1 Enum Type

Enum représentant les différents types possibles de Case :

0. VIDE
1. BOMBE

— Méthodes :

1. **toString()** : Retourne la représentation d'un Type de Case, c'est-à-dire le caractère à afficher pour ce type. Devrait être un de :

   VIDE : Unicode 0x2B1C, «WHITE LARGE SQUARE»

   BOMBE : Unicode 0x1F4A3, «BOMB»

   — Retour : La représentation du Type de Case.
   — Type de retour : String

2 Enum Marque

Enum représentant les différentes marques possibles sur une Case :

0. VIDE
1. BOMBE
2. INCONNUE

— Méthodes :

1. **toString()** : Retourne la représentation d'une marque de Case, c'est-à-dire le caractère à afficher pour cette marque. Devrait être un de :

   VIDE : Unicode 0x2B1C, «WHITE LARGE SQUARE»

   BOMBE : Unicode 0x1F6A9, «TRIANGULAR FLAG ON POST»

   INCONNUE : « ? »

   — Retour : La représentation du Type de Case.
   — Type de retour : String

3 Classe Case

Une case du jeu Démineur.

La case est initialement cachée et non marquée. Lorsque la case est marquée, la marque passe successivement de «vide» à «bombe» puis à «inconnue».

— Propriétés :

1. **type** : Le type de case soit «vide» ou «bombe».
   — Type : Type
   — Accès : lecture seule
2. **marque** : La marque sur la case soit «vide», «bombe» ou «inconnue».
   — Type : Marque
   — Accès : lecture et écriture
3. **découverte** : Vrai si et seulement si la case est découverte.
   — Type : boolean
   — Accès : lecture et écriture

— Constructeurs :

1. **Case(Type)** : Constructeur paramétré.
   — Paramètres :
   (i) unType (Type) : Le type de Case
2. **Case()** : Constructeur sans paramètre. Initialise une case vide.

— Méthodes :

1. **découvrir()** : mutateur de la propriété *découverte*. La case ne pouvant pas être rechachée, cette méthode est à sens unique.
2. **marquer()** : mutateur de la propriété *marque*. À chaque appel de cette méthode, la marque passe successivement de «vide» à «bombe», à «inconnue» puis revient à «vide».
3. **estDécouverte()** : accesseur de la propriété *découverte*
   — Retour : Vrai si et seulement si la case est découverte.
   — Type de retour : boolean
4. **toString** : Retourne la représentation d'une case en chaîne de caractères. Si la case est découverte, il s'agit de la représentation de son type, sinon il s'agit de la représentation de sa marque.

— Retour : La représentation de la case en chaîne de caractères : son type si elle est découverte, sa marque sinon.
  — Type de retour : String

4 Classe Grille

Une grille de Démineur. La grille rectangulaire de longueur et largeur entre 5 et `Integer.MAX_VALUE` inclusivement est constituée de Cases initialement toutes cachées. Lorsque le joueur découvre une Case, toutes ses voisines «vides» sont aussi découvertes. Découvrir une Case sans bombes adjacentes découvre automatiquement toutes les voisines.

— Propriétés :

1. **largeur** : Le nombre de Cases horizontalement sur la grille.
   — Type : int
   — Accès : lecture seule
2. **hauteur** : Le nombre de Cases verticalement sur la grille.
   — Type : int
   — Accès : lecture seule

— Constructeurs :

1. **Grille(int, int)** : Constructeur paramètré. Initialise une Grille de dimensions données.
   — Paramètres :
   (i) uneLargeur (int) : La largeur de la nouvelle grille
   (ii) uneHauteur (int) : La hauteur de la nouvelle grille
   — Assertions :
   (i) La largeur et la hauteur doivent être comprises entre 5 et `Integer.MAX_VALUE` inclusivement.
   — Exception : IllegalArgumentException
   — Message : La grille doit avoir une largeur et une hauteurs comprises entre 5 et `Integer.MAX_VALUE` inclusivement.
   — **Grille()** : Constructeur sans paramètres. Initialise une Grille de 10x10.

— Méthodes :

1. **getFaceCase(int, int)** : Retourne la représentation d'une Case sur la Grille. Si la case est cachée, elle est représentée par sa marque, sinon, elle est représentée par l'une des options suivantes :
   — Son Type s'il s'agit d'une bombe
   — une espace si elle est vide et n'est adjacente à aucune bombe
   — Le nombre de bombes adjacentes (entre 1 et 8)

— Paramètres :
  (i) x (int) : La coordonnée horizontale de la case voulue
  (ii) y (int) : La coordonnée verticale de la case voulue
— Retour : La représentation de la case aux coordonnées (x,y)
  — Type de retour : String
— Assertions :
  (i) La case existe
    — Exception : IllegalArgumentException
    — Message : Les coordonnées spécifiées sont hors de la grille.

2. **compterVoisins(int, int)** : Retourne le nombre de voisins d'une case sur lesquels se trouvent une bombe
— Paramètres :
  (i) x (int) : La coordonnée horizontale de la case voulue
  (ii) y (int) : La coordonnée verticale de la case voulue
— Retour : Le nombre de cases voisine de type «bombe»
  — Type de retour : int
— Assertions :
  (i) La case existe
    — Exception : IllegalArgumentException
    — Message : Les coordonnées spécifiées sont hors de la grille.

3. **toString()** : Retourne une représentation en chaîne de caractères de la grille. Les bords de la grilles sont dessinées en caractères unicode «BOX DRAWING» et pour chaque case, sa représentation actuelle est données, selon qu'elle soit cachée ou marquée.
— Retour : représentation en chaîne de caractères de la Grille.
  — Type de retour : String

4. **initialiser(int, int, int)** : Constitue la grille de Cases aléatoirement, sachant que la case aux coordonnées (x,y) doit être vide.
— Paramètres :
  (i) x (int) : La coordonnée horizontale de la case vide
  (ii) y (int) : La coordonnée verticale de la case vide
  (iii) nbBombes (int) : Le nombre total de bombes à placer sur la Grille.

— Assertions :

(i) La case «vide» existe
- — Exception : IllegalArgumentException
- — Message : Les coordonnées spécifiées sont hors de la grille.

(ii) Le nombre de bombes doit être plus petit que $largeur \times hauteur$
- — Exception : IllegalArgumentException
- — Message : nbBombes doit être plus petit que le nombre de cases.

5. **getCase(int, int)** : Retourne la case aux coordonnées (x,y)
   — Paramètres :
   (i) x (int) : La coordonnée horizontale de la case voulue
   (ii) y (int) : La coordonnée verticale de la case voulue
   — Retour : La case aux coordonnées (x,y)
   — Type de retour : Case
   — Assertions :
   (i) La case existe
   — Exception : IllegalArgumentException
   — Message : Les coordonnées spécifiées sont hors de la grille.
6. **découvrir(int, int)** : découvre la case aux coordonnées données.
   — Paramètres :
   (i) x (int) : La coordonnée horizontale de la case voulue
   (ii) y (int) : La coordonnée verticale de la case voulue
   — Retour : Le Type de la case découverte.
   — Type de retour : Type
   — Assertions :
   (i) La case existe
   — Exception : IllegalArgumentException
   — Message : Les coordonnées spécifiées sont hors de la grille.
7. **marquer(int, int)** : marque la case aux coordonnées données.
   — Paramètres :

(i) x (int) : La coordonnée horizontale de la case voulue

(ii) y (int) : La coordonnée verticale de la case voulue

— Retour : La nouvelle marque de la case.
  — Type de retour : Marque
— Assertions :

(i) La case existe
  — Exception : IllegalArgumentException
  — Message : Les coordonnées spécifiées sont hors de la grille.

8. **toutRévéler()** : découvre toutes les cases.
9. **estRéussi()** : vérifie que le jeu est réussi ou non.
   — Retour : Vrai si et seulement si toutes les cases et uniquement les cases de type «vide» sont découvertes.
     — Type de retour : boolean

5 Classe Joueur

Un joueur de démineur. Il est identifié de façon unique par son pseudonyme.

Le niveau d'un joueur est donné par le nombre de points accumulé en gagnant des parties (indépendamment du nombre de parties total jouées). Selon le niveau de difficulté d'une partie, le joueur reçoit un certain nombre de points de la façon suivante :

1. Facile : 1 point
2. Moyen : 4 points
3. Difficile : 10 points

— Propriétés :

1. nom : Le nom complet du joueur
   — Type : String
   — Accès : lecture seule
2. pseudonyme : Le pseudonyme du joueur
   — Type : String
   — Accès : lecture seule
3. niveau : un niveau numérique représentant la qualité du joueur.
   — Type : int
   — Accès : lecture seule
4. parties : La liste des parties jouées par le joueur
   — Type : ArrayList<Partie>
   — Accès : lecture seule

— Constructeurs :

1. Joueur : Constructeur complet du joueur. Utile pour construire un joueur en précisant toutes ses caractéristiques.
   — Paramètres :
   (i) unNom (String) : Le nom du Joueur
   (ii) unPseudonyme (String) : Le pseudonyme du Joueur
   (iii) unNiveau (int) : Le niveau du Joueur
   (iv) desParties (ArrayList<Partie>) : La liste des parties du Joueur
   — Assertions :

(i) unNom ne peut pas être null ou une chaîne vide.

(ii) unPseudonyme ne peut pas être null ou une chaîne vide.

(iii) unNom ne peut pas être < 0.

(iv) desParties ne peut pas être null.

2. Joueur : Constructeur avec valeurs par défaut. Utile pour un joueur qui vient de s'inscrire. Le Joueur est créé avec son nom et son pseudonyme fournis, un niveau à 0 et une liste de parties vide.
   — Paramètres :
      (i) unNom (String) : Le nom du Joueur
      (ii) unPseudonyme (String) : Le pseudonyme du Joueur
   — Assertions :
      (i) unNom ne peut pas être null ou une chaîne vide.
      (ii) unPseudonyme ne peut pas être null ou une chaîne vide.

— Méthodes :

1. ajouterPartie : ajoute une partie à la liste des parties
   — Paramètres :
      (i) unePartie (Partie) : La partie à ajouter
   — Assertions :
      (i) La partie n'est pas nulle

6 Classe JoueurDAO

La classe d'accès aux données du Joueur.
— Hérite de : DAO<Joueur>
— Méthodes :

1. lire : instancie un Joueur à partir des informations provenant de la source de données.
   — Paramètres :
      (i) identifiant (Object) : Le pseudonyme du Joueur
   — Retour : Le Joueur provenant de la source de données ou null s'il n'a pas été trouvé.
      — Type de retour : Joueur

— Assertions :

(i) identifiant est une chaîne de caractère non nulle et non vide.

— Lance :

(i) DAOException : si un problème survient lors de la lecture.

2. créer : ajoute un nouveau Joueur dans la source de données.
   — Paramètres :

   (i) unJoueur (Joueur) : Le Joueur à ajouter

   — Retour : Le Joueur ajouté à la source de données tel qu'il peut avoir été modifié lors de l'ajout.
      — Type de retour : Joueur
   — Assertions :

   (i) unJoueur est non null

   — Lance :

   (i) DAOException : si un problème survient lors de la création.

3. modifier : modifie un Joueur dans la source de données.
   — Paramètres :

   (i) unJoueur (Joueur) : Le Joueur tel qu'il doit devenir dans la source de données.

   — Retour : Le Joueur après modification dans la source de données.
      — Type de retour : Joueur
   — Assertions :

   (i) unJoueur est non null

   — Lance :

   (i) DAOException : si un problème survient lors de la modification.

4. supprimer : supprime un Joueur de la source de données.
   — Paramètres :

   (i) unJoueur (Joueur) : Le Joueur à supprimer

   — Assertions :

   (i) unJoueur est non null

   — Lance :

(i) DAOException : si un problème survient lors de la modification.

5. trouverTout : Recherche tous les joueurs à partir de la source de données
   - Retour : Une liste de Joueurs représentant tous les joueurs trouvés dans la source de données
     - Type de retour : ArrayList<Joueur>
   - Lance :
     (i) DAOException : si un problème survient lors de la modification.
6. trouverParNiveau : Recherche tous les joueurs d'un certain niveau dans la source de données.
   - Paramètres :
     (i) niveau (int) : Le niveau des joueurs recherchés
   - Retour : Une liste de Joueurs représentant tous les joueurs trouvés dans la source de données
     - Type de retour : ArrayList<Joueur>
   - Assertions :
     (i) niveau est >= 0.
   - Lance :
     (i) DAOException : si un problème survient lors de la modification.

7 Classe Partie

Le résultat d'une partie de démineur.

Le niveau de difficulté d'une partie est déterminé selon la taille de la grille et le nombre de mines comme suit :

1. Facile : 9x9, 10 mines
2. Moyen : 16x16, 40 mines
3. Difficile : 24x24, 99 mines

— Propriétés :

1. id : L'identifiant unique de la partie
   — Type : int
   — Accès : lecture seule
2. temps : Temps écoulé en secondes depuis le début de la partie ou jusqu'à ce qu'elle soit terminée.
   — Type :
   — Accès : lecture seule
3. date : Date de début de la partie
   — Type : LocalDateTime
   — Accès : lecture seule
4. date : Date de fin de la partie
   — Type : LocalDateTime
   — Accès : complet
5. niveauDifficulté : Niveau de difficulté de la partie.
   — Type : NiveauDifficulté
   — Accès : lecture seule

— Constructeurs :

1. Partie : Constructeur complet. Utile pour construire une partie en précisant toutes ses caractéristiques.
   — Paramètres :
   (i) unID (int) : identifiant unique de la partie.
   (ii) uneDateDébut (LocalDateTime) : La date et l'heure de début de la partie
   (iii) uneDateFin (LocalDateTime) : La date et l'heure de fin de la partie, null si le partie est en cours.

(iv) niveauDifficulté (Niveaudifficulté) : Le niveau de difficulté de la partie.

— Assertions :

(i) uneDateFin est null ou après uneDateDébut.

2. Partie : Constructeur avec valeurs par défaut. Utile pour créer une toute nouvelle partie. La partie est créée avec la date et l'heure courante et un id=-1.
   — Paramètres :
   (i) niveauDifficulté (Niveaudifficulté) : Le niveau de difficulté de la partie.

— Méthodes :

1. terminer : Termine une partie en cours en stoquant la date de fin (ne fait rien si la partie est déjà terminée).
2. getTemps : Retourne le nombre de seconde qu'a duré la partie (entre la date de début et la date de fin)
   — Retour : le nombre de seconde qu'a duré la partie. Si la partie est encore en cours, retourne le nombre de secondes écoulées depuis le début.
   — Type de retour : int

8 Classe PartieDAO

La classe d'accès aux données de la Partie
— Méthodes :

1. lire : instancie une Partie à partir des informations provenant de la source de données.
   — Paramètres :
   (i) identifiant (Object) : L'ID de la Partie
   — Retour : Le Partie provenant de la source de données ou null s'il n'a pas été trouvé.
   — Type de retour : Partie
   — Assertions :
   (i) identifiant est un Integer.
   — Lance :

(i) DAOException : si un problème survient lors de la lecture.

2. créer : ajoute un nouvelle Partie dans la source de données.
   - Paramètres :
     (i) unePartie (Partie) : Le Partie à ajouter
   - Retour : Le Partie ajoutée à la source de données telle qu'elle peut avoir été modifiée lors de l'ajout.
     - Type de retour : Partie
   - Assertions :
     (i) unePartie est non null
   - Lance :
     (i) DAOException : si un problème survient lors de la création.
3. modifier : modifie une partie dans la source de données.
   - Paramètres :
     (i) unePartie (Partie) : Le Partie telle qu'elle doit devenir dans la source de données.
   - Retour : Le Partie après modification dans la source de données.
     - Type de retour : Partie
   - Assertions :
     (i) unePartie est non null
   - Lance :
     (i) DAOException : si un problème survient lors de la modification.
4. supprimer : supprime une partie de la source de données.
   - Paramètres :
     (i) unePartie (Partie) : La Partie à supprimer
   - Assertions :
     (i) unePartie est non null
   - Lance :
     (i) DAOException : si un problème survient lors de la suppression.
5. trouverTout : Recherche toutes les parties à partir de la source de données

— Retour : Une liste de Parties représentant toutes les parties trouvées dans la source de données
  — Type de retour : ArrayList<Partie>
— Lance :
  (i) DAOException : si un problème survient lors de la recherche.

6. trouverParDifficulté : Recherche toutes les parties d'un certain niveau de difficulté dans la source de données.
— Paramètres :
  (i) difficulté (Niveau) : Le niveau de difficulté des parties recherchées
— Retour : Une liste de Parties représentant toutes les parties trouvés dans la source de données
  — Type de retour : ArrayList<Partie>
— Lance :
  (i) DAOException : si un problème survient lors de la recherche.

7. trouverParDate : Recherche toutes les parties ayant débuté à une certaine date dans la source de données.
— Paramètres :
  (i) date (Date) : La date de début des parties recherchées
— Retour : Une liste de Parties représentant toutes les parties trouvées dans la source de données
  — Type de retour : ArrayList<Partie>
— Lance :
  (i) DAOException : si un problème survient lors de la recherche.