Examen de Programmation Java

EISTI – ING1 2010-2011

9 juin 2011, durée : 3h

Conditions d'examen

L'architecture (nom des répertoires) de votre projet Java est imposée de la manière suivante :

- **rendujava**
  - **sources** (les sources Java)
  - **production** (tout production issue des sources)
    - **bytecode** (les .class)
    - **exe** (les jars exécutables)
  - **build.xml** (le fichier de pilotage du projet)

Le répertoire **rendujava** doit être à la racine de votre compte pour assurer le rendu automatique. Merci d'effacer le répertoire de production avant de quitter l'examen.

La javadoc de l'API 1.6 est disponible dans le répertoire **/usr/share/doc/sun-java6-jdk/html**.

Rappel du barème des pénalités (pour chaque classe Java écrite) :

Code non compilable	-50%
Norme de programmation Java non respectée	-10%
Warnings (annotations @deprecated ou @SuppressWarnings interdites)	-25%
Non respect des consignes (noms de classe, de package, et des fichiers déposés)	-10%
Commentaires non obligatoires	cf question B.1.j

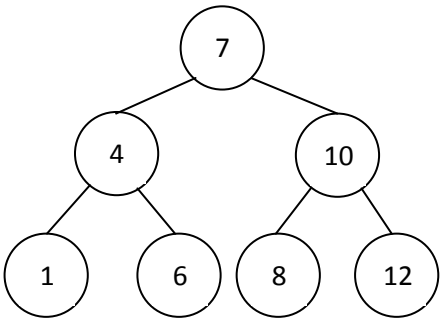
Introduction du sujet

On cherche à réaliser un arbre binaire de recherche (ABR) avec des valeurs génériques. Les valeurs d'un ABR possèdent forcément un ordre qui permet de comparer et de placer les valeurs dans l'ABR. On rappelle qu'un ABR est défini comme suit :

- un ABR est un arbre binaire : chaque nœud de l'arbre a au plus 2 fils ;
- la valeur d'un nœud est plus grande que toutes les valeurs du sous-arbre gauche ;
- la valeur d'un nœud est plus petite que toutes les valeurs du sous-arbre droit.

Les valeurs d'un ABR *possèdent donc forcément un ordre* qui permet de les comparer.

Exemple d'un ABR sur les entiers avec l'ordre naturel ascendant :

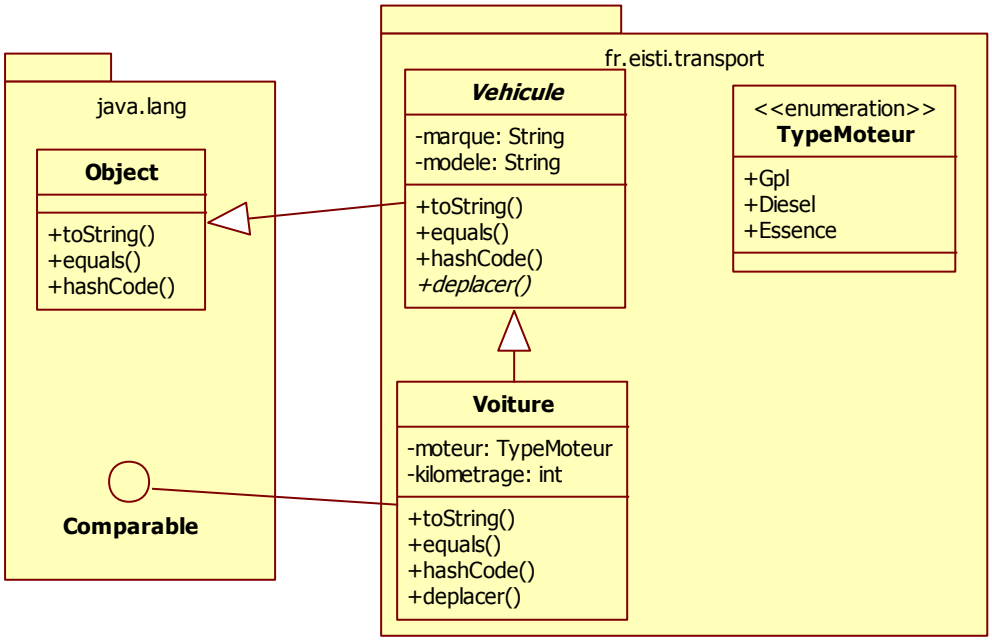


On vérifie bien que les valeurs 1, 4, 6 sont inférieures à 7, les valeurs 8, 10, 12 sont supérieures à 7 ; au niveau inférieur, on a bien $1 < 4 < 6$ et $8 < 10 < 12$.

Questions

A - Sujet d'application : véhicules et voitures

L'objectif final de l'examen sera de créer un ABR sur des voitures. La première partie concerne donc la définition d'une classe Voiture. Comme sujet d'application, on se propose de faire un ABR sur des voitures. Le diagramme UML ci-dessous nous servira de guide. Notez que les accesseurs, les paramètres des méthodes, les paramètres de généricité et les méthodes des interfaces connues de l'API – définition et implémentation – ne sont pas indiqués.

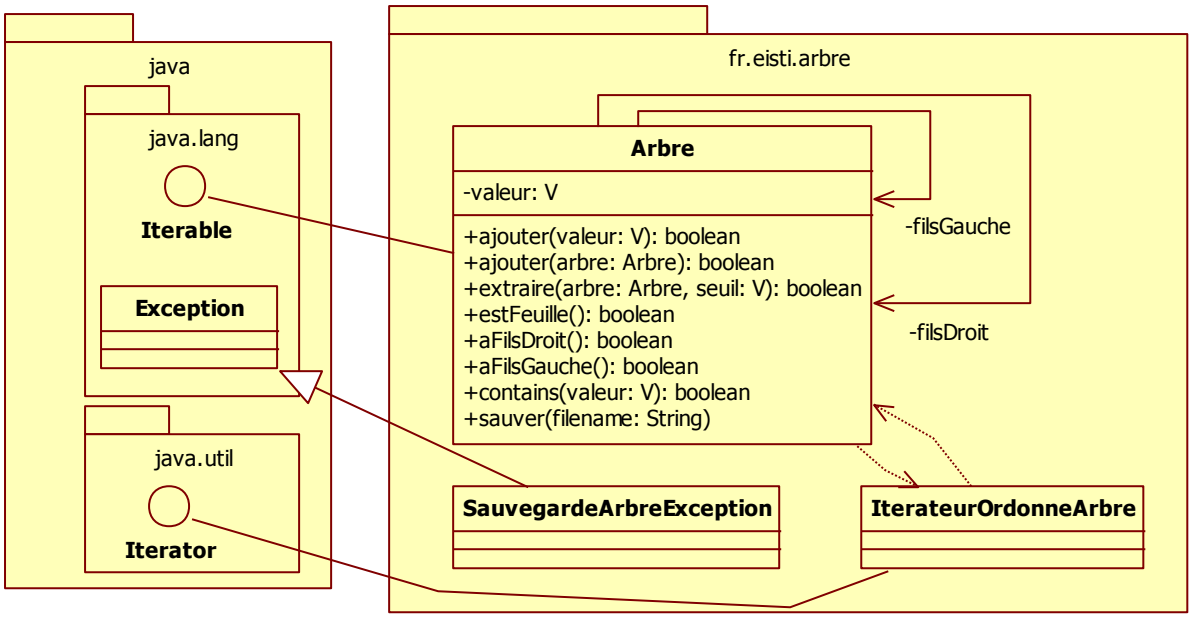


Les indications supplémentaires suivantes vous sont données :

1. classe Véhicule :
    a. écrire un constructeur complet initialisant les 2 propriétés et des accesseurs en lecture ;
    b. 2 véhicules sont égaux si leur marque et leur modèle sont égaux ;
    c. le hashCode d'un véhicule est égal à la somme des hashCodes de sa marque et de son modèle ;
    d. la représentation textuelle d'un véhicule est comme suit : marque modèle.
2. classe Voiture :
    a. écrire un constructeur complet initialisant les 4 propriétés et des accesseurs en lecture ;
    b. deux voitures sont égales si elles sont égales en tant que véhicule et si leur moteur et leur kilométrages sont égaux ;
    c. le hashCode d'une voiture est égal à la somme de son kilométrage, du hashCode de son type de moteur et du hashCode de la voiture en tant que véhicule ;
    d. la représentation textuelle d'une voiture est comme suit : *marque modèle typeDeMoteur kilométrage* km;
    e. la méthode déplacer incrémente le kilométrage d'une unité ;
    f. l'ordre sur les voitures est défini sur le kilométrage, puis en cas d'égalité sur le type de moteur (Gpl < Diesel < Essence) ; en cas de nouvelle égalité, cet ordre considère les voitures identiques (inconsistance avec la méthode equals).

B - Arbre binaire de recherche générique

La deuxième étape consiste à écrire les classes pour manipuler des arbres binaires de recherche dont les valeurs sont génériques. Le diagramme UML suivant nous servira de guide. Notez que les accesseurs, les paramètres de généricité et les méthodes des interfaces connues de l'API – définition et implémentation – ne sont pas indiqués.



Les indications supplémentaires suivantes vous sont données :

1. **classe Arbre :**
    a. cette classe permet de représenter à la fois un nœud de l'arbre et l'arbre en entier à travers son nœud racine ;
    b. une valeur doit respecter le contrat java.lang.Comparable apportant l'ordre des éléments de l'arbre ;
    c. écrire un constructeur initialisant la valeur du nœud racine de l'arbre à une valeur passée en paramètre ; si la référence est nulle l'exception java.lang.NullPointerException sera lévée avec un message indiquant le problème ;
    d. écrire les accesseurs en lecture pour la valeur du nœud racine, le sous-arbre gauche et le sous arbre droit ;
    e. les fonctions estFeuille, aFilsDroit et aFilsGauche permettent de connaître l'état d'un nœud de l'arbre ;
    f. la méthode qui ajoute une valeur dans l'arbre :
        - vérifie que la valeur n'est pas une référence nulle (exception java.lang.NullPointerException sinon)
        - compare la valeur en paramètre avec la valeur courante :
            - si l'ordre est identique, n'ajoute pas la valeur et renvoie faux
            - si le paramètre est plus petit, insère la valeur à gauche (récursivement s'il y a déjà un fils gauche)
            - si le paramètre est plus grand, insère la valeur à droite (récursivement s'il y a déjà un fils droit)
    g. la méthode contains cherche une valeur dans l'arbre :
        - si la valeur est une référence nulle, renvoie faux ;
        - compare la valeur en paramètre avec la valeur courante :
            - si l'ordre est identique, renvoie vrai ;
            - si le paramètre est plus petit, recherche récursivement à gauche si possible et sinon renvoie faux ; (même chose à droite si plus grand)
    h. la méthode sauver permet d'enregistrer l'arbre en tant qu'objet Java dans un fichier dont le nom est passé en paramètre ; en cas de problème de sauvegarde, une exception SauvegardeArbreException est levée ; cette dernière encapsulera toute exception qui aurait pu intervenir en interne dans la fonction ;
    i. l'itération d'un arbre se fait via un IterateurOrdonneArbre (cf ci-dessous)

j. commenter l'entête de la classe, les attributs, le constructeur et la méthode ajouter une valeur en utilisant au maximum les balises Javadoc ;
k. les méthodes ajouter un arbre et extraire vers un arbre ne sont pas à coder ; vous donnerez seulement leur signature (vous pouvez renvoyer la valeur faux pour faire plaisir au compilateur) :
   - la méthode ajouter permet d'ajouter un sous-arbre avec des valeurs du même type générique ou d'un type plus spécialisé ;
   - la méthode extraire permet d'ajouter dans l'arbre passé en paramètre toutes les valeurs de l'arbre courant qui sont plus grandes qu'un certain seuil ; l'arbre destinataire contient des valeurs du même type que l'arbre courant ou d'un type plus général.

2. classe IterateurOrdonneArbre :

a. le constructeur prend en paramètre l'arbre a parcourir et crée une file des éléments à parcourir ; le parcours suit l'ordre défini sur les éléments de l'arbre ; pour cela, écrire une fonction privée initialiserFile qui remplit récursivement la file comme suit :
   - enfiler les éléments à gauche (plus petits) ;
   - enfiler la valeur courante ;
   - enfiler les éléments à droite (plus grands) ;
b. implémenter les méthodes de l'itérateur (sauf remove – cf documentation) en défilant les éléments au fur et à mesure du parcours (on sort bien en 1er les plus petits éléments).

C - Application sur un ABR d'entiers et un ABR de voitures

1. Ecrire une **1ère application** qui manipule un ABR d'entiers :
   a. créer une liste des entiers de 1 à n avec n égal à 10 par défaut ou passé en ligne de commande (si le paramètre n'est pas un entier, on revient au paramètre par défaut) ;
   b. mélanger la liste aléatoirement ;
   c. ajouter toutes les valeurs dans un ABR d'entiers ;
   d. afficher dans l'ordre croissant les valeurs de l'ABR.
2. Ecrire une **2ème application** qui manipule un ABR de voitures :
   a. créer un l'ABR avec une 1ère voiture Fiat de modèle Palio avec un moteur GPL et 100000 km ;
   b. ajouter une voiture Renault de modèle Clio avec un moteur Essence de 198000 km ;
   c. ajouter une voiture Renault de modèle Clio avec un moteur Diesel de 105000 km ;
   d. ajouter une voiture Renault de modèle Clio avec un moteur Diesel de 85000 km ;
   e. ajouter une voiture Ferrari de modèle 599 GTO avec un moteur Essence de 210000 km ;
   f. afficher dans l'ordre croissant les valeurs de l'ABR.

D - Gestion automatisée de projet

Ecrire un fichier de gestion de projet Ant qui respecte la hiérarchie de fichiers proposés en introduction et propose les cibles suivantes :

- **compile** : création du répertoire de production et pour le bytecode puis compilation des classes du projet ;
- **jars** (cible par défaut) : compilation puis création du répertoire pour les exécutables et création de 2 jars exécutables :
  - testInteger.jar pour la 1ère application ;
  - testVoiture.jar pour la 2ème application ;
- **clean** : supprime le répertoire contenant le bytecode ;
- **cleanall** : supprime tout sauf les sources et le fichier de pilotage Ant.