Projet 2 : Les arbres binaires
Le jeu des questions

Rappel : Tout votre code doit être soigneusement commenté au format `javadoc`.

1 Arbres binaires : rappels

Un arbre est une structure de données constituée de *nœuds* contenant chacun le même type d'informations. Chaque nœud contient des connexions sur d'autres nœuds appelés *sous-arbres* et il n'y a pas de cycle. De plus il existe un nœud qui n'est sous-arbre d'aucun autre : la *racine* de l'arbre. Les nœuds qui, au contraire, n'ont aucun sous-arbre sont appelés *feuilles*.

En plus de son *contenu*, chaque nœud d'un arbre *binaire* garde une référence sur deux sous-arbres (c'est-à-dire deux autres nœuds) appelés généralement sous-arbre gauche et sous-arbre droit.

Nous avons vu en cours la structure d'un arbre binaire contenant des *entiers* (`int`) :

```
public class ArbreBinaire {
    // le contenu du noeud
    private int contenu ;
    // les sous-arbres gauche et droit
    private ArbreBinaire gauche, droit ;
    ...
}
```

Nous allons dans ce projet l'adapter pour un arbre contenant des *chaînes de caractères* (`String`).

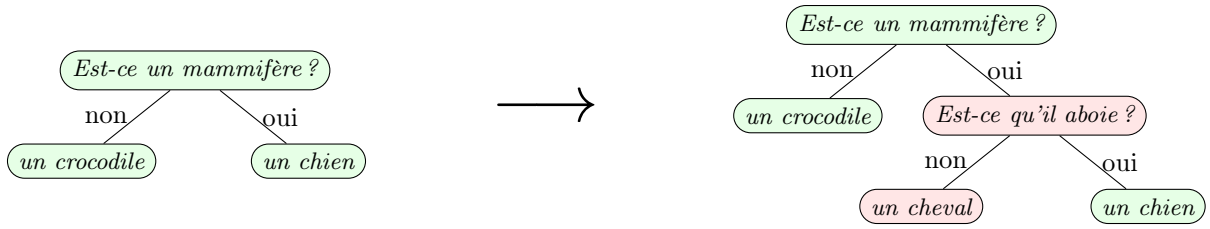
2 Arbres binaires : jeu des questions

On veut écrire un programme qui reconnaît des animaux en posant des questions. La machine possède au départ très peu de connaissances et apprend progressivement en jouant. Voici un exemple d'exécution :

Pensez à un animal, je vais essayer de le deviner...
– Est-ce un mammifère ?
- `oui` *(réponse donnée par le joueur)*
– Est-ce un chien ?
- `non`
– Qu'est-ce que c'est ?
- `un cheval`
– un cheval ! Je ne connais pas cet animal. Donnez-moi une question qui permette de différencier un cheval d'un chien :
- `Est-ce qu'il aboie ?`
– Quelle doit être la réponse pour un cheval ?
- `non`
– Voulez-vous rejouer ?
- `oui`
Pensez à un animal, je vais essayer de le deviner...
– Est-ce un mammifère ?
- `non`
– Est-ce un crocodile ?
- `oui`
– Trouvé ! Voulez-vous rejouer ?
- `oui`
Pensez à un animal, je vais essayer de le deviner...
– Est-ce un mammifère ?
- `oui`
– Est-ce qu'il aboie ?
- `oui`
– Est-ce un chien ?
- `oui`
– Trouvé ! Voulez-vous rejouer ?
- `non`

Principe et algorithmes

Pour stocker ses connaissances, le programme peut utiliser un arbre binaire, puisque pour chaque question la réponse est soit oui, soit non. En fonction des réponses données aux questions, le programme s'oriente vers diverses possibilités. Dans l'exemple d'exécution précédent, le processus d'apprentissage consiste à transformer l'arbre initial (à gauche) en un nouvel arbre (à droite) de la façon suivante :



On peut distinguer deux phases successives : la **recherche** dans l'arbre (pour poser des questions) et la **modification** de l'arbre (pour apprendre de nouveaux animaux).

Poser des questions judicieuses

Chaque nœud qui n'est pas une feuille contient une *question*. La réponse donnée par le joueur permet de savoir dans quel sous-arbre se trouve la suite du questionnaire. Lorsqu'on arrive à une feuille, on dispose seulement du nom de l'animal, mais il n'est pas difficile de formuler la question correspondante à partir de là.

Apprendre les nouveaux animaux

Lorsqu'on atteint une feuille, si la réponse donnée par le joueur est « oui », tout va bien car l'ordinateur a deviné juste. En revanche, si la réponse est « non », il faut :

1. demander le nom de l'animal choisi par le joueur (dans l'exemple : `un cheval`) ;
2. créer deux nouvelles feuilles qui vont devenir les sous-arbres de l'ancienne feuille : dans l'une, on place l'ancien animal (celui qui n'était pas le bon, ici : `un chien`), dans l'autre celui choisi par le joueur (`un cheval`) ;
3. demander au joueur une nouvelle question permettant de distinguer les deux animaux : cette question occupe le contenu de l'ancienne feuille (`Est-ce qu'il aboie ?`) ;
4. déterminer sur quelle branche placer chaque nouvelle feuille (selon la réponse à cette question pour le nouvel animal).

Étapes demandées

Q 1. Après avoir réfléchi aux attributs et surtout aux constructeurs nécessaires, écrivez en Java la classe `NoeudArbre` qui représente un arbre binaire contenant des **chaînes de caractères**.

Q 2. Écrivez une méthode `String toString()` qui affiche l'ensemble de l'arbre en parcours **préfixe**. Testez-la avec au moins les deux arbres ci-dessus. Exemple pour l'arbre final :

```
"Est-ce un mammifère ?" "un crocodile" "Est-ce qu'il aboie ?" "un cheval" "un chien"
```

Q 3. Écrivez dans `NoeudArbre` une méthode `rechercherAnimal` qui pose les questions adéquates les unes après les autres, jusqu'à ce que le programme ait gagné ou perdu, ainsi qu'une méthode `main` qui gère le jeu. À la fin du jeu, il faut afficher l'arbre final.

Q 4. Écrivez dans `NoeudArbre` une méthode `apprendre` permettant d'apprendre un nouvel animal pour le nœud courant. Cette méthode sera appelée à la fin de `rechercherAnimal` si le programme n'a pas réussi à deviner l'animal du joueur.

Q 5. Modifiez la méthode `main` pour donner la première question et les deux premières feuilles en ligne de commande. Exemple : `java NoeudArbre "Est-ce un vertébré ?" "un escargot" "un chat"`

Q 6. Écrire une méthode `String definir(String animal)` qui donne l'enchaînement questions-réponses permettant de définir l'animal spécifié (s'il existe). Exemple pour l'abre final ci-dessus :

```
$ java NoeudArbre --definir "un cheval" "Est-ce un mammifère ?" "un crocodile" "Est-ce qu'il aboie ?" "un
cheval" "un chien"
Est-ce un mammifère ? -> oui ; Est-ce qu'il aboie ? -> non => un cheval
```

Q 7. *QUESTION SUBSISDIAIRE :* Permettre de donner à l'exécution en ligne de commande *toutes les informations pour construire un arbre*, sous forme préfixe (autrement dit, permettant de reconstruire l'arbre à partir des données fournies par la méthode `toString()`). On peut supposer pour cela que toute question (donc tout ce qui doit occuper un nœud autre qu'une feuille) se termine par un `"?"`. Si l'affichage à la fin du jeu se fait sur la sortie d'erreur `System.err` (la sortie standard étant déjà utilisée pour poser les questions), on peut alors charger l'arbre à partir d'un fichier texte et sauver le résultat : `eval java NoeudArbre $(<abreInitial.txt) 2>arbreFinal.txt`

Déposez les sources sur la plateforme moodle dans un fichier pr2-*monLogin*-*monBinome*.tgz
Pour créer un fichier `pr2-toto-titi.tgz` dans le répertoire `travail` : `tar cvfz pr2-toto-titi.tgz sources`