Bienvenue dans la Matrice.
Vous contrôlez un personnage, perdu dans la Matrice, qui doit récolter des ressources au fur et à mesure de son périple afin de se nourrir, et retrouver son chemin.

Créez le package matrice.

La Matrice est un tableau de 10 lignes et 10 colonnes. Chaque élément de ce tableau a une valeur parmi les 4 suivantes

- null
- Blé
- Bois
- Pierre

Blé, bois et pierre sont des ressources, qui héritent donc de la classe Ressource. La classe Ressource a deux attributs :

- le poids : par défaut, celui-ci est de zéro.
- un identifiant unique sous forme de String qui permet de différencier deux ressources.

Pour le blé le poids est de 1, pour le bois, le poids est de 2 et la pierre a un poids de 3.

Les classes Bois et Pierre instancient toutes les deux l'interface Utilisable qui a une méthode dont la signature est
public ObjetManufacture utiliser(Ressource[] r)

ObjetManufacture est la classe des objets manufacturés comme le feu et la farine. Elle n'a qu'un seul attribut qui est le type d'objet.
Ex: ObjetManufacture maFarine = new ObjetManufacture("farine");

La méthode utiliser de Bois prend un tableau de 5 bois en paramètre et retourne un objet ObjetManufacture de type Feu.
La méthode utiliser de Pierre, prend une tableau de 10 blé en paramètre et retourne un objet ObjetManufacture de type Farine.
Bien entendu, les instances des objets donnés en paramètre de ces implémentations doivent être différentes, on ne peut pas utiliser 6 fois le même objet Bois ou 10 fois le même Blé.

Créez ces classes.

Votre classe héros a quant à elle les attributs suivants:

- un nom
- une liste d'objets Ressource, qui contient le stock de blé
- une liste d'objets Ressource, qui contient le stock de bois
- un objet Pierre
- un ObjetManufacture Farine
- un ObjetManufacture Feu

- un poids emporté, qui vaut zéro à la création de la classe et qui ne peut pas dépasser 13.
- le nombre de parties jouées
- la case courante, sous la forme d'une paire (x,y)

Le constructeur de la classe héros prend un nom en paramètre, initialise tous les objets à null, les compteurs à 0 et les coordonnées de départ à (0, 0).

La méthode jouer a pour signature public int jouer(Ressource[][] map)
Le but de cette méthode est de partir de la case 0, 0 de la map et d'arriver à la case 9, 9 avec un pain en un minimum de déplacements. Chaque déplacement est comptabilisé et le total est la valeur de retour de la méthode.

Vous ne pouvez vous déplacer que en haut, à bas, à droite ou à gauche (créez un ENUM avec ces valeurs, ainsi que la méthode privée seDeplacer qui prend en paramètre la direction et qui lance une exception si vous essayez de vous déplacer en dehors du tableau).
Sur chaque case se trouve est objet de type Ressource, qui peut être soit null, soit un blé, soit une pierre, soit un bois. Attention, il faut prendre en compte le poids emporté : s'il est supérieur à 13, notre héros ne peut pas se déplacer.

Implémentez la méthode private void prendre() qui permet de prendre la ressource sur la case sur laquelle vous êtes. Vous l'ajoutez à votre inventaire, et vous mettez la valeur de cette case à null. N'oubliez pas de prendre en compte le poids ici aussi.

Vous pouvez implémenter la méthode private void jeter(Ressource r) qui permet de jeter une ressource sur une case, si celle-ci a une valeur null. La valeur de la case devient donc la ressource que vous avez jetée. Pensez à retirer le poids de votre sac.

Vous devez récupérer 10 blés et une pierre pour faire de la farine. Mais aussi six bois pour faire du feu. Attention, vous ne pouvez porter des objets que pour un poids total de 13. Le blé pèse 1, le bois 2 et la pierre 3. Vous devez donc soit récupérer tout le bois pour faire un feu, et ensuite récupérer le blé et la pierre, soit récupérer le blé et la pierre, faire la farine puis jeter la pierre et récupérer le bois.

Créez une méthode public fairePain qui vérifie que farine et feu sont non null et renvoie true si c'est le cas

Le but est donc de partir de la case (0,0) et d'arriver à la case (9,9) avec un pain, en un minimum de coups.

Vous devez afficher le nombre de coups à la fin de votre partie, ainsi que le fait que le pain ait été fait ou pas.

Les méthodes et attributs demandés par l'exercice sont obligatoires mais il n'est pas nécessaire de toutes les utiliser si vous voulez faire un algo simple . Vous avez la possibilité de rajouter des méthodes et des attributs de votre choix, afin de créer un algo le plus optimal.

Le code sera testé sur 10 matrices différentes. Deux matrices d'exemple seront données pour vos tests, libre à vous de la modifier selon les cas que vous voulez tester.

```
Ressource[][] matrix1 = new Ressource[10][10];

        for (int i = 0; i < 10; i++)
        {
                matrix1 [i][i] = new Ble("ble" + i);
        }

        matrix1 [0][1] = new Pierre("Paul");
        matrix1 [2][5] = new Bois("bois25");
        matrix1 [3][1] = new Bois("bois31");
        matrix1 [4][8] = new Bois("bois48");
        matrix1 [5][2] = new Bois("bois52");
        matrix1 [5][3] = new Bois("bois53");
        matrix1 [9][1] = new Bois("bois91");


Ressource[][] matrix2 = new Ressource[10][10];

for (int i = 0; i < 10; i++)
        {
                if (i % 2 == 0)
                {
                        matrix2 [i][3] = new Ble("ble" + i);
                } else
                {
                        matrix2 [i][4] = new Ble("ble" + i);
                }

        }
        matrix2 [8][5] = new Pierre("Jacques");
        matrix2 [0][5] = new Bois("bois05");
        matrix2 [5][1] = new Bois("bois51");
        matrix2 [6][8] = new Bois("bois68");
        matrix2 [8][2] = new Bois("bois82");
        matrix2 [8][3] = new Bois("bois83");
        matrix2 [9][9] = new Bois("bois99");


Ressource[][] matrix3 = new Ressource[10][10];
```

```
for (int i = 0; i < 10; i++)
	{
		if (i % 2 == 0)
		{
			matrix3 [i][3] = new Ble("ble" + i);
		} else
		{
			matrix3 [i][4] = new Ble("ble" + i);
		}

	}
	matrix3 [8][5] = new Pierre("Jacques");
	matrix3 [0][0] = new Bois("bois00");
	matrix3 [0][1] = new Bois("bois01");
	matrix3 [0][2] = new Bois("bois02");
	matrix3 [0][3] = new Bois("bois03");
	matrix3 [0][4] = new Bois("bois04");
```

La notation prendra en compte les points suivants :

- respect des conventions de nommage
- respect des consignes
- clarté du code
- performance de l'algorithme (classement)
- javadoc
- commentaires

Pour ceux qui veulent aller plus loin, le blé pousse en diagonale, c'est-à-dire que si du blé se trouve à la case (5,5), il y aura au moins un blé sur les cases (4,4), (4,6), (6,4), (6,6).

```
Ressource[][] matrix4 = new Ressource[10][10];
	for (int i = 0; i < 10; i++) {
		if (i % 2 == 0) {
```

```
                matrix4[i][0] = new Ble("ble" + i);
        } else {
                matrix4[i][1] = new Ble("ble" + i);
        }
}
matrix4[9][9] = new Pierre("Jacques");
matrix4 [3][1] = new Bois("bois31");
matrix4 [4][8] = new Bois("bois48");
matrix4 [8][2] = new Bois("bois82");
matrix4 [8][3] = new Bois("bois83");
matrix4 [8][9] = new Bois("bois89");

Ressource[][] matrix5 = new Ressource[10][10];
matrix5 [0][1] = new Ble("blé 01");
matrix5 [1][2] = new Ble("blé 12");
matrix5 [2][3] = new Ble("blé 12");
matrix5 [3][4] = new Ble("blé 12");
matrix5 [4][5] = new Ble("blé 12");
matrix5 [3][6] = new Ble("blé 12");
matrix5 [2][7] = new Ble("blé 12");
matrix5 [1][8] = new Ble("blé 12");
matrix5 [0][9] = new Ble("blé 12");
matrix5 [3][8] = new Ble("blé 12");


matrix5 [5][5] = new Pierre("Jacques");
matrix5 [3][1] = new Bois("bois31");
matrix5 [4][8] = new Bois("bois48");
matrix5 [8][2] = new Bois("bois82");
matrix5 [8][3] = new Bois("bois83");
matrix5 [8][9] = new Bois("bois89");


Ressource[][] matrix6 = new Ressource[10][10];
matrix6 [0][0] = new Ble("blé 01");
matrix6 [1][1] = new Ble("blé 12");
matrix6 [0][2] = new Ble("blé 12");
matrix6 [1][3] = new Ble("blé 12");
matrix6 [0][4] = new Ble("blé 12");
matrix6 [1][5] = new Ble("blé 12");
matrix6 [0][6] = new Ble("blé 12");
matrix6 [1][7] = new Ble("blé 12");
matrix6 [0][8] = new Ble("blé 12");
matrix6 [1][9] = new Ble("blé 12");


matrix6 [5][5] = new Pierre("Jacques");
matrix6 [3][1] = new Bois("bois31");
```

```
matrix6 [4][8] = new Bois("bois48");
matrix6 [8][2] = new Bois("bois82");
matrix6 [8][3] = new Bois("bois83");
matrix6 [8][9] = new Bois("bois89");


Ressource[][] matrix7 = new Ressource[10][10];
matrix7 [5][5] = new Ble("blé 55");
matrix7 [4][4] = new Ble("blé 44");
matrix7 [3][3] = new Ble("blé 33");
matrix7 [6][6] = new Ble("blé 66");
matrix7 [7][7] = new Ble("blé 77");
matrix7 [4][6] = new Ble("blé 46");
matrix7 [3][7] = new Ble("blé 37");
matrix7 [2][8] = new Ble("blé 28");
matrix7 [6][4] = new Ble("blé 64");
matrix7 [7][3] = new Ble("blé 73");


matrix7 [1][4] = new Pierre("Jacques");
matrix7 [3][1] = new Bois("bois31");
matrix7 [4][8] = new Bois("bois48");
matrix7 [8][2] = new Bois("bois82");
matrix7 [8][3] = new Bois("bois83");
matrix7 [8][9] = new Bois("bois89");

Ressource[][] matrix8 = new Ressource[10][10];
matrix8 [5][5] = new Ble("blé 55");
matrix8 [4][4] = new Ble("blé 44");
matrix8 [5][3] = new Ble("blé 53");
matrix8 [6][4] = new Ble("blé 64");
matrix8 [7][5] = new Ble("blé 75");
matrix8 [6][6] = new Ble("blé 66");
matrix8 [3][5] = new Ble("blé 35");
matrix8 [4][6] = new Ble("blé 46");
matrix8 [5][7] = new Ble("blé 57");
matrix8 [6][8] = new Ble("blé 68");


matrix8 [1][4] = new Pierre("Jacques");
matrix8 [3][1] = new Bois("bois31");
matrix8 [4][8] = new Bois("bois48");
matrix8 [8][2] = new Bois("bois82");
matrix8 [8][3] = new Bois("bois83");
matrix8 [8][9] = new Bois("bois89");

Ressource[][] matrix9 = new Ressource[10][10];
matrix9 [0][2] = new Ble("blé 02");
```

```
matrix9 [0][4] = new Ble("blé 04");
matrix9 [0][6] = new Ble("blé 06");
matrix9 [0][8] = new Ble("blé 08");
matrix9 [1][1] = new Ble("blé 11");
matrix9 [1][3] = new Ble("blé 13");
matrix9 [1][5] = new Ble("blé 15");
matrix9 [1][7] = new Ble("blé 17");
matrix9 [1][9] = new Ble("blé 19");
matrix9 [2][8] = new Ble("blé 28");


matrix9 [0][1] = new Pierre("Jacques");
matrix9 [0][0] = new Bois("bois00");
matrix9 [0][3] = new Bois("bois03");
matrix9 [0][5] = new Bois("bois05");
matrix9 [0][7] = new Bois("bois07");
matrix9 [0][9] = new Bois("bois09");


Ressource[][] matrix10 = new Ressource[10][10];
matrix10 [0][2] = new Ble("blé 02");
matrix10 [0][4] = new Ble("blé 04");
matrix10 [0][6] = new Ble("blé 06");
matrix10 [0][8] = new Ble("blé 08");
matrix10 [1][1] = new Ble("blé 11");
matrix10 [1][3] = new Ble("blé 13");
matrix10 [1][5] = new Ble("blé 15");
matrix10 [1][7] = new Ble("blé 17");
matrix10 [1][9] = new Ble("blé 19");
matrix10 [2][8] = new Ble("blé 28");


matrix10 [0][0] = new Pierre("Jacques");
matrix10 [0][1] = new Bois("bois00");
matrix10 [0][3] = new Bois("bois03");
matrix10 [0][5] = new Bois("bois05");
matrix10 [0][7] = new Bois("bois07");
matrix10 [0][9] = new Bois("bois09");
```