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
   — Assertions :
   (i) La largeur doit être entre 5 et `Integer.MAX_VALUE` inclusivement.
   — Exception : IllegalArgumentException
   — Message : La grille doit avoir une largeur comprise 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