ºº 003EXxpópñpl90’`¡

Versión 2.0

98978543e21º

Manual

Grupo Trivial 3a

2015

Contenido

[Visión general 3](#_Toc416249111)

[Manual de usuario 3](#_Toc416249112)

[Usuario cliente 4](#_Toc416249113)

[Administrador 4](#_Toc416249114)

[Manual de sistema 4](#_Toc416249115)

[Parte parser 4](#_Toc416249116)

[Main 4](#_Toc416249117)

[Bussiness. Extractor 5](#_Toc416249118)

[MenuExtractor 5](#_Toc416249119)

[Extractor 5](#_Toc416249120)

[Business. Parser 6](#_Toc416249121)

[Parser 6](#_Toc416249122)

[ParserGIFT - ParserQTI- ParserXML 6](#_Toc416249123)

[ParserQTI 6](#_Toc416249124)

[Business. Preguntas 6](#_Toc416249125)

[Pregunta 6](#_Toc416249126)

[Serializador 7](#_Toc416249127)

[Serializador 7](#_Toc416249128)

[Serializador. JSON 7](#_Toc416249129)

[JsonSerialImpl 7](#_Toc416249130)

[Persistencia 7](#_Toc416249131)

[Connection 7](#_Toc416249132)

[Insert 7](#_Toc416249133)

[Parte juego 8](#_Toc416249134)

[Paquete business 8](#_Toc416249135)

[Paquete core 8](#_Toc416249136)

[Paquete game 8](#_Toc416249137)

[Clase IniciarJuegoAction 8](#_Toc416249138)

[Clase RegistrarseAction 8](#_Toc416249139)

[Clase ValidarseAction 8](#_Toc416249140)

[Clase Action 8](#_Toc416249141)

[Clase Partida 8](#_Toc416249142)

[Paquete Tablero 9](#_Toc416249143)

[Paquete colores 9](#_Toc416249144)

[Paquete jugadores 9](#_Toc416249145)

[Clase jugador 9](#_Toc416249146)

[Paquete nodos 9](#_Toc416249147)

[Clase NodoFinal 9](#_Toc416249148)

[Clase NodoQuesito 9](#_Toc416249149)

[Clase NodoNormal 9](#_Toc416249150)

[Paquete tableros 9](#_Toc416249151)

[Clase TableroCircular 9](#_Toc416249152)

[Clase TableroLineal 10](#_Toc416249153)

[Clase TableroRueda 10](#_Toc416249154)

[Paquete Util 10](#_Toc416249155)

[Clase LectorCircular 10](#_Toc416249156)

[Clase LectorFormaRueda 10](#_Toc416249157)

[Clase ContenedorDeNodos 10](#_Toc416249158)

[Clase PreguntasAleatorias 10](#_Toc416249159)

[Clase SelectorDeColores 10](#_Toc416249160)

[Paquete modelo 10](#_Toc416249161)

[Clase Pregunta 10](#_Toc416249162)

[Clase Usuario 10](#_Toc416249163)

[Paquete persistencia 10](#_Toc416249164)

[Clase Connection 10](#_Toc416249165)

[Clase Insert 10](#_Toc416249166)

[Paquete presentación 11](#_Toc416249167)

[Paquete game 11](#_Toc416249168)

[Clase TableroLinealEntity 11](#_Toc416249169)

[Clase CasillaEntity 11](#_Toc416249170)

[Clase TableroEntity 11](#_Toc416249171)

[Clase AssetsManager 11](#_Toc416249172)

[Clase DesktopLauncher 11](#_Toc416249173)

[Clase ScreenInicio 11](#_Toc416249174)

[Clase ScreenJuego 11](#_Toc416249175)

[Clase ScreenSelectDificultad 11](#_Toc416249176)

[Paquete login 11](#_Toc416249177)

[Clase FondoLogin 11](#_Toc416249178)

[Clase VentanaAdmin 11](#_Toc416249179)

[Clase VentanaLogin 11](#_Toc416249180)

[Clase VentanaRegistro 11](#_Toc416249181)

# Visión general

En este manual, se describe el funcionamiento del módulo, que permite a un usuario, registrarse en el juego, iniciar una partida de trivial seleccionando el número de jugadores con los que desea jugar y consultar sus estadísticas e información de los jugadores. Nos centraremos en dos tipos diferentes de visiones:

1. **A nivel de usuario**: esta parte de la aplicación está pensada para un usuario cliente o administrador haga uso de ella. Se explicará cómo se realiza la interacción con el juego.
2. **A nivel de sistema**: instrucciones para la utilización de este código por distintos desarrolladores. Gracias a éste se permitirá una lectura rápida que nos ayudará a entender el código y funcionamiento del proyecto para su utilización y modificación posterior. Se explicará detalladamente la funcionalidad de las diversas clases que conforman el proyecto. El sistema está formado por una serie de jerarquías de clases que cooperan entre sí para lograr el correcto funcionamiento de la aplicación.

# Manual de usuario

Descripción del funcionamiento de la interfaz que será utilizada por un usuario cliente, éste usuario podrá registrarse en la aplicación, validarse para poder participar en una partida de trivial y consultar sus estadísticas de las partidas ganadas, preguntas acertadas, etc… Además, también se explica el funcionamiento de las pantallas de administrador, para introducir las preguntas en la base de datos y poder cargarlas.

## Usuario cliente

## Administrador

El administrador, al iniciar la aplicación, solo tendrá acceso a una pantalla, en ella, tendrá una lista de opciones a realizar y se le pedirá que introduzca el número de la opción que desee:

* Introduciendo **0**: salir de la aplicación.
* Introduciendo **1**: nos pedirá que introduzcamos la ruta del fichero con las preguntas que queramos añadir a la base de datos. Éste fichero debe tener uno de los siguientes formatos:
  1. **GITF**
  2. **XML simple (QTI)**

Si la ruta del fichero es correcta, y el fichero no contiene ningún error, las preguntas se almacenarán en la base de datos.

* Introduciendo **2**: esta opción nos permite leer un fichero de manera automática. Nos pedirá que introduzcamos cada cuanto tiempo queremos que se actualicen los datos del fichero en la base de datos. Por último, nos pedirá que introduzcamos la ruta del fichero como en el caso anterior.
* Introduciendo **otra cosa**: nos mostrará un mensaje indicando que la opción elegida no es correcta.

# Manual de sistema

El sistema está implementado con un conjunto de paquetes que se irán explicando a continuación. Se utiliza el patrón factoría para comunicar las capas de la lógica de negocio y persistencia con la capa de presentación.

# Parte parser

### Main

Esta clase únicamente se encarga de lanzar la aplicación, en ella se crea una instancia de la clase Extractor y se ejecuta su método run.

## Bussiness. Extractor

### MenuExtractor

Esta clase se encarga de la interacción con el administrador de la aplicación, en este caso, el usuario. En ella mediante el constructor y otros métodos se imprimen las distintas opciones con las que cuenta el programa, dependiendo de la respuesta del usuario se realizará una tarea u otra.

**handleOptions:** método que se encarga de realizar una determinada acción dependiendo de la opción introducida por el usuario.

* Si se introduce 0, la aplicación finalizará.
* Si se introduce 1, se comenzará con la lectura y subida de datos de forma manual.
* Si se introduce 2, se realizará la lectura y subida de datos de forma automática cada cierto tiempo.
* En otro caso, no hará nada.

**handleTime:** este método se encarga de guardar el tiempo que el usuario introduce para realizará la lectura y subida de datos de forma automática.

**handleConnection:** este método según la opción elegida por el administrador, se conectará con la base para almacenar los datos o no.

### Extractor

Realiza la tarea de intermediario ya que se encarga de trasladarle al *parser* la información proporcionada por el usuario que este necesita para su correcto funcionamiento.

En su método principal, run, toma los datos que ha introducido el usuario de la aplicación a través de una instancia de la clase anteriormente descrita, entre ellos el nombre del fichero que va a ser *parseado*.

Para ello realiza una serie de invocaciones a instancias de clases que posteriormente serán descritas así como a métodos propios de la clase:

**initialize***,* lleva a cabo una serie de tareas:

* Selecciona el tipo de *parser* que va a ejecutarse en función del formato de fichero de entrada.
* Le solicita al *parser* seleccionado el mapa con los objetos Pregunta que ha creado a partir del fichero.
* Crea el fichero de salida en el formato JSON correspondiente.
* En función de la elección del usuario lleva a cabo la carga del fichero de salido en la base de datos.

**setParser :**se encarga de crear la instancia correcta del *parseador* en función del formato del fichero de entrada.

## Business. Parser

### Parser

Se trata de una clase abstracta que sirve como generalización para los diferentes *parser* que aparecen en la aplicación. Cuenta con un atributo, "**nombreArchivo**" que será heredado por todas sus clases hijas. Se declara la signatura del método abstracto "**getPregunta**" que sus clases hijas implementaran posteriormente.

### ParserGIFT - ParserQTI- ParserXML

Cuentan con una función principal, denominada **getPregunta** cuya función es generar un mapa de objetos Pregunta a partir del fichero especificado. Para ello realiza las siguientes acciones:

* Selecciona y carga el fichero adecuada para poder recorrer su contenido.
* Crea las diferentes partes de las cuales consta un objeto de tipo Pregunta, es decir, su identificador, la pregunta, el “*array*” con las posibles respuestas y la respuesta correctas.
* A continuación comienza a recorrer el archivo de entrada para ir identificando los atributos antes mencionados a partir de las especificaciones del formato seleccionado (Anexo1).
* Cada vez que se han identificado en el archivo los cuatro atributos necesarios se crea un objeto Pregunta y este es almacenado en un mapa de preguntas a través de su identificador.
* Finalmente el método retorna el mapa con todas las preguntas que aparecen en el fichero y cierra este.

### ParserQTI

Además de la función mencionada anteriormente, esta clase cuenta con un privado **getIDRespuestaCorrecta,** que pasándole por parámetro la lista de respuestas y el id de la pregunta y el “namespace” del “*xml”* , retorna el id de la respuestas correcta.

## Business. Preguntas

### Pregunta

Se trata de una clase, que representa una pregunta en un formato de datos intermedio. Cuenta con todos los atributos necesarios que forman una pregunta:

* *pregunta*
* *respuestaCorrecta*
* *respuestasIncorrectas*

Este último es de tipo “*array*”, ya que cada pregunta cuenta con varias respuestas incorrectas. Además cada uno de estos cuenta con sus métodos de acceso y modificación (“*getters*” y “*setters*”) correspondientes.

## Serializador

### Serializador

Se trata de una interfaz que se usa como puerta de entrada para la transformación del tipo de datos almacenado internamente al formato de salida especificado. Únicamente cuenta con la descripción del método **createFile** que se encargan de implementar sus clases hijas descritas posteriormente, se creará una clase hija por cada formato de salida que el sistema quiera soportar.

## Serializador. JSON

### JsonSerialImpl

Se trata de la clase concreta que implementa la interfaz anteriormente descrita, en este caso se encarga de transformar los datos internos al formato JSON.

Como ya se sabía esta clase cuenta con la implementación del método **createFile** el cual hace uso de la librería “*Gson*” para crear el fichero de salida. La librería recibe como parámetro los objetos Pregunta que se han extraído del fichero de entrada y los transforma correctamente al formato JSON por lo tanto únicamente queda ir añadiendo los resultados que nos proporciona la librería al fichero deseado que es lo que se realiza al final de dicho método.

## Persistencia

### Connection

Clase que se encarga de conectar la aplicación con la base de datos en la que se desea trabajar, en este caso utilizaremos el sistema de [*base de datos*](http://es.wikipedia.org/wiki/Base_de_datos) *“[NoSQL](http://es.wikipedia.org/wiki/NoSQL)”*, “**mongoDB”**.

### Insert

Esta clase se encarga de la inserción en la base de datos del fichero JSON creado por el *Serializador*. Cuenta con una función principal que se encarga de invocar a los métodos necesarios para la incorporación de las preguntas del fichero a la base de datos. Estos métodos son:

* **importJSON**: que se encarga de cargar el fichero generado en la salida para poder leer su contenido.

**Insert:** se encarga de ir leyendo las diferentes líneas del fichero JSON y mediante la transformación de objetos que permite el uso de la clase JSON.java se introduce cada pregunta en la base de datos. Como las operaciones de lectura de fichero pueden causar problemas se tienen en cuenta varios errores que pueden ocurrir durante el procesamiento del archivo.

# Parte juego

## Paquete business

En este paquete se almacena toda la lógica del juego, está subdividido en otros tres paquetes.

### Paquete core

Este paquete contiene la clase Core que es la encargada crear y ejecutar la ventana principal de la aplicación en la que el usuario podrá realizar el registro y comenzar el juego.

### Paquete game

En este paquete se encuentran las clases encargadas de iniciar el juego.

#### Clase IniciarJuegoAction

Esta clase tiene una lista en la que se guardan los usuarios que van a participar en la partida junto con el color. Métodos:

* IniciarJuego: almacena en la lista anterior los usuarios que van a participar.
* Execute: añade la lista de usuarios a la base de datos.

#### Clase RegistrarseAction

Métodos:

* RegistrarseAction: crea un nuevo Usuario con los datos pasados como parámetros (Login, contraseña, nombre, apellidos, email y edad).
* ExistUserNameEmail: comprueba que en la base de datos no esté ningún usuario con el mismo nombre de usuario, email o nombre.
* Execute: añade a la base de datos el nuevo usuario.

#### Clase ValidarseAction

Métodos:

* GetUsuario: este método obtiene el usuario que se está validando a través de los datos que se le pasan: login y contraseña.
* UsuarioDuplicado: se comprueba que un jugador no se pueda validar dos veces en el juego para que no se repita ninguno en el momento de iniciar la partida.
* IsCorrecto: muestra mensajes de error si el usuario ya se ha validado, si el color ya se ha elegido o si la contraseña o login son incorrectos.

#### Clase Action

Es la encargada de realizar la conexión con la base de datos y obtener los datos alamacenados.

#### Clase Partida

Esta clase tiene los métodos encargados de añadir usuarios a una lista y de eliminarlos.

### Paquete Tablero

Contiene los diferentes paquetes:

#### Paquete colores

Este paquete tiene la interface color que puede ser amarillo, azul, incoloro, morado, naranja, rojo o verde.

#### Paquete jugadores

Este paquete contiene otro patrón factoría que se utiliza para crear nuevos jugadores.

##### Clase jugador

Tiene los atributos:

* Quesitos: lista de quesitos donde se añadirán los colores de los quesitos que se obtengan.
* Actual: nodo con la posición del tablero en la que se encuentra.
* Tablero: atributo de la clase Tablero.

Métodos:

* MoverDerecha/MoverIzquierda: se le pasa el número de posiciones que se quiere mover y se actualiza la posición del atributo “*actual*”, al nuevo nodo.
* AñadirQuesito: añade un color a la lista de quesitos.
* ComprobarQuesitos: comprueba si la lista quesitos contiene todos los colores y devuelve true o false.
* RemoveQuesito: quita un quesito de la lista.

#### Paquete nodos

Este paquete tiene una factoría para generar nodos, y las siguientes clases:

##### Clase NodoFinal

Será el último nodo del tablero.

##### Clase NodoQuesito

Serán los nodos en los que se puedan conseguir quesitos.

##### Clase NodoNormal

Serán el resto de nodos del tablero que tendrán un color y un nodo anterior y otro siguiente.

#### Paquete tableros

Éste paquete genera clases tableros que pueden ser circulares, lineales o de tipo rueda. Estos tableros contienen un nodo raíz que será el primero, un nodo pivote y un contenedor donde se encuentran los datos de todos los nodos que se van a enlazar.

##### Clase TableroCircular

El constructor se encarga de enlazar los nodos de los extremos para que el tablero esté en forma de círculo.

##### Clase TableroLineal

No se enlazan los nodos extremos ya que se ponen a null, de esta manera solo se podrá ir y volver en la misma línea.

##### Clase TableroRueda

El constructor se encarga de enlazar los nodos de los extremos para que el tablero esté en forma de círculo y además, los nodos con quesito se enlazan con nodos hacia arriba para que el círculo tenga sus radios.

#### Paquete Util

Este paquete tiene dos clases encargadas de cargar los nodos de los diferentes tipos de tableros.

###### Clase LectorCircular

Crea un contenedor de nodos y lo llena de quesitos.

###### Clase LectorFormaRueda

Carga un fichero que se le pasa como parámetro y va introduciendo cada nodo en el contenedor de nodos.

##### Clase ContenedorDeNodos

Clase que guarda en listas los nodos exteriores del tablero, los interiores y los nodos quesitos.

##### Clase PreguntasAleatorias

Crea una instancia de sí misma y tiene el método dado que obtiene un número aleatorio de 1 a 6.

##### Clase SelectorDeColores

Permite obtener uno de los colores del trivial pasándole un String con su nombre.

## Paquete modelo

Este paquete contiene las clases que se añadirán a la base de datos.

### Clase Pregunta

Tiene los atributos id, pregunta, respuestas incorrectas y respuesta correcta.

### Clase Usuario

Tiene los atributos login, contraseña, nombre, apellidos, email, edad, número de jugadas y número partidas ganadas.

## Paquete persistencia

Este paquete se encarga del acceso a la base de datos y de realizar las consultas para insertar datos. Esto se realiza mediante las clases:

#### Clase Connection

Conecta con la base de datos.

#### Clase Insert

Inserta elementos en la base de datos.

## Paquete presentación

Este paquete contiene dos sub paquetes donde se crea toda la parte visual del juego o interfaz.

#### Paquete game

##### Clase TableroLinealEntity

Esta clase se encarga de dibujar el tablero en una ventana mediante diferentes imágenes.

##### Clase CasillaEntity

Dibuja la imagen que se pintará en el nodo correspondiente y obtiene el siguiente nodo a dibujar.

##### Clase TableroEntity

Se encarga de llamar a las clases anteriores y a partir un nodo inicial, va dibujando todas las casillas.

##### Clase AssetsManager

Es la clase que carga toda la configuración de los assets ( fuentes, sonidos, atlas, skins...).

##### Clase DesktopLauncher

Contiene el main que dibuja la ventana principal.

##### Clase ScreenInicio

Esta clase se encarga de cargar la imagen de fondo de la ventana, de pintar todos los botones, posicionarlo todo en la ventana.

##### Clase ScreenJuego

Esta clase se encarga de dibujar el tablero con los diferentes usuarios que se validaran en el juego y mostrar las preguntas con las respuestas.

##### Clase ScreenSelectDificultad

Clase que muestra las diferentes opciones de tablero y permite seleccionar una de ellas (circular, lineal…).

#### Paquete login

##### Clase FondoLogin

Guarda la imagen de fondo de la ventana de login.

##### Clase VentanaAdmin

Carga la ventana del administrador con sus imágenes.

##### Clase VentanaLogin

Esta clase es la encargada de dibujar la ventana de login con diferentes paneles para registrarse o validarse, título, lista de jugadores, botón de inicio de juego, botón de registro, colores de jugadores, botón para ir a la ventana de registro.

##### Clase VentanaRegistro

Esta clase muestra una ventana con campos donde se le pide al usuario que introduzca los datos para realizar el registro.