

INF-111 Travail pratique #2

Équipe de 2

Remise: 10 mars 2017

À quoi pensez-vous ?

Conception et réalisation : Pierre Bélisle (copyright 2017).

Objectifs

Les objectifs que vise ce travail sont de (d') :

- Concrétiser les notions théoriques sur les collections de Java.
- Maintenir vos bonnes pratiques de découpage en sous-programmes, d'utilisation de paramètres par références et d'enregistrements.
- Implémenter et d'utiliser de classes (constructeurs, accesseurs, mutateurs, ...) et des types de données abstraits (type encapsulé).

Mise en contexte

Il s'agit pour ce travail d'écrire une application d'intelligence artificielle simple. Vous devez reproduire l'application <http://fr.akinator.com/objets/jeu> qui demande à l'utilisateur de penser à un animal, un objet ou un personnage, ensuite l'application tente de le trouver en posant des questions et en prenant en compte des indices fournis par l'utilisateur.

Note : Prenez le temps de vous familiariser avec la solution d'Akinator en jouant quelques parties avant de vous lancer dans ce travail.

Notre application est une variante moins évoluée que celle présentée par Akinator. En effet, dans notre version, l'utilisateur ne doit répondre que par oui ou non. Il y a d'autres fonctionnalités que nous avons laissées de côté par manque de temps dû à la complexité des solutions possibles. L'application aura quand même des fonctionnalités intéressantes. Par exemple, nous pourrions ajouter des réponses et des questions ou retrouver une erreur que l'utilisateur a faite en donnant ses indices (Oui ou Non).

C'est un problème qui peut être résolu d'une multitude de façon. Pour vous guider vers une implémentation qui fait utilisation des notions du cours, la conception a été réalisée et vous est imposée. De plus, il vous est interdit de modifier les fichiers fournis.

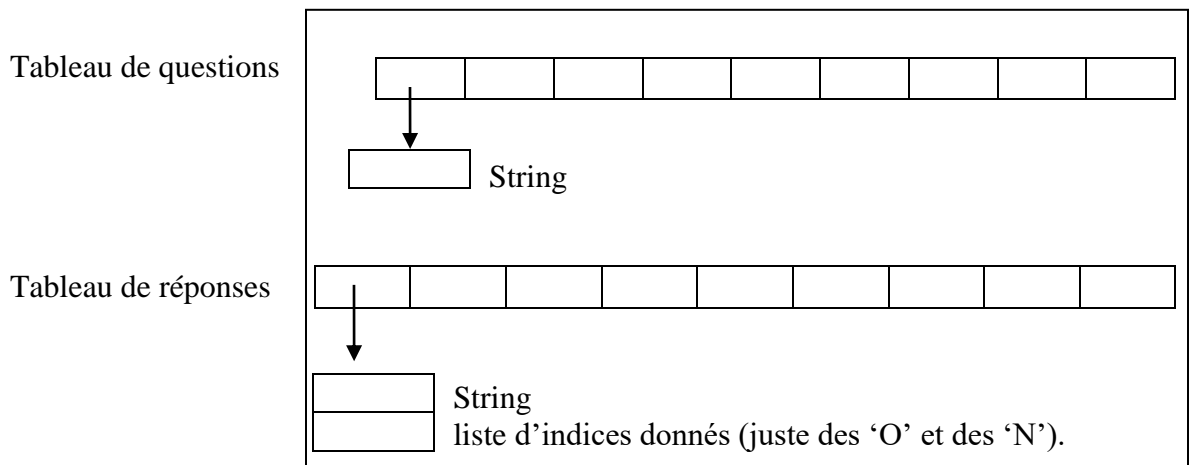
Structures de données imposées

Le premier problème à résoudre est de trouver qu'elles sont les données nécessaires et comment nous les conservons pour les retrouver par la suite. Les données à garder en mémoire sont : les *questions* et des *réponses*, toutes deux de type String.

Les éléments de type *questions* seront conservées dans une collection de **String** de votre choix - choisir **Vector** ou **ArrayList**. Chaque case (indice) doit correspondre à une question unique.

Les éléments de type *réponse* contiennent la réponse (String) et une liste des indices qu'il faut répondre aux questions pour y aboutir. Vous aurez donc une classe **Reponse** qui contient un attribut de type **String** pour la réponse et une structure dynamique d'indices qui ont été fournis par l'utilisateur au cours d'une séquence de jeu. Cette structure de données dynamique doit être une classe **Liste**, implémentée comme il vous a été enseigné dans **votre** cours par **votre** enseignant. Toutes les réponses sont conservées dans un tableau statique de **Reponse**.

Voici une image des attributs de la classe **BdQuestionsReponses** qui contient les réponses et les questions

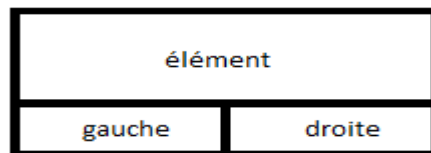


En résumé, 2 classes à écrire **Reponse** et **BdQuestionsReponses**. Deux attributs dans chacune pour l'instant. D'autres attributs seront ajoutés en cours de route.

Arbre de connaissance (structure pour trouver les réponses).

Avec la structure précédente, nous pouvons conserver toutes les questions, toutes les réponses et l'ordre des indices données. Il reste à trouver une stratégie pour relier le tout et permettre de jouer.

Nous avons décidé d'utiliser la technique de chaînage dynamique avec des nœuds liés ensembles. Plutôt que d'avoir des références vers le nœud précédent et le nœud suivant comme une liste doublement chaînée¹, nous allons avoir des références comme un nœud vers la 'gauche' et un nœud vers la 'droite' (imaginez que l'on regarde la structure de donnée de face).



Dans un nœud, il peut y avoir soit une question, soit une réponse. On distingue la réponse de la question à l'aide des références gauche et droite. Les réponses entraînent la fin d'une séquence de jeu, elles sont des cul-de-sac. Par conséquent, si les deux références sont nulles, c'est que l'élément du nœud est une réponse.

Dans un nœud question, il y a toujours au moins un nœud à gauche.

La convention qui sera utilisé est que le nœud gauche fait toujours référence au chemin à suivre quand la réponse est positive, alors que le nœud droit servira la réponse négative.

Vous connaissez maintenant tous les éléments structurels de la solution à produire, reste à détailler les algorithmes.

L'algorithme d'ajout des questions et réponses

La base de données (bd) est vide si un fichier dont le nom est donné par Constantes.NOM_FICHIER_BD est absent dans le répertoire de votre projet (pas dans src mais au même niveau). Si vous détruisez ce fichier, le programme recommence avec une nouvelle bd. Ce fichier est mis à jour à chaque fois que l'utilisateur ajoute une réponse et une question valides.

Comme il deviendra évident, on n'ajoute une réponse que si la bd ne la contient pas déjà, une réponse n'est donc jamais dupliquée. Pour l'explication de l'algorithme, on présume que cela a déjà été validé.

¹ Il est possible que vous n'ayez pas vu encore cette théorie mais ce n'est pas grave pour l'instant.

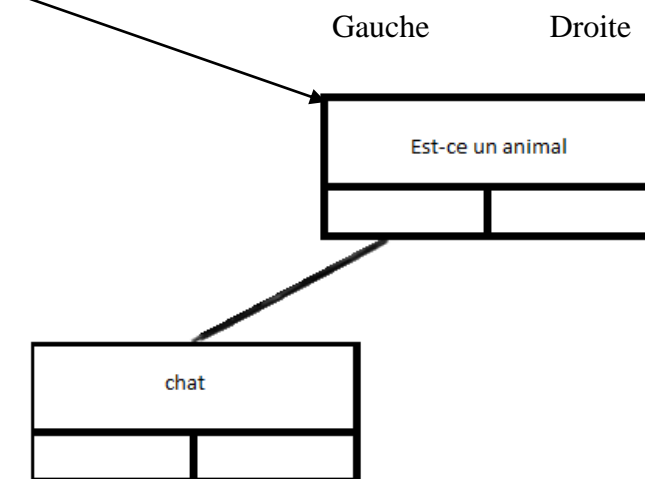
Dans ces conditions, il n'y a que quatre cas possibles à gérer dans le cadre du jeu:

1) La bd est vide

Dans ce cas, il faut d'abord créer deux nœuds. Un avec la question et l'autre, à sa gauche, avec la réponse. Toutes les autres références sont nulles. On retient la référence du premier nœud (tête de l'arbre) selon le principe du chaînage. Prenez note qu'à chaque fois que l'utilisateur entre une nouvelle réponse et sa question, il faut créer deux nœuds {réponse et question}.

Voici un exemple de la structure initiale résultant de l'initialisation, le processus par lequel la paire question/réponse initiale est entrée est expliqué plus loin.

premierNoeud



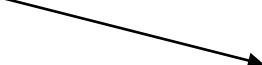
2) La dernière réponse de l'utilisateur est négative et il n'y a pas de nœud à droite de la question.

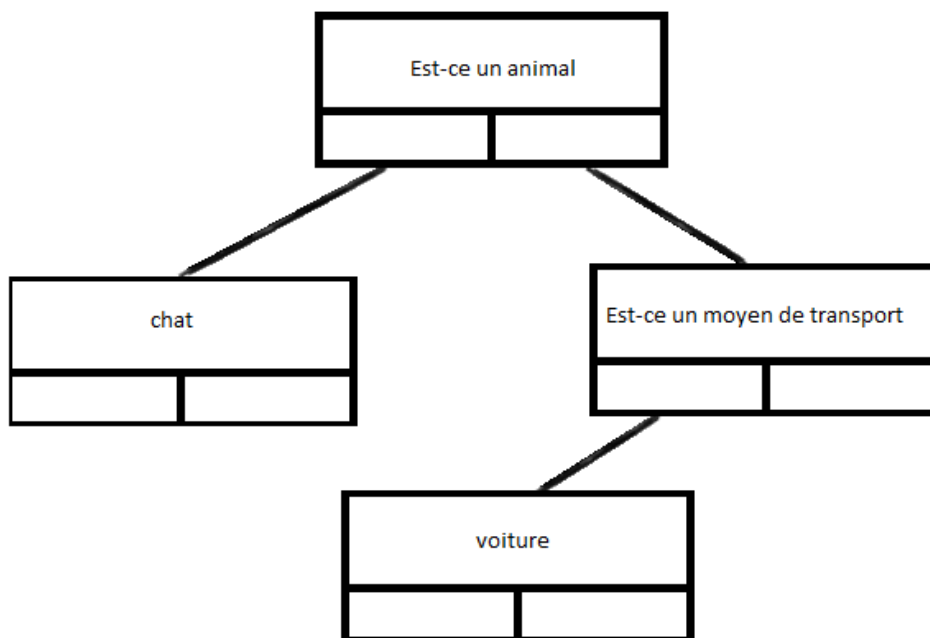
Quand le jeu aboutit à cette référence vide, c'est que le jeu ignore la réponse. À ce moment, il faut compléter la base de données. Pour cela, il faut créer les nœuds avec la nouvelle question et sa réponse à gauche, puis les référencer à droite du nœud qui contient la question à laquelle l'utilisateur a répondu par la négative.

Prenons l'exemple précédent « Est-ce un animal », est posée à l'utilisateur. S'il répond non, on s'aperçoit qu'à droite il n'y a pas de réponse. On lui demande sa réponse ainsi qu'une question pour laquelle on peut répondre par l'affirmative et on l'ajoute à l'arbre. Imaginons que l'utilisateur entre « voiture » et « Est-ce un moyen de transport ».

Voici le nouvel arbre de connaissance, qui a été actualisé avec la nouvelle question/référence.

premierNoeud





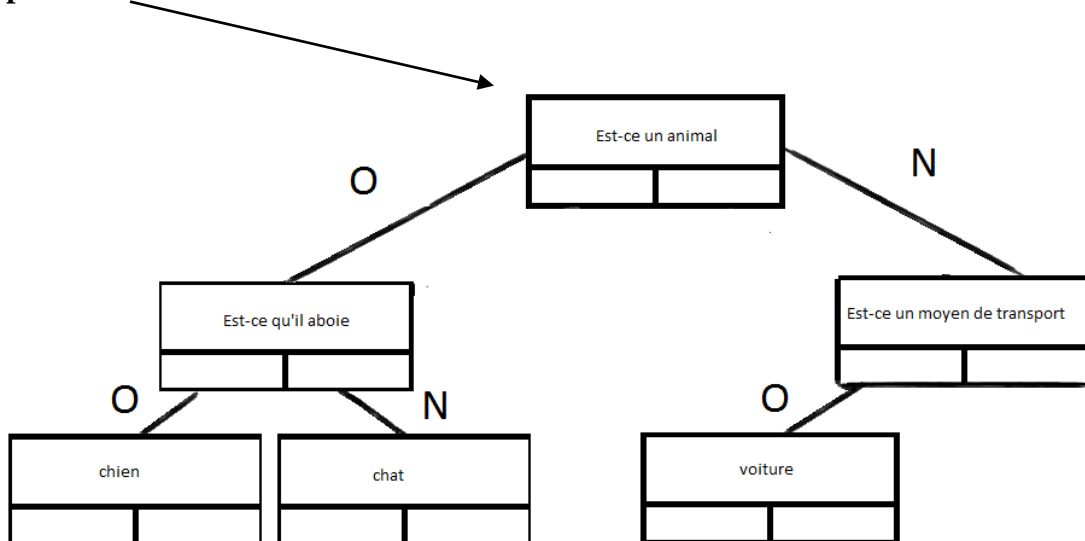
3) La dernière réponse de l'utilisateur est positive mais la réponse à gauche n'est pas la bonne réponse.

Dans ce cas, il faut créer les nœuds avec la nouvelle question et sa réponse à gauche. La réponse qui se trouve à gauche de la question posée à l'utilisateur (chat) devient la droite de la nouvelle question. La nouvelle question est mise à gauche du nœud qui contient la question à laquelle l'utilisateur a répondu par l'affirmative.

Prenons l'exemple précédent où la question, « Est-ce un animal ? » est posée à l'utilisateur. S'il a répondu oui à la question mais non à « c'est un chat », on lui demande sa réponse et une question pour laquelle on peut répondre par l'affirmative. Imaginons que l'utilisateur entre « chien » et « Est-ce qu'il aboie ».

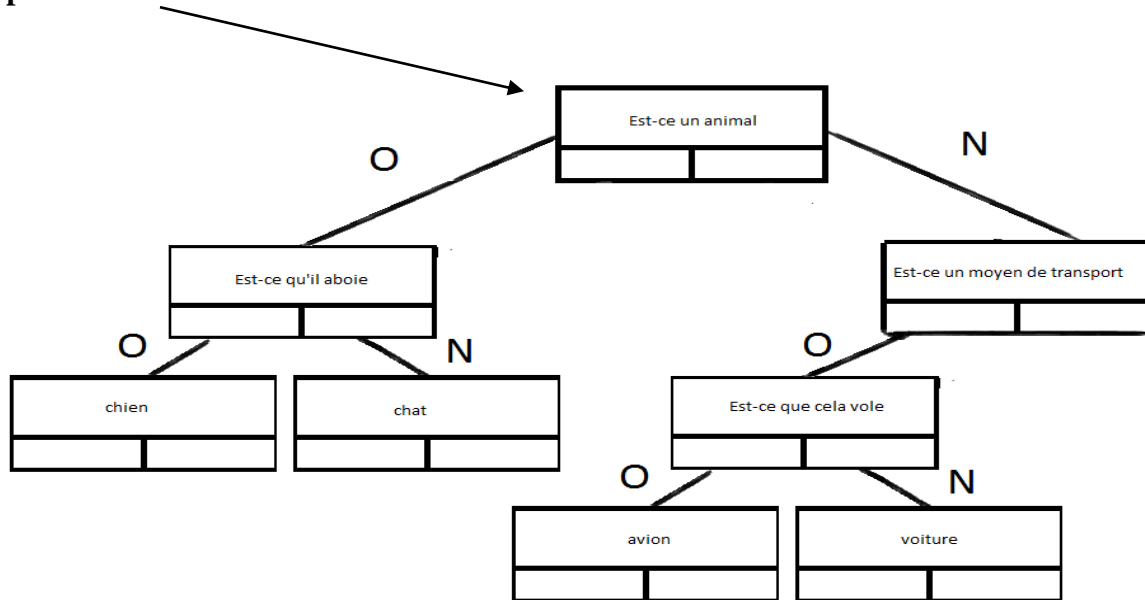
Voici le nouvel arbre de connaissance, notez l'insertion de la nouvelle question/réponse, à gauche.

premierNoeud



Autre exemple, prenons ce nouvel arbre et imaginons que l'utilisateur rejoue. Il répond 'N' à la première question et 'O' à la deuxième mais « voiture » n'est pas sa réponse. Il entre une nouvelle réponse et sa question. Imaginons « avion » et « Est-ce que cela vole ». Voici le nouvel arbre de connaissance.

premierNoeud



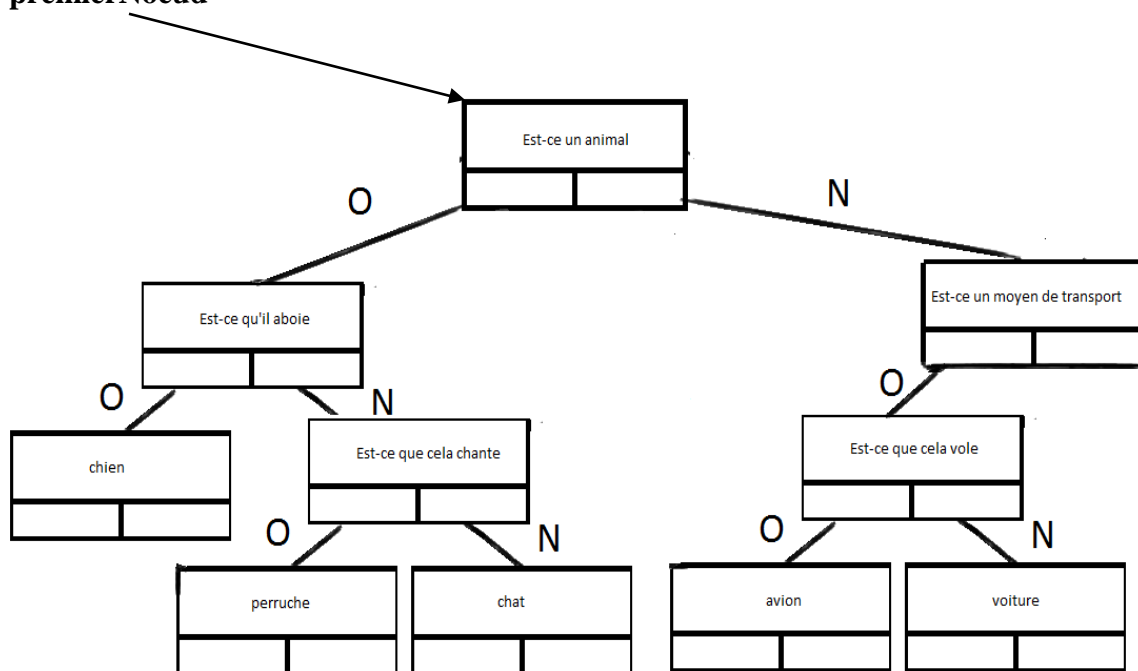
4) La dernière réponse de l'utilisateur est négative mais la réponse à droite n'est pas la bonne réponse.

Dans ce cas, il faut créer les nœuds avec la nouvelle question et la réponse à sa gauche. La réponse qui se trouve à droite de la question posée à l'utilisateur devient la droite de la nouvelle question. La nouvelle question est mise à droite du nœud qui contient la question à laquelle l'utilisateur a répondu par la négative.

Prenons l'exemple précédent où l'utilisateur répond 'O' à la première question et 'N' à la deuxième mais « chat » n'est pas encore sa réponse. Il entre une nouvelle réponse et sa question. Imaginons « perruche » et « Est-ce que cela chante ».

Voici le nouvel arbre de connaissance.

premierNoeud



Le déroulement du jeu revient à se promener dans cet arbre et de poser les questions rencontrées en allant à gauche ou à droite selon les indices de l'utilisateur (Oui ou Non). On parcourt jusqu'à ce qu'on rencontre une réponse ou qu'il n'y a plus de question à poser.

Si les deux références du nœud courant sont nulles, c'est alors la réponse que l'on propose à l'utilisateur et s'il dit que ce n'est pas exact, on ne sait donc pas à quoi l'utilisateur pense.

Développement de la première partie du travail

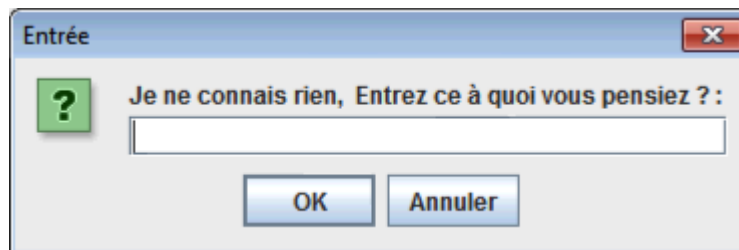
Modules fournis (Prenez en connaissance avant de commencer).

- **UtilitaireFichier** : Ce module contient le code nécessaire à l'obtention et à la sauvegarde de la bd.
- **Constantes** : Contient quelques définitions de constantes dont le nom de fichier.
- **DemarrerJeuDivinateur** : Contient le programme principal qui s'occupe d'obtenir la bd et de démarrer le jeu du divinateur tant que l'utilisateur veut jouer. La présentation du jeu et le message de fin sont en commentaires le temps de l'implémentation de votre travail.
- **UtilitaireES** : En plus de la procédure de présentation du jeu, il contient la procédure principale du jeu et des interactions avec l'utilisateur et la gestion des différentes possibilités d'annulation. Il utilise l'objet de la classe **BdQuestionsReponses** reçu en paramètre. Vous devez le compléter en Écrivant la procédure manquante décrite plus loin.
- **BdQuestionsReponses** Un objet de cette classe est instancié dans le programme principal. Comme pour le premier travail, vous devez écrire les entêtes des sous-programmes appelés mais manquants et un corps vide ({}). Vous les développez au fur et à mesure qu'une fonctionnalité est réalisée.

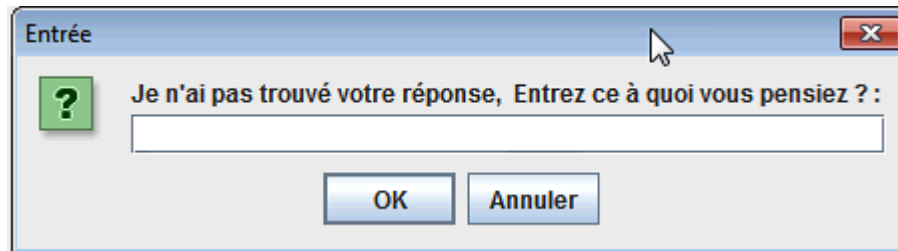
Algorithme de demanderReponseValide() du module UtilitaireES.

Cette procédure est appelée chaque fois que l'utilisateur doit entrer une réponse.

Si la bd est vide, **vous avouez ne rien connaître** et vous demandez une première réponse et une question associée pour laquelle il faut répondre dans l'affirmative.

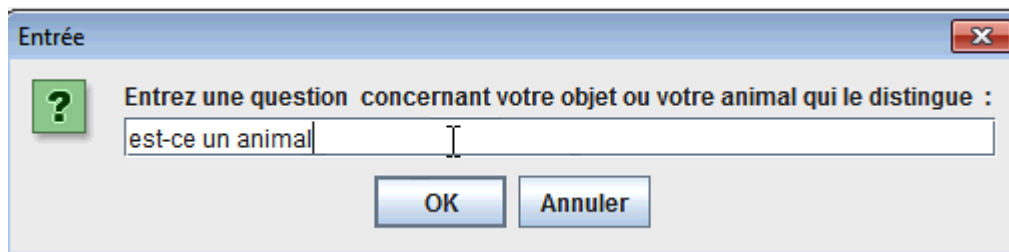


Sinon vous **avouez ne pas trouver** et vous demandez la réponse de l'utilisateur que vous convertissez en minuscule (les messages peuvent varier selon votre style).



Une fois la nouvelle réponse entrée, validez si la réponse entrée existe dans la bd. Si elle existe, demandez à la bd de vous retourner une chaîne de caractères qui montre l'erreur et affichez la (nous y revenons sur ce point plus loin, en attendant, vous pouvez simplement afficher un message d'avis).

Si la bd n'est pas vide **et** que l'utilisateur clique sur ok **et que la réponse n'existe pas**, vous demandez la question associée à sa réponse. Les chaînes vides ("") sont ignorées et non ajoutées dans la bd.



Si l'utilisateur clique sur ok et que la chaîne saisie n'est pas vide, vous mettez la question en minuscule et vous l'ajoutez à la bd à l'aide de la méthode prévue décrite plus loin.

Dans le cas où l'utilisateur clique sur Annuler, la procédure se termine et rien n'est ajouté à la base de donnée.

Note : On ne peut pas valider la pertinence de la question mais nous gérons la casse des caractères (case-sensitive). Peu importe comment la question est entrée, il n'y a que la version de la chaîne en minuscule qui est conservée dans la bd (Plusieurs chaînes ont la même version minuscule de la précédente : Est-cE UN anIMaL, EST_CE UN animal, est-ce un ANIMAL, ...). Pas besoin de s'occuper des accents. Par exemple, les questions « est-ce qu'on peut la levée » et « est-ce qu'on peut la levee » sont considérées comme étant différentes.

Le nœud de l'arbre

Nous avons mentionné qu'une réponse ne se retrouve qu'une fois dans la bd mais qu'en est-il des questions ? Il est possible que la même question serve à distinguer plusieurs réponses. Pensez à une question du genre « Est-ce que cela est rouge » qui peut s'appliquer autant à la couleur des yeux d'un gars saoul qu'au nez d'un renne connu.

Il est donc acceptable d'avoir plusieurs fois la même question. Pour éviter la répétition des questions que nous avons déjà dans une collection, nous allons seulement conserver la position de la question dans la collection. Il en est de même pour les réponses, on ne retient que sa position dans la collection des réponses déjà définies dans **BdQuestionsReponses**.

Votre mandat est d'écrire une classe qui représente un nœud de l'arbre tel que décrit. Il contient trois attributs qui sont : un entier et deux références soit une vers le nœud de gauche et l'autre vers le nœud de droite. Les attributs doivent **obligatoirement** être **privés**. Vous découvrirez à quel moment et comment les utiliser plus loin dans le document.

Note : Le constructeur par défaut n'a aucun sens alors vous n'en mettez pas. Question de sauver un peu de temps, nous vous faisons grâce également du constructeur par copie d'objet et des méthodes clone() et equals() qui ne sont pas utiles dans ce projet. Autrement dit, vous n'avez que le constructeur par copie d'attributs, les accesseurs et les mutateurs à écrire.

InfoJeu

Nous allons regrouper tout ce qui est utile pour la stratégie de parcours de l'arbre dans une structure de type enregistrement (champ à accès publique). Encore ici c'est pour sauver un peu de temps dans l'écriture du code. De plus, la structure d'arbre qu'on utilise est un TDA existant et connu avec lequel il vient beaucoup de théories de base mathématiques que nous n'avons pas besoin de connaître pour ce projet.

Les champs de ce type-enregistrement sont :

- Une référence sur le premier nœud de l'arbre.
- Deux références qui serviront au parcours de l'arbre :
 - Une référence sur un nœud courant.
 - Une référence sur le nœud précédent au nœud courant. Cette dernière est nulle si le nœud courant est le premier nœud.
- Un booléen pour retenir si le dernier indice fourni de l'utilisateur à une **question de la bd** était positive ou non. Ce booléen est nécessaire pour savoir où lier les nœuds (gauche ou droite).

À noter :

- Lorsqu'on demande si la réponse que le jeu propose est exacte, ne compte pas comme un indice à une **question de la bd**.
- Il n'y a qu'une variable du type infoJeu dans la classe **BdQuestionsReponses**.

Fin de la première partie

Dû à la procrastination du travail numéro 1 que nous voulons éviter, vous devez développer le code dans l'ordre suggéré et valider la première partie avec votre **responsable du labo** au plus tard dans la semaine du **7 au 10 mars** pendant la séance de laboratoire. Après cela, les validations n'auront plus lieu et aucune question qui concerne cette partie n'aura de réponse. En d'autres mots, il n'est plus possible d'attendre l'avant-veille de la remise pour commencer à travailler et espérer de l'aide.

Vous devez montrer vos classes **UtilitaireES**, celle qui représente un nœud de l'arbre et votre type-enregistrement **InfoJeu**. Les attributs (décrits page 2) et les coquilles vides doivent être écrits dans **BdQuestionsReponses**. Évidemment, le tout bien commenté avec le respect des bonnes pratiques exigées habituellement.

Partie 2

BdQuestionsReponses

C'est dans la classe **BdQuestionsReponses** que tout le travail qui simule l'intelligence du jeu est implémenté. Vous avez plusieurs méthodes à écrire dû à l'encapsulation des attributs. Vous reconnaîtrez l'utilité d'initialiser la bd, de répondre à « est-ce que la bd est vide ? », d'ajouter une réponse et sa question, quelle est la question à laquelle l'utilisateur a mal répondu en donnant ses indices (O ou N),

- **Initialiser la bd** : Il suffit de remettre tous les attributs dans le même état que lors de l'instanciation (appel au constructeur).
- **La bd est-elle vide** : On peut dire que la bd est vide s'il n'y a aucune question dans la collection de questions ou des réponses.
- **Choisir la première question** : Cette procédure place la référence nœud courant du type-enregistrement **InfoJeu** sur le premier nœud. La dernière réponse de l'utilisateur est mise à faux et la référence sur le nœud précédent est mise à nulle. Nous ajoutons à cela une liste instanciée mais vide pour recueillir les indices donnés par l'utilisateur. Elle est utilisée au moment de trouver une erreur de l'utilisateur. Il faut retenir les 'O' et 'N' de l'utilisateur. Vous pouvez utiliser les constantes définies et prévues à cet effet.
- **Une réponse est trouvée** : Retourne si les deux références du nœud courant sont nulles. Cela veut dire que c'est une réponse.
- **Obtenir la chaîne actuelle** : Retourne la chaîne associée au nœud courant. Il faut prendre l'indice et l'utiliser dans la bonne collection selon que c'est une question ou une réponse.

- **Obtenir si une réponse existe déjà :** Il est nécessaire de vérifier si la réponse existe déjà dans la collection. Cette fonction retourne si une chaîne de caractères reçue existe déjà dans la collection des réponses.

- **Ajouter une réponse et sa question :** Cette procédure reçoit deux chaînes de caractères. Elle ajoute la question dans sa collection, ajoute la réponse dans sa collection (avec une liste vide d'indices au départ), ajuste les nœuds de l'arbre et la liste des indices donnés si cela s'applique.

En pseudocode :

Début

Sauvegardez la bd dans un fichier de sécurité (un autre nom).

Si la question existe dans la collection

Vous retenez sa position.

Sinon

Vous retenez la position de la nouvelle question après l'avoir ajouté dans la collection.

En utilisant la position retenue, **créez les nœuds et ajustez l'arbre** tel que décrit antérieurement (le pseudocode vous est fourni dans la description suivante).

Sauvegardez la bd.

Fin

Note : Sauvegarder la bd dans un deuxième fichier avant de la modifier permet de travailler avec un fichier valide pour recommencer si vous avez des bogues pendant le développement. Après une exécution erronée, vous pouvez remplacer le fichier original par la copie de sécurité et reprendre où vous en étiez sans avoir à entrer à nouveau toutes les questions et les réponses antérieures qui sont valides.

- **Créez les nœuds et ajustez l'arbre :** Revoici les étapes pour créer les nœuds et les liés correctement ensembles mais en pseudocode. Idéalement, il faut avoir accès à l'indice de la question retenue lors de l'ajout de la question et à la réponse à ajouter (incluant la liste d'indices) et sa position dans la collection de réponse.

En pseudocode :

Début

Il faut l'indice de la réponse à ajouter et l'indice de la question.

On crée un premier nœud avec l'indice de la réponse et deux références nulles.

Si la bd est vide, on crée un nœud avec l'indice de la question et on lie le premier nœud de la réponse à sa gauche en y mettant sa référence.

Sinon

Il faut obtenir la réponse du nœud courant (pour modifier sa liste d'indices lorsqu'il change de côté).

On se fait une copie de la liste d'indices donnés et on ajoute 'N' à la fin. (elle remplace la liste de la réponse si elle change de côté).

Si la dernière réponse de l'utilisateur est positive (dans infoJeu)

On crée un nœud avec l'indice de la question, on lie la réponse à sa gauche et le nœud courant s'en va à sa droite.

Ce nouveau nœud s'en va à gauche du nœud précédent (on est certain qu'il n'est pas **null**).

On met la copie de la liste dans la réponse du nœud courant que nous avons obtenue (parce que la réponse a changé de côté).

Sinon si la réponse est négative et que le nœud courant est une question, on lie le nouveau nœud à la droite du nœud précédent

Sinon, c'est qu'on est sur une question et il n'y a rien à droite alors on lie le nouveau nœud à gauche du précédent et on met la copie de la liste comme étant la liste des indices donnés pour la réponse du nœud courant obtenue.

Fin du bloc

Fin du premier si

Il reste à ajouter un 'O' à la liste d'indices donnés originale et la fournir à la réponse qui a été ajoutée.

Fin

Note : À la fin, l'arbre est créé ou maintenu à jour et chaque réponse conserve la liste d'indices qui sont nécessaires pour la retrouver. La liste d'indices donnés créée durant un tour du jeu est utilisée aussi pour retrouver une erreur de l'utilisateur décrit plus loin. Il ne faut pas la détruire.

- **Se déplacer dans l'arbre :** Cette procédure reçoit l'indice de l'utilisateur (O ou N), déplace les références de l'attribut de type InfoJeu, ajuste l'attribut booléen selon l'indice reçu et ajoute le bon caractère à la liste des indices donnés (elle est utilisée par **UtilitaireES.demarrerDivinateur**).

En pseudocode :

Début

Si la réponse est oui (on va à gauche)

Retenez l'actuel nœud courant comme étant le nœud précédent.
Déplacez le nœud courant vers la gauche.
Retenez que la dernière réponse est positive.
Ajoutez le caractère à la liste des indices donnés.

Sinon si la réponse est non et qu'il y a un nœud à droite du nœud courant
(on va à droite)

Retenez l'actuel nœud courant comme étant le nœud précédent.
Déplacez le nœud courant vers la gauche.
Retenez que la dernière réponse est négative.
Ajoutez le caractère à la liste des indices donnés.

Sinon (on ne va nulle part)

Retenez que la dernière réponse est négative.
Il ne reste plus de question à poser.

Fin

Retournez s'il reste des questions

Fin

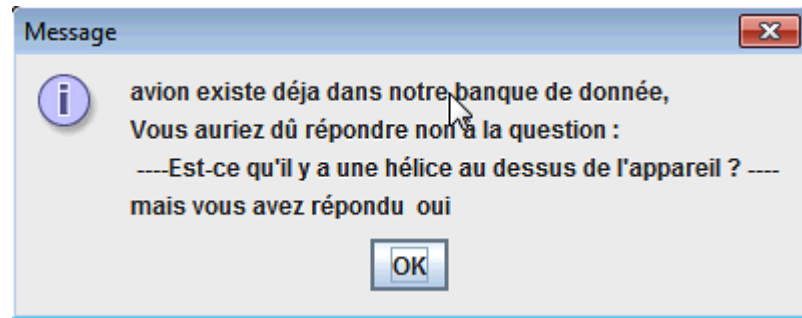
*** Vous devez être en mesure de jouer sans toutefois pouvoir détecter les erreurs.

- **L'utilisateur s'est trompé.**

Écrivez une fonction qui retourne une chaîne de caractères correspondant au déroulement du jeu. Vous devez remplacer le message temporaire que vous avez mis lorsque vous demandez une réponse valide dans le module UtilitaireES. La chaîne à construire a la forme suivante :

| |
|---|
| _____ existe déjà dans notre base de données, Vous auriez dû répondre _____ à la question : ---- _____ ? ---- mais vous avez répondu _____ |
|---|

Exemple :



C'est le temps d'utiliser la liste d'indices donnés de la réponse du nœud courant et la comparer avec la liste d'indices donnés tout au long du tour de jeu. À chaque fois que les réponses à la même position dans la liste sont identiques, on déplace une référence temporaire utilisée pour parcourir l'arbre. Encore, si la valeur de l'indice est Constantes.REPONSE_POSITIVE, on déplace la référence à gauche sinon, on le déplace à droite. Aussitôt qu'un des indices est différent dans les deux listes, on sait à quelle question l'utilisateur s'est trompé grâce à cette référence.

En pseudocode :

Début

On se positionne au début de la liste d'indices donnés de la bd.

On crée une copie de la liste.

On obtient la réponse que l'utilisateur a donné qui se trouve dans la bd pour accéder à sa liste d'indices.

On obtient le premier nœud de l'arbre dans une variable temporaire pour le parcours de l'arbre sans déplacer le nœud courant.

On obtient un caractère de chaque liste et on les compare tant qu'il y en a et qu'ils sont identiques. Dans cette boucle, on déplace la variable temporaire à gauche ou à droite selon l'indice.

En sortant de la boucle, nous avons l'indice fautif et la variable temporaire qui est sur le nœud contenant la question qui avait été posée. Il reste à concaténer la chaîne à retourner.

Fin

NOTE : Nous savons qu'il est impossible de se retrouver ici si les indices de l'utilisateur étaient tous bons parce que cela est détecté dans la procédure qui demande une réponse valide du module **UtilitaireES** que vous devez avoir écrit.

Considération additionnelle

Si vous voyez que vous n'y arriverez pas, vous pouvez consulter votre responsable du labo (par courriel aussi). N'attendez pas à la dernière minute pour vous apercevoir que vous n'y arrivez pas. Commencez tôt!!! Si vous arrivez à la troisième semaine avec des questions de la première partie, vous n'aurez pas de réponse.

13. Barème de correction

La moitié des points est accordée pour la bonne exécution du jeu.

Le total enlevé pour les mauvaises pratiques de programmation peut dépasser 50% des points. Les mêmes critères que pour le tp1 s'appliquent auxquels nous ajoutons :

- Écrire plusieurs appels au même accesseur inutilement. Mettre la valeur obtenue de l'accesseur dans une variable et l'utiliser à la place (-0.5 par appel excédentaire).
- Non-respect de l'encapsulation (-10).
- Non utilisation des collections de Java (-20).
- Non utilisation de la liste présentée en classe (-20).

Il est facile pour vous de traduire tous les algorithmes avant de commencer à déboguer. NOUS VOUS INTERDISONS CETTE APPROCHE.

Vous devez résoudre une fonctionnalité complètement et qu'elle fonctionne **parfaitement** avant de vous attaquer à la suivante. Autrement dit, si le jeu plante et que vous avez écrit tout le code, c'est 0 pour la fonctionnalité et la correction s'arrête. Le reste du code est considéré comme étant non fait.

Bon travail!