JSON est un format de sérialisation facile à lire à la fois pour les humains et pour les machines. Une valeur JSON est :

- un nombre
- une chaine de caractère
- la valeur **null**
- la valeur **true**
- la valeur **false**
- un tableau de valeur de la forme "[valeur,...]"
- une table associative de chaine de caractère vers des valeurs de la forme "{ clé :valeur, ...}".

Question 1 Lire la page web `http://www.json.org/json-fr.html`.

1 Lexer

On considérera pour ce TP que le texte à parser est donné comme une String. On commence par créer une classe StringLexer qui va simplifier le parsing. Cette classe contient une chaine de caractère et une position et implémente des méthodes pour manipuler l'avancement du parsing dans la chaine de caractère. La classe StringLexer implémente l'interface Lexer.

```
class ParseException extends Exception {
  public ParseException(int position, char found, char expcted);
}

interface Lexer{
  /* Renvoie le prochain caractère à lire sans le modifier */
  public char current();
  /* renvoie le prochain caractère à lire et incrémente la
     position du caractère courant */
  public char get() throws EOFException ;
  /* incrémente la position pour atteindre un caractère
     blanc( espace, tabulation, saut de ligne ) */
  public void skipWhiteSpace();
  /* Incrémete la position jusqu'à atteindre un caractère blanc */
  public void next() throws EOFException ;
  /* vérifie que le caractère courant est celui attendu
     ou renvoi une exception */
  public void check(char c) throws ParseException;
}
```

Question 2 Implémenter la classe StringLexer. Ajouter un constructeur public prenant en entrée une String. *Un caractère est testé comme blanc avec* Character.isWhitespace(**char**))

2 Type de Base

Question 3 Créé une classe abstraite de base appelée JsonValue, contenant une methode de signature **public void** parse(Lexer l) **throws** EOFException,ParseException .

La méthode parse sera utilisé dans la suite pour initialiser les champs en parsant le contenue du lexer.

On commence par parser les chaines de caractère. Une chaine de caractère en JSON est une suite de caractère Unicode délimité par des “ " ”. Pour éviter les conflits, le caractère “ " ” est représenté par “ \ " ”.

Question 4 Implémenter une classe JsonString héritant de JsonValue et avec un constructeur vide et un constructeur à partir d'une String. La méthode parse fait l'assertion que le caractère courant du lexer est une “ " ” marquant le début d'une chaîne de caractère. *On pourra utiliser un StringBuilder pour fabriquer la chaine à partir des données lues. Ici on n'échappe que les apostrophes, le format JSON échappe d'autres caractères.*

Pour parser les nombres on va utiliser la propriété qu'en JSON un nombre est toujours suivit par un caractère diffèrent de [0-9+-eE.] ou la fin du fichier. On va supposer qu'un nombre est toujours représenté par un **double**

Question 5 Implémenter une classe JsonNumber héritant de JsonValue et avec un constructeur vide et un constructeur à partir d'un **double**. La méthode parse fait l'assertion que le caractère courant du lexer fait partie de [-0-9]. *On pourra utiliser* Double.parseDouble(String s).

Question 6 Implémenter les classes JsonNull, JsonTrue, JsonFalse héritant toutes de JsonValue qui représente les valeurs **null**, **true**, **false**.

3 Factory

On remarque que dans le format JSON le premier caractère d'une valeur permet de connaître son type :

- “ " ” → JsonString
- “[-0-9]” → JsonNumber
- “n” → JsonNull
- “t” → JsonTrue
- “f” → JsonFalse
- “[” → tableau
- “{” → objet

Question 7 Écrire une méthode factory **public static** Value parseValue(Lexer l) **throws** EOFException,ParseE dans la classe JsonValue, qui appelle le constructeur nulle de chaque objet en fonction du premier caractère, puis appelle la méthode parse de l'objet créé. *On ignore pour le moment le cas des tableaux et des objets. On prêtera attention au caractère blanc avant et après les valeurs*

Question 8 Implémenter des méthodes toString() pertinentes dans toutes les classes.
Tester !

4 Type hiérarchique

Question 9 Implémenter une classe JsonArray héritant de JsonValue. La méthode parse fait l'assertion que le caractère courant du lexer est "[". Ajouter ce cas à la méthode JsonValue.parseValue()

Question 10 Implémenter une classe JsonObject héritant de JsonValue. La méthode parse fait l'assertion que le caractère courant du lexer est "[". Ajouter ce cas à la méthode JsonValue.parseValue()

Question 11 Tester sur les exemples disponibles sur `http://json.org/example.html`.

Question 12 Créé une classe CharBufferLexer qui hérite de Lexer et qui est construit avec un CharBuffer.

Question 13 Reprendre les classes JsonString et JsonNumber pour correspondre exactement à la spécification de Json (caractère échappé format des nombres).

Question 14 *** Utiliser la réflexion pour transformer un objet JSON en objet Java. *C'est déjà implémenté par exemple dans la librairie GSON.*