Enumeration, Vue, Iterateur.

Exercice 1 - Parsing d'expressions

Écrire une classe `fr.umlv.calc.ExprParser` avec une méthode `Expr parse(Scanner sc)` qui construit une expression arithmétique de type `fr.umlv.calc.Expr` à partir de ce qu'elle lit sur le scanner passé en argument.
On suppose pour cela que l'expression lue par le scanner est en notation préfixe, c'est-à-dire qu'on exprime l'opérateur, puis ses deux opérandes (on se limite au cas classique des opérations binaires +, -, / et *).
Par exemple, l'expression préfixe :

```
+ - 7 3 / 5 * + 4 5 - 4 2
```

représente en notation infixe l'expression

```
(7 - 3) + 5 / ((4 + 5) * (4 - 2))
```

Pour l'analyse, on pourra supposer que tous les opérateurs débutent par des caractères différents (on peut faire un "switch").
Dans le cas où l'expression analysée serait incorrecte, on lèvera une exception de la classe `ExprParseException` qu'il faudra définir. Faite en sorte d'obliger le programmeur à récuperer cette exception. L'exception devra représenter le plus fidèlement possible le problème rencontré.

Exercice 2 - Vue d'une liste

Qu'affiche le code ci-dessous :

```
String[] array=new String[]{"toto","titi","tutu"};
List<String> list=Arrays.asList(array);
Collections.sort(list);

System.out.println(array[0]);
```

Expliquer le concept de vue.

Exercice 3 - Jeux de cartes

On souhaite modéliser un jeux de 32 cartes (8 * 4 couleurs). Les 8 rangs sont `SEVEN`, `EIGHT`, `NINE`, `TEN`, `JACK`, `QUEEN`, `KING`, `ACE` et les quatres suites `CLUBS`, `DIAMONDS`, `HEARTS`, `SPADES`.

1. Écrire une classe `Card` prenant en paramètre un rang et une suite. Utiliser les énumérations pour modéliser les rangs et les suites.
2. Écrire la méthode `toString()` affichant la carte `new Card(Rank.ACE,Suit.SPADES)` comme ceci `ACE of SPADES`.
   Note: les valeurs d'une énumération possèdent une méthode `name()`.
3. Créer une méthode `List<Card> newDeck()` qui renvoie un jeu de 32 cartes.
   Note : La classe d'une énumération possède un méthode statique `values()` qui

renvoie un tableau contenant les valeurs de l'énumération.

4 Comment interdire que plusieurs cartes d'une même type existe ?

5 En supposant que l'ordre de trie des cartes est d'abord en fonction de leur suite puis de leur rang. Que doit-on faire pour que le code suivant trie les cartes ?

```
List<Card> deck=newDeck();
    Collections.shuffle(deck);
    System.out.println(deck);
    Collections.sort(deck);
System.out.println(deck);
```

Note: les valeurs d'une énumération possède une méthode `ordinal()` ou mieux les valeurs de l'énumération sont comparables.

Exercice 4 - Jeux de cartes (suite :)

Pour tester vos différentes méthodes, utilisez la classe `Card` de l'exercice précédent.

1 Écrire une méthode `reverseIterator()` dans une classe `Utils` qui prend en paramètre une liste de carte et qui renvoie un iterateur (`Iterator`) qui permet de parcourir la liste en ordre inverse.
Dans un premier temps, utiliser une `List<Card>`
Pour cela, vous devrez créer une classe `ReverseIterator` implantant l'interface `Iterator`.

2 Modifier la méthode `reverseIterator` pour implanter l'iterateur sous forme de classe anonyme.

3 Écrire une méthode `shuffleIterator` qui renvoie un iterateur parcourant de façon mélangée une liste passée en argument.
Pour ne pas changer la liste initiale, on créra un tableau d'entier contenant les valeurs de `0` à la `taille de la liste-1` sur lequel on appliquera le shuffle (pensez aux vues). On se servira de ce tableau comme d'index des éléments à parcourir lors de l'itération.

4 A quoi sert l'interface `java.lang.Iterable`
Écrire une méthode `shuffleIterable` prenant une liste en paramètre et qui renvoie un `Iterable` permettant d'obtenir des iterateurs parcourant la liste de façon mélangée.

5 Enfin, écrire une méthode `shuffleList` prenant une liste en paramètre et qui renvoie une `List` mélangée qui est une vue de la liste initiale.
On considerera que la liste initiale ne peux pas changée de taille.