PROJECTE PROGRAMACIÓ

KENKEN

GRUPO 14.2
VERSIÓN 2.0 - 27/05/2024

- DIAGRAMA DE CLASES Y DESCRIPCIÓN DE SUS MÉTODOS Y ATRIBUTOS

David Cañadas López
david.canadas

Raúl Gilabert Gámez
raul.gilabert

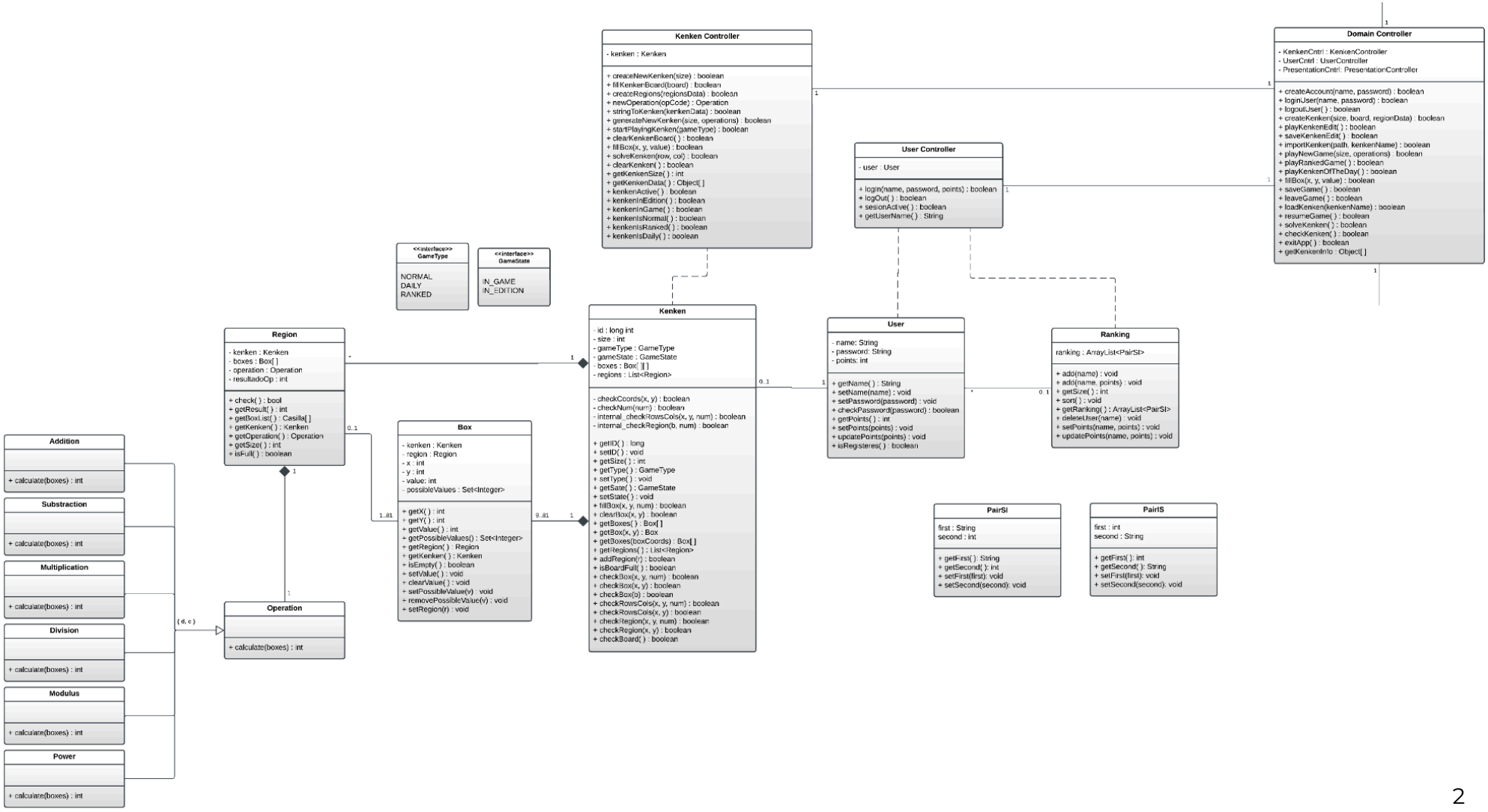
Guillem Nieto Ribó
guillem.nieto.ribo

Pau Zaragoza Gallardo
pau.zaragoza

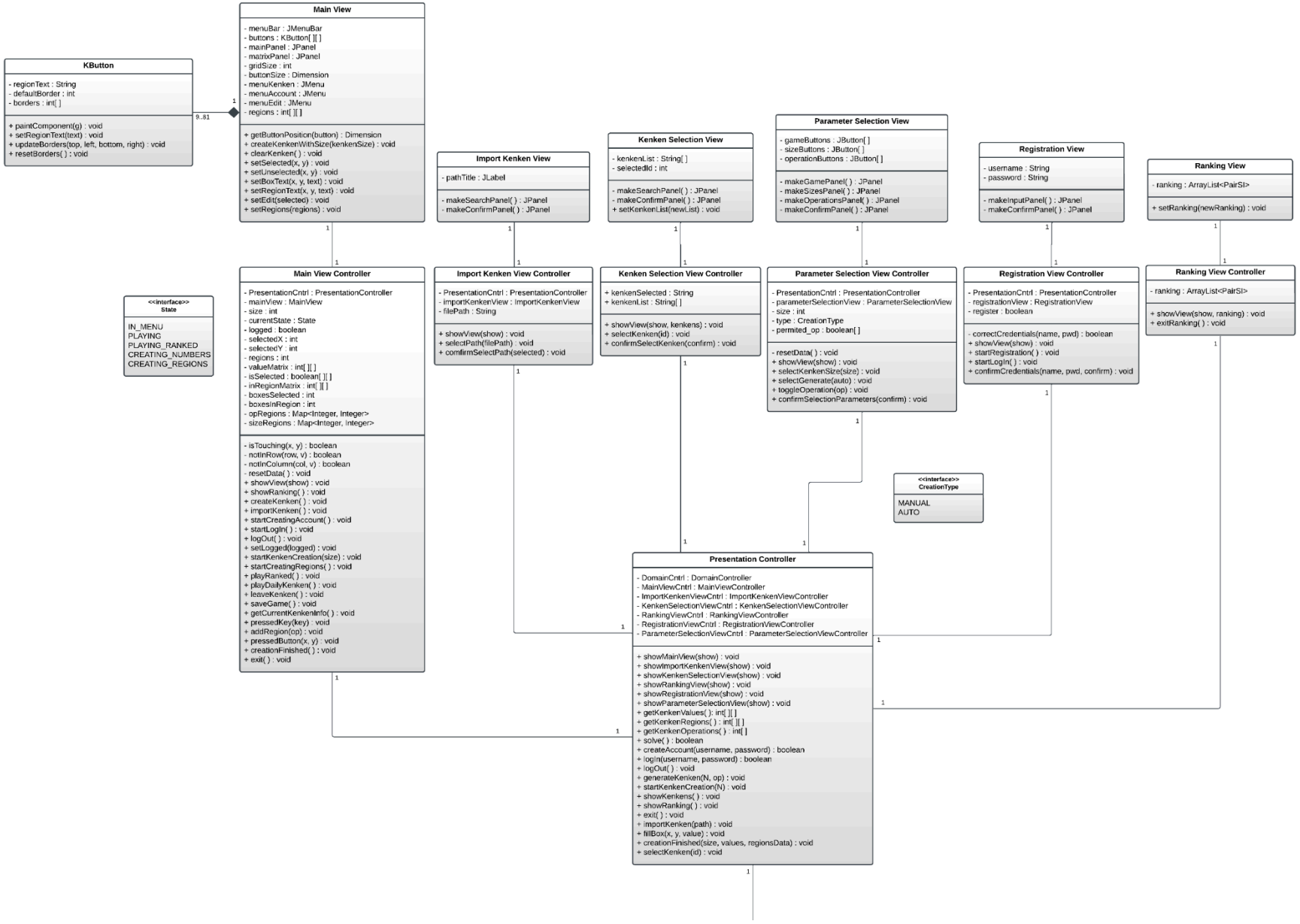
ÍNDICE

Diagrama de clases

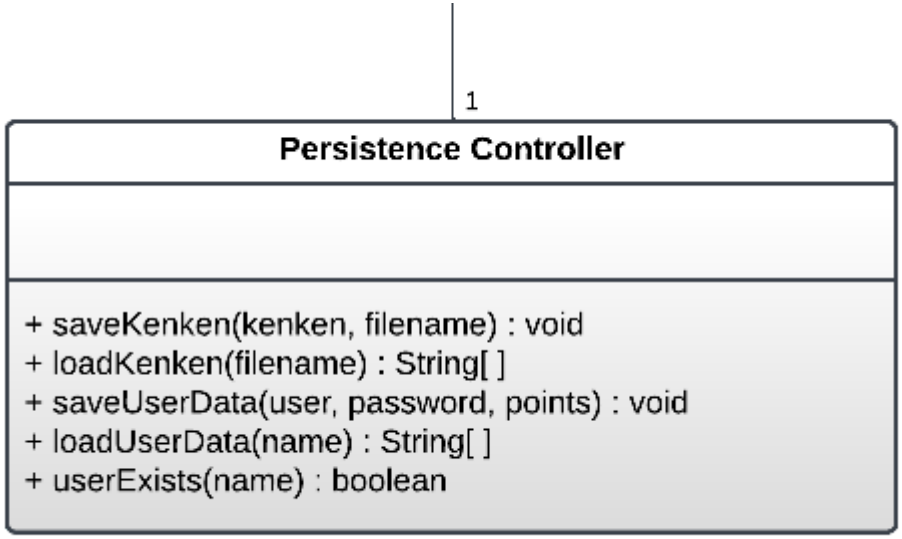
DOMINIO



PRESENTACIÓN



PERSISTENCIA



Asignación a cada miembro del grupo

PRESENTACIÓN:

- **PresentationController →** Guillem Nieto Ribó
- **MainViewController →** Guillem Nieto Ribó
- **ImportKenkenViewController →** Guillem Nieto Ribó
- **KenkenSelectionViewController →** Guillem Nieto Ribó
- **ParameterSelectionViewController →** Guillem Nieto Ribó
- **RankingViewController →** Guillem Nieto Ribó
- **RegistrationViewController →** Guillem Nieto Ribó

- **MainView →** David Cañadas López
- **ImportKenkenView →** David Cañadas López
- **KenkenSelectionView→** David Cañadas López
- **ParameterSelectionView →** David Cañadas López
- **RankingView →** David Cañadas López
- **RegistrationView →** David Cañadas López

DOMINIO:

- **DomainController →** Pau Zaragoza Gallardo
- **KenkenController →** Pau Zaragoza Gallardo
- **UserController →** Pau Zaragoza Gallardo

- **Kenken →** David Cañadas López
- **Region →** Guillem Nieto Ribó
- **Box →** Guillem Nieto Ribó
- **Operation (and all subclasses) →** Pau Zaragoza Gallardo
- **User →** Raúl Gilabert Gámez
- **Ranking →** Raúl Gilabert Gámez
- **PairSI →** Raúl Gilabert Gámez

PERSISTENCIA:

- **PersistenceController →** Raúl Gilabert Gámez

Descripción de clases

DOMINIO

Clase: **Kenken**

Atributos:

- `private long id`: número distintivo que identifica el Kenken.
- `private final int size`: número de filas y columnas del Kenken (2 < `size` < 10).
- `private GameType gameType`: el tipo de partida del Kenken.
- `private GameState gameState`: el estado actual de la partida del Kenken.
- `private final Box[][] boxes`: matriz con las casillas del Kenken.
- `private List<Region> regions`: lista con todas las regiones del Kenken.
- `public enum GameType`: enumeración de los posibles tipos de partida a jugar.
- `public enum GameState`: enumeración de los posibles estados de la partida.

Métodos:

- `public Kenken (int size)`

Crea una nueva instancia de Kenken con tamaño `size` y tipo de partida `NORMAL`.

- `public Kenken (int size, GameType gameType)`

Crea una nueva instancia de Kenken con tamaño `size` y tipo de partida `gameType`.

- `public Kenken (Box[][] boxes)`

Crea una nueva instancia de Kenken usando la matriz de casillas `boxes`.

- `private boolean checkCoords (int x, int y)`

Comprueba si las coordenadas horizontal y vertical de son válidas en este Kenken (0 < `x` < `size`, 0 < `y` < `size`).

- `private boolean checkNum (int num)`

Comprueba si el valor `num` es válido (0 < `num` < `size`).

- `public long getID ()`
- `public void setID (long id)`

Getter y setter del atributo `id`.

- `public int getSize ()`

Getter del atributo `size`.

- `public GameType getType ()`
- `public void setType (GameType gameType)`

Getter y setter del atributo `gameType`.

- `public GameState getState ()`
- `public void setState (GameState gameState)`

Getter y setter del atributo `gameState`.

- `public boolean fillBox (int x, int y, int num)`

Asigna el valor num a la casilla de coordenadas (x, y).

- `public boolean clearBox (int x, int y)`

Elimina el valor que tuviera la casilla de coordenadas (x, y).

- `public Box getBox (int x, int y)`

Devuelve una referencia a la casilla de coordenadas (x, y).

- `public Box[] getBoxes ()`

Devuelve un array con todas las casillas del tablero.

- `public Box[] getBoxes (int[] boxCoords)`

Devuelve un array con todas las casillas del tablero cuyas coordenadas corresponden con las coordenadas de `boxCoords` (por parejas).

- `public List<Region> getRegions ()`

Devuelve una lista con todas las regiones del tablero.

- `public boolean addRegion (Region r)`

Añade la región que referencia `r` al tablero.

- `public boolean isBoardFull ()`

Comprueba si todas las casillas del tablero contienen un valor.

- `public boolean **checkBox** (int x, int y, int num)`

Comprueba si el valor `num` en la casilla de coordenadas (`x`, `y`) es válido en su fila, columna y región (si la hubiera).

- `public boolean **checkBox** (int x, int y)`

Comprueba si la casilla de coordenadas (`x`, `y`) tiene un valor válido en su fila, columna y región (si la hubiera).

- `public boolean **checkBox** (Box b)`

Comprueba si la casilla que referencia `b` tiene un valor válido en su fila, columna y región (si la hubiera).

- `private boolean **internal_checkRowsCols** (int x, int y, int num)`

Implementa el algoritmo que comprueba la fila y columna de la casilla de coordenadas (`x`, `y`) y de valor `num`.

- `public boolean **checkRowsCols** (int x, int y, int num)`

Comprueba si las coordenadas y el valor son válidos, luego relega el trabajo a `internal_checkRowsCols`.

- `public boolean **checkRowsCols** (int x, int y)`

Comprueba si las coordenadas son válidas, luego relega el trabajo a `internal_checkRowsCols`.

- `private boolean **internal_checkRegion** (Box b, int num)`

Implementa el algoritmo que comprueba el resultado de la región de la casilla de coordenadas (`x`, `y`) y de valor `num` a partir de los valores de las casillas que la componen.

- `public boolean **checkRegion** (int x, int y, int num)`

Comprueba si las coordenadas y el valor son válidos, luego relega el trabajo a `internal_checkRegion`.

- `public boolean **checkRegion** (int x, int y)`

Comprueba si las coordenadas son válidas, luego relega el trabajo a `internal_checkRegion`.

- `public boolean` **`checkBoard`** `()`

Implementa un algoritmo sencillo que revisa el tablero y verifica si el Kenken ha sido resuelto correctamente.

Clase: `Region`

Clase que representa una región de un kenken.

Atributos:

- `private Kenken` **`kenken`**: Kenken al que pertenece la region.
- `private Box[]` **`boxes`**: vector con las casillas de la region.
- `private Operation` **`operation`**: operación asociada a la región.
- `private int` **`resultOp`**: resultado de la operación realizada con las casillas de la región.

Métodos:

- `public` **`Region`** `(Kenken k, Box[] b, Operation op)`

Crea una nueva instancia de Region asociada a un Kenken `k`, con las casillas `b` y la operación `op`. En esta creadora se entiende que las casillas ya contienen un valor y `resultOp` se inicializa calculando el resultado de la operación sobre esos valores.

- `public` **`Region`** `(Kenken k, Box[] b, Operation op, int r)`

Crea una nueva instancia de Region asociada a un Kenken `k`, con las casillas `b`, la operación `op`, y resultado `r`.

- `public boolean` **`check`** `()`

Devuelve `true` si el calculo de la operación asignada a la región sobre sus casillas es igual a `resultOp`. En caso contrario devuelve `false`.

- `public int` **`getResult`** `()`

Getter del atributo `resultOp`.

- `public Box[]` **`getBoxList`** `()`

Getter del atributo `boxes`.

- `public Kenken` **`getKenken`** `()`

Getter del atributo `kenken`.

- `public` **`getOperation`** `()`

Getter del atributo `operation`.

- `public int` **`getSize`** `()`

Devuelve el tamaño de la región, es decir el tamaño del atributo `boxes`.

- `public boolean **isFull** ()`

Devuelve `true` si todas las casillas de la región contienen valor. En caso contrario devuelve `false`.

Clase: Box

Clase que representa una casilla de un kenken.

Atributos:

- `private Kenken` **`kenken`**: Kenken al que pertenece la casilla.
- `private Region` **`region`**: región a la que pertenece la casilla.
- `private int` **`x, y`**: coordenadas de la casilla en el tablero.
- `private int` **`value`**: valor que contiene la casilla.
- `private Set<Integer>` **`possibleValues`**: conjunto de posibles valores correctos. Usados durante la resolución manual.

Métodos:

- `public` **`Box`** `(Kenken k, int x, int y)`

Crea una nueva instancia de Box asociada a un Kenken `k`, y cuyas coordenadas son (`x`, `y`).

- `public` **`Box`** `(Kenken k, int x, int y, int v)`

Crea una nueva instancia de Box con valor inicial `v` asociada a un Kenken `k`, y cuyas coordenadas son (`x`, `y`).

- `public` **`Box`** `(Kenken k, int x, int y, int v)`

Crea una nueva instancia de Box con valor inicial `v` asociada a un Kenken `k`, y cuyas coordenadas son (`x`, `y`).

- `public int` **`getValue`** `()`
- `public boolean` **`setValue`** `(int v)`

Getter y setter del atributo `value`.

- `public boolean` **`clearValue`** `()`

Elimina el valor actual de la casilla.

- `public boolean` **`isEmpty`** `()`

Comprueba si la casilla no tiene un valor asignado.

- `public boolean` **`getRegion`** `()`
- `public boolean` **`setRegion`** `()`

Getter y setter del atributo `region`.

- `public boolean` **`getKenken`** `()`

Getter del atributo `kenken`.

- `public boolean` **`getPossibleValues`** `()`

Getter del atributo `possibleValues`.

- `public boolean` **`addPossibleValue`** `(int v)`

Añade el valor `v` al atributo `possibleValues`.

- `public boolean` **`removePossibleValue`** `(int v)`

Elimina el valor `v` del atributo `possibleValues`.

Clase: `Operation (y sus subclases)`

Atributos:

Métodos:

- `public` **`Operation`** `()`

Crea una nueva instancia de Operation. Al ser una clase abstracta no se puede crear un objeto Operation directamente, por lo que esta creadora se llamará desde una de sus subclases.

- `abstract public int` **`calculate`** `(Box[] boxes)`

Método abstracto que debe ser implementado por las clases que extiendan Operation. Define la lógica de la operación a realizar sobre el array de casillas.

Clase: `User`

Atributos:

- `private String` **`name`**: nombre del usuario que también es el identificador.
- `private String` **`password`**: contraseña del usuario.
- `private int` **`points`**: puntuación de los kenkens resueltos por el usuario.

Métodos:

- `public` **`User`** `()`

Crea una nueva instancia de con nombre y contraseña vacío y 0 puntos.

- `public` **`User`** `(String name)`

Crea una nueva instancia de con nombre `name`, contraseña vacía y 0 puntos.

- `public` **`User`** `(String name, String password)`

Crea una nueva instancia de con nombre `name`, contraseña `password` y 0 puntos.

- `public String` **`getName`** `()`

Devuelve el nombre del usuario.

- `public void` **`setName`** `(String name)`

Asigna `name` al nombre del usuario.

- `public void` **`setPassword`** `(String password)`

Asigna `password` a la contraseña del usuario.

- `public boolean` **`checkPassword`** `(String password)`

Comprueba que `password` sea igual que la contraseña almacenada.

- `public int` **`getPoints`** `()`

Devuelve los puntos del usuario.

- `public void` **`setPoints`** `(int points)`

Pone los puntos del usuario a `points`.

- `public void` **`updatePoints`** `(int points)`

Incrementa los puntos del usuario en `points` (si se pasa un número negativo se puede decrementar).

- `public boolean` **`isRegistered`** `()`

Comprueba que el usuario esté registrado mirando si tiene o no nombre de usuario asignado.

Clase: `Ranking`

Atributos:

- `private ArrayList<PairSI> ranking`: Ránking de usuarios.

Métodos:

- `public Ranking ()`

Crea una nueva instancia de Ranking con una lista vacía.

- `public void add (String name)`

Añade el jugador identificado por `name` con 0 puntos.

- `public void add (String name int points)`

Añade el jugador identificado por `name` con `points` puntos.

- `public int getSize ()`

Devuelve el tamaño del ranking.

- `public void sort ()`

Ordena los datos del ránking en base a los puntos de los jugadores de forma descendente.

- `public ArrayList<PairSI> getRanking ()`

Devuelve el ránking de usuarios.

- `public void deleteUser (String name)`

Elimina el jugador del ránking.

- `public void setPoints (String name int points)`

Asigna al jugador identificado por `name` con `points` puntos.

- `public void updateUser (String name int points)`

Incrementa los puntos del usuario identificado por `name` en `points` (si se pasa un número negativo se puede decrementar).

Clase: `PairSI`

Atributos:

- `private String first`: Primer elemento del pair.
- `private int second`: Segundo elemento del pair.

Métodos:

- `public PairSI (String first, int second)`

Crea una nueva instancia de PairSI con los elementos.

- `public getFirst ()`

Devuelve el primer elemento.

- `public getSecond ()`

Devuelve el segundo elemento.

- `public setFirst (String first)`

Asigna `first` al primer elemento.

- `public setSecond (int second)`

Asigna `second` al segundo elemento.

Clase: `DomainController`

Controlador de la capa de dominio.

Atributos:

- `private PersistenceController` **`PersistenceCntrl`**: Controlador para gestionar la persistencia de datos.
- `private UserController` **`UserCntrl`**: Controlador para gestionar los usuarios.
- `private KenkenController` **`KenkenCntrl`**: Controlador para gestionar los kenken.

Métodos:

- `public boolean` **`createAccount`** `(String name, String password)`

Crea una cuenta de usuario y inicia sesión. Este método comprueba que no existe un usuario con el mismo nombre. A continuación, guarda los datos de usuario en la capa de persistencia e inicia la sesión del usuario.

- `public boolean` **`loginUser`** `(String name, String password)`

Inicia sesión. Este método comprueba que existe el usuario con el mismo nombre. A continuación, carga los datos del usuario de la capa de persistencia, comprueba que la contraseña es correcta e inicia la sesión del usuario.

- `public boolean` **`logoutUser`** `()`

Cierra la sesión del usuario. Este método guarda los datos del usuario de la capa de persistencia y cierra la sesión del usuario.

- `public boolean` **`createKenken`**
  `(int size, int[][] board, String[] regionsData)`

Crea un kenken desde cero con la información que proporciona el usuario. Este método comprueba que hay una sesión activa y que los parámetros son correctos. A continuación, crea un kenken con el tamaño especificado, lo rellena con los valores deseados y crea las correspondientes regiones.

- `public boolean` **`playKenkenEdit`** `()`

Pone en juego el kenken que está en edición. Este método comprueba que hay un kenken en edición. A continuación, limpia los valores del tablero y inicializa la partida.

- `public boolean` **`saveKenkenEdit`** `(String kenkenName)`

Guarda el kenken en edición en la capa de persistencia. Este método comprueba que hay una sesión activa y que hay un kenken en edición. A continuación, limpia los valores del tablero, obtiene la información del kenken en edición y la guarda en la capa de persistencia.

- `public boolean **importKenken** (String path, String kenkenName)`

Importa el kenken a partir de un archivo y lo guarda en la persistencia. Este método comprueba que no hay un kenken activo. A continuación, obtiene la información del archivo seleccionado por el usuario y lo guarda en el sistema.

- `public boolean **playNewGame** (int size, boolean[] operations)`

Crea y pone en juego una partida con la información que proporciona el usuario. Este método comprueba que no hay un kenken activo. A continuación, genera un kenken, limpia los valores del tablero y inicializa la partida.

- `public boolean **playRankedGame** ()`

Crea y pone en juego una partida clasificatoria. Este método comprueba que no hay un kenken activo. A continuación, genera un kenken clasificatorio (9x9 y todas las operaciones válidas), limpia los valores del tablero e inicializa la partida.

- `public boolean **playKenkenOfTheDay** ()`

Crea y comienza a jugar el Kenken de Día. Este método comprueba que no hay un kenken activo. A continuación, genera el kenken del día, limpia los valores del tablero e inicializa la partida.

- `public boolean **fillBox** (int x, int y, int value)`

Rellena la casilla deseada con el valor proporcionado. Este método comprueba que hay un kenken activo. A continuación, rellena la casilla de las coordenadas (x, y) con el valor deseado y devuelve si hay conflictos con otras casillas.

- `public boolean **saveGame** ()`

Guarda el kenken en juego para continuar con la partida más adelante. Este método comprueba que hay un kenken en juego y que la partida no es clasificatoria. A continuación, obtiene la información del kenken en juego y la guarda en la capa de persistencia.

- `public boolean **leaveGame** ()`

Abandona la partida en juego. Este método comprueba que hay un kenken en juego. A continuación, resta puntos al usuario en caso de ser una partida clasificatoria y cierra el kenken.

- `public boolean **loadKenken** (String kenkenName)`

Carga y pone en juego un kenken guardado en el sistema. Este método comprueba que no hay un kenken activo. A continuación, carga el kenken de la capa de persistencia y lo pone en juego.

- `public boolean **solveKenken** ()`

Resuelve el kenken. Este método comprueba que hay una kenken en juego y que la partida es normal. A continuación, resuelve el kenken y devuelve si tiene solución.

- `public boolean **checkKenken** (int size, boolean[] operations)`

Comprueba si el kenken tiene solución. Este método comprueba que hay una kenken en juego. A continuación, devuelve si la solución del kenken es correcta.

- `public boolean **exitApp** ()`

Guarda lo necesario antes de que se cierre la aplicación. En caso de que haya un kenken en juego, guarda la partida. En caso de que haya una sesión activa la cierra.

- `public Object[] **getKenkenInfo** ()`

Devuelve la información del kenken actual. Devuelve la información del tablero, la topología de las regiones, las operaciones de cada región y su resultado.

Clase: `UserController`

Controlador de usuarios.

Atributos:

- `private User` **`user`**: Usuario actual.

Métodos:

- `public boolean` **`logIn`** `(String name, String password, int points)`

Inicia la sesión del usuario especificado. Este método crea un usuario registrado.

- `public boolean` **`logOut`** `()`

Cierra la sesión del usuario actual. Este método crea un usuario no registrado.

- `public boolean` **`sesionActive`** `()`

Devuelve si el usuario está registrado en el sistema.

- `public String` **`getUserName`** `()`

Devuelve el nombre del usuario actual.

Clase: `KenkenController`

Controlador de kenkens.

Atributos:

- `private Kenken` **`kenken`**: Kenken actual.

Métodos:

- `public boolean` **`createNewKenken`** `(int size)`

Crea un kenken. Este método crea kenken con el tamaño especificado y lo pone en modo edición.

- `public boolean` **`fillKenkenBoard`** `(int[][] board)`

Rellena cada casilla del kenken con el valor deseado. Este método rellena cada casilla del kenken con el valor deseado.

- `public boolean` **`createRegions`** `(String[] regionsData)`

Rellena el tablero del kenken. Este método decodifica la información de las regiones y va creando tales regiones y se las va asignando al kenken.

- `public Operation` **`newOperation`** `(int opCode)`

Devuelve una instancia de la operación deseada. Este método devuelve la operación con el código proporcionado como parámetro.

- `public boolean` **`stringToKenken`** `(String[] kenkenData)`

Crea un kenken a partir de un String. Este método decodifica la información del kenken y lo va creando. Además lo pone en modo edición.

- `public boolean` **`generateNewKenken`** `(int size, boolean[] operations)`

Genera un kenken de cierto tamaño y con ciertas operaciones permitidas.

- `public boolean` **`generateDailyKenken`** `()`

Genera el kenken del día.

- `public boolean` **`startPlayingKenken`** `(GameType gameType)`

Pone en juego el kenken actual. Este método asigna el tipo de partida al kenken y lo pone en juego.

- `public boolean` **`clearKenkenBoard`** `()`

Limpia el tablero del kenken actual. Este método limpia el valor de cada casilla del kenken.

- `public boolean` **`fillBox`** `(int x, int y, int value)`

Rellena la casilla deseada con el valor proporcionado. Este método rellena la casilla de las coordenadas (x, y) con el valor deseado y devuelve si hay conflictos con otras casillas. En caso de haber conflicto y no ser una partida clasificatoria no rellenará la casilla.

- `public boolean solveKenken (int row, int col)`

Resuelve el kenken actual de forma recursiva y devuelve si tiene solución. Este método comprueba cada valor posible en la casilla seleccionada, si es posible vuelve a llamarse a este método pero para la siguiente casilla, si no lo es lo borra y prueba el siguiente valor. Si ningún valor es una posible solución para la casilla devuelve false. Si en cambio es la última casilla y encuentra una solución, devuelve true.

- `public boolean clearKenken ()`

Borra el kenken actual. Este método comprueba cada valor posible en la casilla seleccionada, si es posible vuelve a llamarse a este método pero para la siguiente casilla, si no lo es lo borra y prueba el siguiente valor. Si ningún valor es una posible solución para la casilla devuelve false. Si en cambio es la última casilla y encuentra una solución, devuelve true.

- `public int getKenkenSize ()`

Devuelve el tamaño del kenken actual.

- `public Object[] getKenkenData ()`

Devuelve la información del kenken actual.

- `public boolean kenkenActive ()`

Devuelve si hay un kenken activo.

- `public boolean kenkenInEdition ()`

Devuelve si hay un kenken activo y en edición.

- `public boolean kenkenInGame ()`

Devuelve si hay un kenken activo y en juego.

- `public boolean kenkenIsNormal ()`

Devuelve si el kenken es de tipo normal.

- `public boolean kenkenIsRanked ()`

Devuelve si el kenken es de tipo clasificatorio.

- `public boolean kenkenIsDaily ()`

Devuelve si el kenken es el kenken del día.

PRESENTACIÓN

Clase: `PresentationController`
Encargado de instanciar y controlar el controlador de dominio y los controladores de las vistas. Hace de intermediario entre estos.

Clase: `ImportKenkenViewController`
Implementa la interfaz gráfica donde se selecciona un fichero que contiene los datos necesarios para generar un Kenken y poderlo jugar.

Clase: `KenkenSelectionViewController`
Es el intermediario entre la vista de selección de kenken y el controlador de presentación. Dispone de los métodos y estructuras de datos necesarias para las funciones relacionadas con la selección de un kenken ya creado para jugar y para la reanudación de una partida.

Clase: `MainViewController`
Es el intermediario entre la vista principal y el controlador de presentación. Dispone de los métodos y estructuras de datos necesarias para las funciones relacionadas con jugar y crear henkens y con el menú principal.

Clase: `ParameterSelectionViewController`
Es el intermediario entre la vista de seleccionar parámetros y el controlador de presentación. Dispone de los métodos y estructuras de datos necesarias para que el usuario pueda personalizar su creación de kenkens (creación manual o automática, tamaño, operaciones permitidas).

Clase: `RankingViewController`
Es el intermediario entre la vista que muestra el ranking y el controlador de presentación. Dispone de los métodos y estructuras de datos necesarias para que el usuario pueda consultar el ranking de jugadores.

Clase: `RegistrationViewController`
Es el intermediario entre la vista registro/inicio de sesión y el controlador de presentación. Dispone de los métodos y estructuras de datos necesarias para que un usuario pueda iniciar sesión o registrarse si no lo está.

Clase: `ImportKenkenView`
Implementa la interfaz gráfica donde se selecciona un fichero que contiene los datos necesarios para generar un Kenken y poderlo jugar.

Clase: `KenkenSelectionView`

Implementa la interfaz gráfica con la que se pueden seleccionar Kenken del sistema para poder jugarlos. También muestra la partida guardada en la cuenta del usuario, si la hubiera.

Clase: `MainView`

Implementa la interfaz gráfica con la que interactúa el usuario cuando se inicia el programa, y donde se crean y juegan los Kenkens. Tiene un menú donde se pueden seleccionar diferentes casos de uso de la aplicación.

Clase: `ParameterSelectionView`

Implementa la interfaz gráfica donde se pueden seleccionar distintos parámetros para la creación de un Kenken. También ofrece al usuario la elección de crear un Kenken manualmente o bien que lo genere el sistema.

Clase: `RankingView`

Implementa la interfaz gráfica donde se muestran las cuentas de usuarios con la mayor puntuación en partidas clasificatorias.

Clase: `RegistrationView`

Implementa la interfaz gráfica con la que el usuario puede crear una cuenta en el sistema, o bien iniciar sesión en una cuenta existente.

Clase: `KButton`

Extensión de la clase JButton. Representa una casilla del Kenken.

PERSISTENCIA

Clase: `PersistenceController`

Controlador para la persistencia de la aplicación.