.Object

Clase encargada de manipular los recursos externos de la aplicación

Author:

Ricardo Rodríguez Nogal

Constructors

ManejadorArchivos

```
public  ManejadorArchivos()
```

Methods

contarOcurrencias

```
public int contarOcurrencias(java.lang.String ocurrencia,
                             java.lang.String cadena)
```

Lee una cadena, cuanta las ocurrencias del String ocurrencia y genera espacios en blanco según el número de tabuladores

Parameters:

ocurrencia - String que contiene la ocurrencia a buscar en la cadena de texto.
cadena - Cadena en la cual se buscarán las ocurrencias.

Returns:

un entero equivalente al número de ourrencias obtenidas.

---

getListaArchivos

```
public com.badlogic.gdx.files.FileHandle[] getListaArchivos()
```

Obtiene una lista de archivs resultado de leer un directorio.

Returns:

Array de String con nombres de archivo.

---

getNumeroItems

```
public int getNumeroItems()
```

Regresa el número de archivos o diectorios.

Returns:

numeroItems.

---

leerDirectorio

```
public void leerDirectorio(java.lang.String ruta)
```

Lee el directorio y regresa una lista de árchivos en él

Parameters:

ruta - String que contiene una ruta para buscar archivs y directorios

---

setNumeroItems

```
public void setNumeroItems(int numeroItems)
```

Establece el número de items.

Parameters:

numeroItems -

---

com.ricardo.pi.common

Class Puntuacion

```
java.lang.Object
    |
    +--com.ricardo.pi.common.Puntuacion
```

< Constructors > < Methods >

---

public class **Puntuacion**
extends java.lang.Object

Se encarga de manejar las puntuaciones del juego

Author:

Ricardo Rodríguez Nogal

Constructors

Puntuacion

```
public  Puntuacion()
```

Inicializa los campos del objeto.

Methods

getCodigoDificultad

`public java.lang.String` **`getCodigoDificultad`**`()`

Obtiene el código de dificultad del juego.

Returns:

String con el código del juego

---

getCodigoJuego

`public java.lang.String` **`getCodigoJuego`**`()`

Obtiene el código del juego

Returns:

String con el código del juego.

---

getIdUsuario

`public java.lang.String` **`getIdUsuario`**`()`

Obtiene el ID del jugador.

Returns:

String con el ID del jugador.

---

getNivelPuntuaciones

`public java.lang.String` **`getNivelPuntuaciones`**`()`

Obtiene el nivel del juego.

Returns:

String con el nivel del juego.

---

getPuntosPuntuaciones

`public java.lang.String` **`getPuntosPuntuaciones`**`()`

Obtiene el puntaje del juego.

Returns:

String con el puntaje del juego.

---

getTiempoPuntuaciones

```
public java.lang.String getTiempoPuntuaciones()
```

Obtiene el tiempo del juego

Returns:

string con formato de tiempo

---

mostrarPuntuacion

```
public void mostrarPuntuacion(java.lang.String consultaSQL)
```

Mustra las puntuaciones del usuario

Parameters:

consultaSQL - String que contiene la consulta a la base de datos.

---

registrarPuntuacion

```
public void registrarPuntuacion()
```

Se encarga de registrar las puntuaciones que el jugador obtuvo en cada juego.

---

setCodigoDificultad

```
public void setCodigoDificultad(java.lang.String codigoDificultad)
```

Estabblece el código de dificultad del juego.

Parameters:

codigoDificultad - String de longitud de tres caracteres.

---

setCodigoJuego

```
public void setCodigoJuego(java.lang.String codigoJuego)
```

Establece el código del juego

Parameters:

codigoJuego - String de longitud de tres caracteres.

---

setIdUsuario

`public void setIdUsuario(java.lang.String idUsuario)`

establece la ID del jugador.

Parameters:

idUsuario - String de longitud de 9 caracteres y caracteres numéricos.

---

setNivelPuntuaciones

`public void setNivelPuntuaciones(java.lang.String nivelPuntuaciones)`

Establece la puntuacion del juego.

Parameters:

nivelPuntuaciones - String con la puntuación del juego.

---

setPuntosPuntuaciones

`public void setPuntosPuntuaciones(java.lang.String puntosPuntuaciones)`

Estable el puntaje del juego.

Parameters:

puntosPuntuaciones - String con caracteres numéricos.

---

setTiempoPuntuaciones

`public void setTiempoPuntuaciones(java.lang.String tiempoPuntuaciones)`

Establece el tiempo de la puntuación del juego.

Parameters:

tiempoPuntuaciones - String con formato "00:00"

---

com.ricardo.pi.common

Class SetDificultad

```
java.lang.Object
    |
    +--com.ricardo.pi.common.SetDificultad
```

< Fields > < Constructors >

---

public class **SetDificultad**
extends java.lang.Object

Se encarga de almacenar los datos de juego de una manera persistente durante todo el juego.

Author:
Ricardo Rodríguez Nogal

Fields

dificultad

`public static int` **`dificultad`**

---

juego

`public static java.lang.String` **`juego`**

---

nivel

`public static int` **`nivel`**

---

rutaRelativa

`public static java.lang.String` **`rutaRelativa`**

---

rutaSemiRelativa

`public static java.lang.String` **`rutaSemiRelativa`**

---

usuario

`public static java.lang.String` **`usuario`**

Constructors

SetDificultad

`public` **`SetDificultad`**`()`

---

com.ricardo.pi.common

Class Usuario

```
java.lang.Object
    |
    +--com.ricardo.pi.common.Usuario
```

< Constructors > < Methods >

---

public class **Usuario**
extends java.lang.Object

Clase que contiene los datos del usuario para poder identificarlo.

Author:

Ricardo Rodríguez Nogal

Constructors

Usuario

```
public  Usuario()
```

Constructor por defecto.

Usuario

```
public  Usuario(java.lang.String matricula,
                java.lang.String apellidoPaterno,
                java.lang.String apellidoMaterno,
                java.lang.String nombre,
                java.lang.String email,
                java.lang.String username,
                java.lang.String password)
```

Constructor que recibe los datos del usuario para guardarlos como atributos.

Parameters:

matricula - String que contiene el número de matrícula del usuario.
apellidoPaterno - String que contiene el apellido paterno del usuario.
apellidoMaterno - String que contiene el apellido materno del usuario.
nombre - String que contiene el nombre del usuario.
email - String que contiene el E-mail del usuario.
username - String que contiene el username del usuario.
password - String que contiene el password del usuario.

Methods

getApellidoMaterno

`public java.lang.String` **`getApellidoMaterno`**`()`

Returns:

regresa el Apellido Paterno del usuario.

---

getApellidoPaterno

`public java.lang.String` **`getApellidoPaterno`**`()`

Returns:

regresa el Apellido Materno deel usuario.

---

getEmail

`public java.lang.String` **`getEmail`**`()`

Returns:

regresa el E-mail del usuario.

---

getMatricula

`public java.lang.String` **`getMatricula`**`()`

Returns:

regresa la Matrícula del usuario.

---

getNombre

`public java.lang.String` **`getNombre`**`()`

Returns:

regresa el nombre del usuario.

---

getPassword

`public java.lang.String **getPassword**()`

Returns:

regresa el password del usuario.

---

getUsername

`public java.lang.String **getUsername**()`

Returns:

regresa el username del usuario.

---

setApellidoMaterno

`public void **setApellidoMaterno**(java.lang.String apellidoMaterno)`

Establece el Apellido Paterno del usuario.

Parameters:

apellidoMaterno - Apellido paterno del usuario.

---

setApellidoPaterno

`public void **setApellidoPaterno**(java.lang.String apellidoPaterno)`

Establece el Apellido Materno del usuario.

Parameters:

apellidoPaterno - Apellido materno del usuario.

---

setEmail

`public void **setEmail**(java.lang.String email)`

Establece el E-mail del usuario.

Parameters:

email - E-mail del usuario.

---

setMatricula

`public void **setMatricula**(java.lang.String matricula)`

Establecel la matrícula del usuario.

Parameters:

matricula - Matrícula del usuario.

---

setNombre

`public void **setNombre**(java.lang.String nombre)`

Establece el nombre del usuario.

Parameters:

nombre - Nombre del usuario.

---

setPassword

`public void **setPassword**(java.lang.String password)`

Establece el password del usuario.

Parameters:

password - String password del usuario.

---

setUsername

`public void **setUsername**(java.lang.String username)`

Establece el Username del usuario.

Parameters:

username - Username del usuario.

---

com.ricardo.pi.common

Class ValidadorDeDatos

```
java.lang.Object
    |
    +--com.ricardo.pi.common.ValidadorDeDatos
```

< Constructors > < Methods >

---

public class **ValidadorDeDatos**
extends java.lang.Object

Author:

Ricardo Rodríguez Nogal

Constructors

ValidadorDeDatos

`public` **`ValidadorDeDatos`**`()`

Constructor por defecto.

Methods

validarMail

`public static boolean` **`validarMail`**`(java.lang.String mail)`

Evalua si una cadena de texto enviada es una dirección de ***mail*** válida.

Parameters:

mail - - Es la cadena de texto a evaluar.

Returns:

true si la cadena de texto es válida, **false** en caso contrario

---

validarMatricula

`public static boolean` **`validarMatricula`**`(java.lang.String matricula)`

Evalua que la cadena enviada contenga sólo dígitos

Parameters:

matricula - - Es la cadena de texto a evaluar.

Returns:

si la cadena de texto es válida, **false** en caso contrario.

---

validarNombreApellido

`public static boolean` **`validarNombreApellido`**`(java.lang.String cadenaTexto)`

Evalua si una cadena de texto sólo contiene caracteres alfabéticos.

Parameters:

cadenaTexto - - Es la cadena de texto a evaluar.

Returns:

true si la cadena de texto es válida, **false** en caso contrario

---

validarNombreUsuario

`public static boolean` **`validarNombreUsuario`**`(java.lang.String nombreUsuario)`

Evalua si una cadena de texto sólo contiene caracteres alfanuméricos.

Parameters:

nombreUsuario - - Es la cadena de texto a evaluar.

Returns:

true si la cadena de texto es válida, **false** en caso contrario

---

validarPassword

`public static boolean` **`validarPassword`**`(java.lang.String password)`

Evalua que la cadena de texto enviada contenga al menos:

- una letra mayúscula
- una letra minúscula
- un dígito
- No permite espacios
- Permite los símbolos "_" y "-"

Parameters:

password - - Es la cadena de texto a evaluar.

Returns:

true si la cadena de texto es válida, **false** en caso contrario.

com.ricardo.pi.common

Class VentanaInstrucciones

```
java.lang.Object
    |
    +--com.badlogic.gdx.scenes.scene2d.Actor
        |
        +--com.badlogic.gdx.scenes.scene2d.Group
            |
            +--com.badlogic.gdx.scenes.scene2d.ui.WidgetGroup
                |
                +--com.badlogic.gdx.scenes.scene2d.ui.Table
                    |
                    +--com.badlogic.gdx.scenes.scene2d.ui.Window
                        |
                        +--com.ricardo.pi.common.VentanaInstrucciones
```

All Implemented Interfaces:

com.badlogic.gdx.scenes.scene2d.utils.Cullable, com.badlogic.gdx.scenes.scene2d.utils.Layout

---

< Constructors > < Methods >

---

public class **VentanaInstrucciones**
extends com.badlogic.gdx.scenes.scene2d.ui.Window

Crea una ventana modal donde se muestran instrucciones de juego..

Author:

Ricardo Rodríguez Nogal

Constructors

VentanaInstrucciones

```
public  VentanaInstrucciones(java.lang.String title,
                             com.badlogic.gdx.scenes.scene2d.ui.Skin skin,
                             java.lang.String mensajeInstrucciones,
                             int tipoBoton)
```

Methods

isRegresar

```
public boolean isRegresar()
```

Returns:

true si regresar = true false en caso contrario.

Package com.ricardo.pi.pantallas

Class Summary

AbstractScreen

PantallaEncuentraCodigo

Se ebcarga de gestionar el juego Encuentra el código

PantallaLoading

PantallaLoguin

Se encarga de gestionar la autenticación del usuario en el sistema utilizando su ID de usuario y su contraseña.

PantallaMemorama

PantallaPrincipal

PantallaPuntuaciones

Se encarga de mostrar las puntuaciones solicitadas por el usuario.

PantallaRegistro

Gestiona el registro del jugador en la base de datos.

PantallaSeleccionarDificultad

PantallaSeleccionarNivel

Se encarga de manejar la selección de niveles para cada juego.

PantallaSopaCodigo

Clase que maneja el juego *Sopa de instrucciones*

---

com.ricardo.pi.pantallas

Class AbstractScreen

```
java.lang.Object
    |
    +--com.ricardo.pi.pantallas.AbstractScreen
```

All Implemented Interfaces:

com.badlogic.gdx.Screen

Direct Known Subclasses:

PantallaEncuentraCodigo, PantallaLoading, PantallaLoguin, PantallaMemorama, PantallaPrincipal, PantallaPuntuaciones, PantallaRegistro, PantallaSeleccionarDificultad, PantallaSeleccionarNivel, PantallaSopaCodigo

---

< Constructors > < Methods >

public abstract class **AbstractScreen**
extends java.lang.Object
implements com.badlogic.gdx.Screen

Constructors

AbstractScreen

```
public  AbstractScreen(POOGame game)
```

Methods

dispose

```
public void dispose()
```

---

pause

```
public void pause()
```

---

resize

```
public void resize(int width,
                   int height)
```

---

resume

```
public void resume()
```

com.ricardo.pi.pantallas

Class PantallaEncuentraCodigo

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaEncuentraCodigo
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaEncuentraCodigo**
extends AbstractScreen

Se ebcarga de gestionar el juego Encuentra el código

Author:
Ricardo Rodríguez Nogal

Constructors

PantallaEncuentraCodigo

```
public  PantallaEncuentraCodigo(POOGame game)
```

Methods

crearVentanaResumen

```
public void crearVentanaResumen(java.lang.String msgResultado)
```

---

hide

```
public void hide()
```

---

render

```
public void render(float delta)
```

show

```
public void show()
```

Se encarga de inicializar y mostrar los objetos del juego.

---

com.ricardo.pi.pantallas

Class PantallaLoading

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaLoading
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaLoading**
extends AbstractScreen

Constructors

PantallaLoading

```
public  PantallaLoading(POOGame game)
```

Methods

hide

```
public void hide()
```

---

render

```
public void render(float delta)
```

---

show

```
public void show()
```

---

com.ricardo.pi.pantallas

Class PantallaLoguin

```
java.lang.Object
    |
    +--AbstractScreen
         |
         +--com.ricardo.pi.pantallas.PantallaLoguin
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaLoguin**
extends AbstractScreen

Se encarga de gestionar la autenticación del usuario en el sistema utilizando su ID de usuario y su contraseña.

Author:
Ricardo Rodríguez Nogal

Constructors

PantallaLoguin

```
public  PantallaLoguin(POOGame game)
```

Methods

hide

```
public void hide()
```

---

mostrarCuadroDialogo

```
public void mostrarCuadroDialogo(java.lang.String tipo,
                                 java.lang.String mensaje)
```

Muestra un cuadro de diálogo con información sobre la autenticación

Parameters:

tipo - ipo del cuadro de diálogo
mensaje - String con el mensaje de resultado de consulta

---

render

```
public void render(float delta)
```

---

show

```
public void show()
```

Se encarga de inicializar los objetos de la ventana y mostrarlos

---

verificarUsuario

```
public java.lang.Boolean verificarUsuario(Usuario usuario)
```

Verifica que el usuario no exista en la base de datos para poderlo registrar

Parameters:

usuario - Objeto que contiene los datos del usuario

Returns:

true si el usuario existe **false** en caso contrario

---

com.ricardo.pi.pantallas

Class PantallaMemorama

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaMemorama
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

public class **PantallaMemorama**
extends AbstractScreen

Constructors

PantallaMemorama

```
public  PantallaMemorama(POOGame game)
```

Methods

calcularPuntajes

```
public void calcularPuntajes()
```

comprobarPares

```
public void comprobarPares()
```

crearLetreros

```
public void crearLetreros()
```

crearVentanaResumen

```
public void crearVentanaResumen()
```

dispose

```
public void dispose()
```

Overrides:

dispose in class AbstractScreen

hide

```
public void hide()
```

pause

```
public void pause()
```

Overrides:

pause in class AbstractScreen

render

```
public void render(float delta)
```

resize

```
public void resize(int width,
                   int height)
```

Overrides:

resize in class AbstractScreen

resume

```
public void resume()
```

Overrides:

resume in class AbstractScreen

show

```
public void show()
```

spawnCartas

```
public void spawnCartas()
```

---

com.ricardo.pi.pantallas

Class PantallaPrincipal

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaPrincipal
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaPrincipal**
extends AbstractScreen

Constructors

PantallaPrincipal

```
public  PantallaPrincipal(POOGame game)
```

Methods

dispose

```
public void dispose()
```

Overrides:

dispose in class AbstractScreen

---

hide

```
public void hide()
```

---

pause

```
public void pause()
```

Overrides:

pause in class AbstractScreen

---

render

```
public void render(float delta)
```

---

resize

```
public void resize(int width,
                   int height)
```

Overrides:

resize in class AbstractScreen

---

resume

```
public void resume()
```

Overrides:

resume in class AbstractScreen

---

show

```
public void show()
```

---

com.ricardo.pi.pantallas

Class PantallaPuntuaciones

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaPuntuaciones
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaPuntuaciones**
extends AbstractScreen

Se encarga de mostrar las puntuaciones solicitadas por el usuario.

Author:
Ricardo Rodríguez Nogal

Constructors

PantallaPuntuaciones

```
public  PantallaPuntuaciones(POOGame game)
```

Methods

hide

```
public void hide()
```

---

obtenerPuntuaciones

```
public void obtenerPuntuaciones()
```

Consulta la base de datos en busca de las puntuaciones para cada juego. para cada juego muestra campos diferentes.

---

render

```
public void render(float delta)
```

---

show

```
public void show()
```

---

com.ricardo.pi.pantallas

Class PantallaRegistro

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaRegistro
```

All Implemented Interfaces:

com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaRegistro**
extends AbstractScreen

Gestiona el registro del jugador en la base de datos.

Author:

Ricardo Rodríguez Nogal

Constructors

PantallaRegistro

```
public  PantallaRegistro(POOGame game)
```

Constructor por defecto

Parameters:

game - Clase principal del juego

Methods

crearPantalla

```
public void crearPantalla()
```

Es llamado para crear los elementos que conforman el formulario de registro
así como de los valores iniciales de cada elemento

---

enviarRegistro

```
public void enviarRegistro()
```

Comprueba que los datos a enviar no se encuentren duplicados en la base de datos
y muestra un mensaje de acurdo al éxito o fallo de la operación.

---

hide

```
public void hide()
```

---

mostrarCuadroDialogo

```
public void mostrarCuadroDialogo(java.lang.String tipo,
                                 java.lang.String mensaje)
```

Muestra un cuadro de diálogo con información resferente al estado del registro del jugador.

Parameters:

tipo - contiene el tipo de cuadro de diálogo para mostrar.
mensaje - contiene el mensaje a mostrar en el cuadro de diálogo.

---

render

```
public void render(float delta)
```

---

show

```
public void show()
```

Se encarga de inicializar los objetos de la ventana y mostrarlos.

---

validarDatos

```
public void validarDatos()
```

Realiza una validación de cada campo del formulario a través de expresiones regulares y muestra en pantalla aquellos campos que después de la validación no contienen datos válidos.

---

com.ricardo.pi.pantallas

Class PantallaSeleccionarDificultad

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaSeleccionarDificultad
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaSeleccionarDificultad**
extends AbstractScreen

Constructors

PantallaSeleccionarDificultad

```
public  PantallaSeleccionarDificultad(POOGame game)
```

Methods

hide

```
public void hide()
```

---

render

```
public void render(float delta)
```

show

```
public void show()
```

---

com.ricardo.pi.pantallas

Class PantallaSeleccionarNivel

```
java.lang.Object
    |
    +--AbstractScreen
        |
        +--com.ricardo.pi.pantallas.PantallaSeleccionarNivel
```

All Implemented Interfaces:
com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaSeleccionarNivel**
extends AbstractScreen

Se encarga de manejar la selección de niveles para cada juego.

Author:
Ricardo Rodríguez Nogal

Constructors

PantallaSeleccionarNivel

```
public  PantallaSeleccionarNivel(POOGame game)
```

Methods

establecerDificultad

```
public void establecerDificultad()
```

Establece la ruta donde dbe buscar los ejercicios para cada juego.

---

hide

```
public void hide()
```

---

render

```
public void render(float delta)
```

---

show

```
public void show()
```

Inicializa los objetos del juego y los muestra.

---

com.ricardo.pi.pantallas

Class PantallaSopaCodigo

```
java.lang.Object
    |
    +--AbstractScreen
           |
           +--com.ricardo.pi.pantallas.PantallaSopaCodigo
```

All Implemented Interfaces:

com.badlogic.gdx.Screen

< Constructors > < Methods >

---

public class **PantallaSopaCodigo**
extends AbstractScreen

Clase que maneja el juego *Sopa de instrucciones*

Author:

Ricardo Rodríguez Nogal

Constructors

PantallaSopaCodigo

`public` **`PantallaSopaCodigo`**`(POOGame game)`

Constructor de principal

Parameters:

game -

Methods

comprobarRespuesta

`public void` **`comprobarRespuesta`**`()`

Comprueba si la lista desordenada se encuentra vacia, de ser así, activa el botón de enviar.

---

crearVentanaResumen

`public void` **`crearVentanaResumen`**`()`

Muestra un cuadro de diálogo mostrando el número de errores obtenidos.

---

hide

`public void` **`hide`**`()`

---

rellenarTablero

`public void` **`rellenarTablero`**`()`

Se encarga de rellenar el tablero con las listas de items ordenando los items al azar.

---

render

`public void` **`render`**`(float delta)`

---

show

```
public void show()
```

Crea los objetos en el escenario y los muestr.

INDEX

S

T

U

V