Exemple : jeu de Morpion (1)

On veut coder une classe permettant de représenter le plateau 3x3 d'un jeu de Morpion (tic-tac-toe) :

```
class JeuMorpion {
   private int[] grille;
   public void initialise() {
      grille = new int[9];
   }
   public int[] getGrille() {
      return grille;
   }
}
```

Exemple : jeu de Morpion (2)

```
class JeuMorpion {
   private int[] grille;
   public void initialise() {
      grille = new int[9];
   }
   public int[] getGrille() {
      return grille;
   }
}
```

Le joueur rond coche la case en haut à gauche :

```
JeuMorpion jeu = new JeuMorpion();
jeu.initialise();
jeu.getGrille()[0] = 1;
```

Convention : 1 représente un rond, 2 une croix et 0 une case vide

Exemple : jeu de Morpion (3)

Ce code est parfaitement fonctionnel mais ... pose **beaucoup** de problèmes:

- L'utilisateur de la classe `JeuMorpion` doit savoir que les cases sont stockées sous forme d'entiers dans un tableau 1D, ligne par ligne (et non colonne par colonne)
- Il doit savoir que la valeur entière 0 correspond à une case non cochée, que 1 correspond à un rond, et que la valeur 2 correspond à une croix.

☞ L'utilisateur doit connaître « le codage » des données

```
JeuMorpion jeu = new JeuMorpion();
jeu.initialise();
jeu.getGrille()[0] = 1;
```

Exemple : jeu de Morpion (4)

```
JeuMorpion jeu = new JeuMorpion();
jeu.initialise();
jeu.getGrille()[0] = 1;
```

- Le code est complètement cryptique pour une personne qui n'est pas intime avec les entrailles du programme. 0, 1, 2 ? Que cela signifie-t-il ? Impossible de le deviner juste en lisant ce code. Il faut aller lire le code de la classe `JeuMorpion` (ce qui devrait être inutile), et en plus ici `JeuMorpion` n'est même pas documentée !
- Le code n'est pas encapsulé : on a un accesseur public vers une variable privée, donc... on ne peut pas la modifier, non ? Malheureusement si : c'est un tableau, donc on peut directement modifier son contenu ce qui viole l'encapsulation.
- Que se passerait-il si pour représenter le plateau de jeu, on décidait de changer et d'utiliser un tableau 2D ? Ou 9 variables entières ?
  - ☞ Le code écrit par l'utilisateur de la classe `JeuMorpion` serait à réécrire !

Exemple : jeu de Morpion (5)

- Si l'utilisateur s'avisait de faire `jeu.getGrille()[23] = 1`; il aurait un message d'erreur (un `ArrayIndexOutOfBoundsException`)
- Si l'utilisateur avait envie de mettre la valeur 3 ou 11 ou 42 dans le tableau, rien ne l'en empêche - mais d'autres méthodes, comme par exemple `getJoueurGagnant()`, qui s'attendent uniquement aux valeurs 1 et 2 ne fonctionneront plus du tout!
- Si l'utilisateur avait envie de tricher et de remplacer un rond par une croix ? Il suffit d'écraser la valeur de la case avec la valeur 2 !

☞ Les méthodes choisies ici donnent un accès non contrôlé aux données et n'effectuent aucune **validation** des données

Jeu de Morpion : bien encapsuler (1)

```
class JeuMorpion {
 private final static int VIDE  = 0;
 private final static int ROND  = 1;
 private final static int CROIX = 2;

 private int[][] grille;

 public initialise() {
  grille = new int[3][3];
  for (int i=0; i < grille.length; ++i) {
   for (int j=0; j < grille[i].length; ++j)
   {
    grille[i][j] = VIDE;
  }
 }
//...
```

Jeu de Morpion : bien encapsuler (2)

```
/**
 * Place un coup sur le plateau.
 * @param ligne La ligne 0, 1, ou 2
 * @param colonne La colonne 0, 1, ou 2
 * @param coup Le coup à placer
 */
private boolean placerCoup(int ligne, int colonne, int coup) {
 if (ligne < 0 || ligne >= grille.length
        || colonne < 0 || colonne >= grille[ligne].length) {
     // traitement de l'erreur ici
 }
 if(grille[ligne][colonne] == VIDE) {
  // case vide, on peut placer le coup
  grille[ligne][colonne] = coup;
  return true;
 } else {
      // case déjà prise, on signale une erreur
      //...
   return false;
 } // suite
```

Jeu de Morpion : bien encapsuler (3)

```
public boolean placerRond(int ligne, int colonne) {
   return placerCoup(ligne, colonne, ROND);
}

public boolean placerCroix(int ligne, int colonne) {
   return placerCoup(ligne, colonne, CROIX);
}

// ici on peut rajouter une methode getJoueurGagnant()

} // fin de la classe JeuMorpion
```

Jeu de Morpion : bien encapsuler (4)

Comment faire maintenant pour faire un rond sur la case en haut à gauche ?

```
JeuMorpion jeu = new JeuMorpion();
jeu.initialise();
valide = jeu.placerRond(0, 0); //boolean déclaré plus haut
```

Et pour faire une croix sur la 1re ligne, 2e colonne ?

```
valide = jeu.placerCroix(0, 1);
```

On aurait pu également décider d'appeler les colonnes 1, 2, 3 au lieu de 0, 1, 2 : c'est une question de convention. C'est justement ce sur quoi il faut se mettre d'accord quand on définit une interface.

Jeu de Morpion encapsulé : avantages

- **Validation** : il est impossible de mettre une valeur invalide dans le tableau (autre que 0, 1, ou 2)
- **Validation** : il est impossible de cocher une case déjà cochée.
- **Séparation des soucis** : le programmeur-utilisateur n'a pas besoin de savoir comment le plateau est stocké, ni qu'il utilise des entiers, ni quelles valeurs correspondent à quoi.
- Le code est compréhensible même par un profane – le nom des méthodes exprime clairement ce qu'elles font et s'explique de lui-même.
- Si on essaie de faire une opération invalide (cocher deux fois la même case, ou une case en dehors du tableau), on obtient un message compréhensible.