PROJECTE PROGRAMACIÓ

KENKEN

GRUPO 14.2
VERSIÓN 1.0 - 22/04/2024

- DIAGRAMA DE CLASES Y DESCRIPCIÓN DE SUS MÉTODOS Y ATRIBUTOS

David Cañadas López
david.canadas

Raúl Gilabert Gámez
raul.gilabert

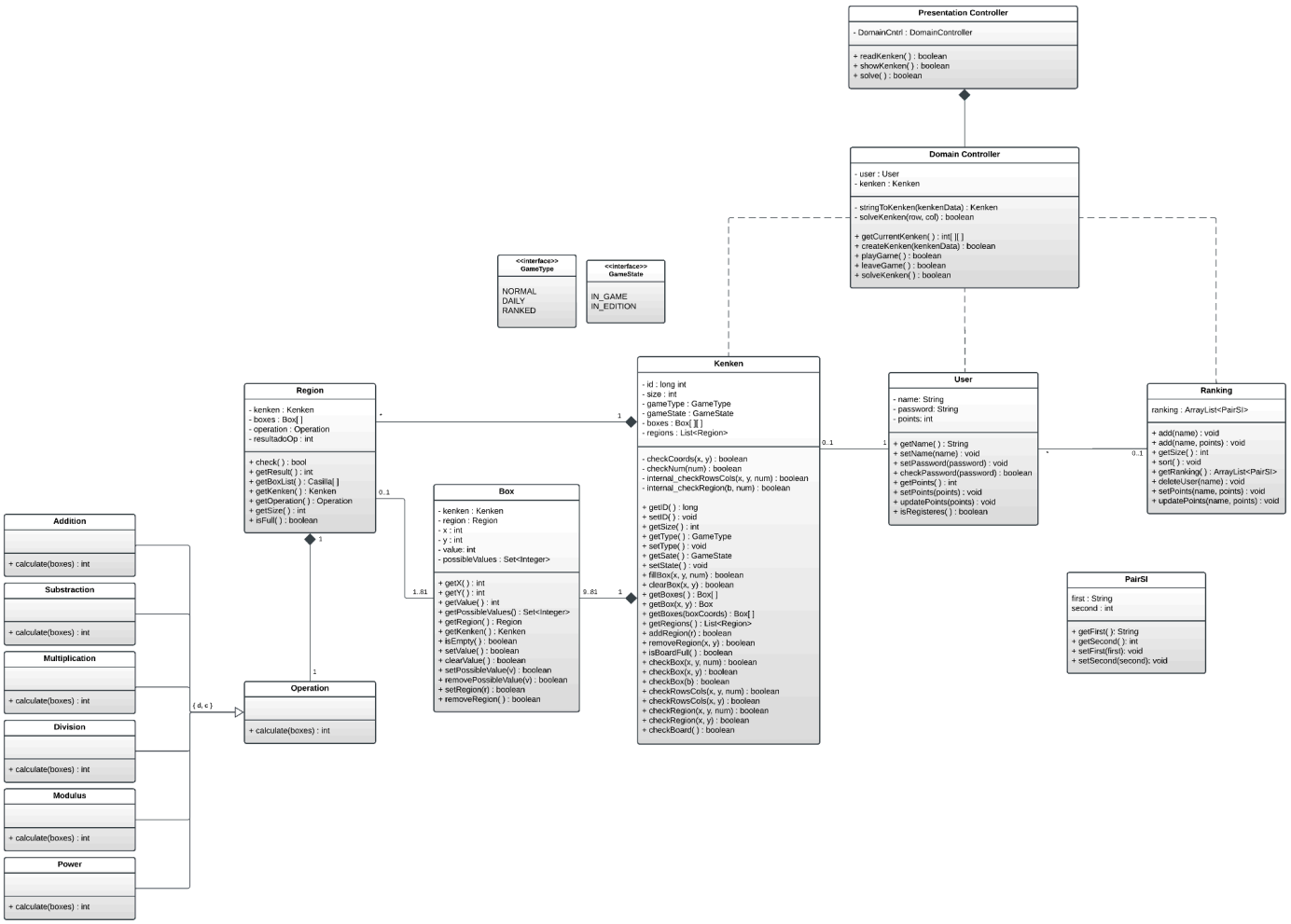
Guillem Nieto Ribó
guillem.nieto.ribo

Pau Zaragoza Gallardo
pau.zaragoza

ÍNDICE

Diagrama de clases.....2
Asignación a cada miembro del grupo.....2
Clase: Kenken.....3
Clase: Region.....7
Clase: Box.....9
Clase: Operation (y sus subclases).....11
Clase: User.....12
Clase: Ranking.....14
Clase: PairSI.....15
Clase: DomainController.....16
Clase: PresentationController.....17

Diagrama de clases



Asignación a cada miembro del grupo

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
PresentationController → Guillem Nieto Ribó + Raúl Gilabert Gámez
DomainController → Pau Zaragoza Gallardo
Driver (Main) → Raúl Gilabert Gámez
PairSI → Raúl Gilabert Gámez

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 removeRegion (int x, int y)`

Elimina del tablero la región a la que pertenece la casilla de coordenadas (x, y).

- `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**

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**

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` **`removeRegion`** `()`

Elimina la asociación a la región por parte de la casilla.

- `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)`

Calcula el resultado de la correspondiente operación. Como es un método abstracto el código de cada operación será independiente en cada subclase

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`**

Atributos:

- `private User` **`user`**: usuario actual en el programa.
- `private Kenken` **`kenken`**: Kenken activo en la partida.

Métodos:

- `public boolean` **`assignKenken`** `(Kenken kenken)`

Setter del atributo `kenken`. Utilizado principalmente para inyectar un kenken en los tests unitarios.

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

Devuelve una referencia a un nuevo Kenken de tamaño `size`.

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

Crea un nuevo Kenken, configura su tamaño y prepara sus casillas y regiones a partir del parámetro `kenkenData`. Devuelve la referencia al Kenken creado.

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

Crea un nuevo Kenken y lo configura a partir del parámetro `kenkenData`, incluyendo su estado y el tipo de partida.

- `public int[][]` **`getCurrentKenken`** `()`

Crea una matriz de números enteros compuesta por los valores de las casillas del Kenken actual.

- `public boolean` **`playGame`** `()`

Cambia el estado del Kenken actual (si lo hubiese) al estado en juego (`IN_GAME`).

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

Elimina el Kenken actual.

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

Comprueba que hay un Kenken activo y que está en el estado en juego. Relega el trabajo de resolverlo a `solveKenken(int row, int col)`.

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

Implementa el algoritmo recursivo que busca una solución para cada casilla vacía restante del Kenken.

Clase: **`PresentationController`**

Atributos:

- `private DomainController` **`DomainCntrl`**: El controlador de dominio del programa.

Métodos:

- `public` **`PresentationController`** `()`

Crea una nueva instancia de PresentationController.

- `public` **`PresentationController`** `(DomainController DomainCntrl)`

Crea una nueva instancia de PresentationController usando un DomainController ya existente.

- `public boolean` **`readKenken`** `()`

Lee por la entrada estándar el kenken en formato estándar introducido por el usuario y lanza la creadora del kenken. Retorna `true` si se ha creado correctamente, y `false` en caso contrario.

- `public boolean` **`showKenken`** `()`

Muestra por pantalla la solución del kenken activo en ese momento.

- `public boolean` **`solve`** `()`

Llama al DomainController para que solucione el kenken activo en ese momento. Si tiene solución lo imprime por pantalla usando el método `showKenken` y retorna `true`. Retorna `false` en caso contrario.