Examen de Programmation Java – ING1 EISTI 2008-2009

Répondre aux différentes questions de cours dans un fichier **reponses.txt**. Le code Java demandé devra être livré dans un répertoire src. Pour les questions de cours, il n'est pas demandé de code Java ; tout temps perdu à écrire du code pour vérifier ses réponses le sera aux détriments des autres exercices. Répondre au fur et à mesure au dernier exercice (ant) pour ne pas vous faire piéger par le temps.

Déposer vos réponses, vos sources Java et le fichier ant dans le répertoire **rendu-java**.

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

Rappel du barême des pénalités (à ramener au prorata de chaque exercice) :

- code non compilable (cf exercice 4) : -10 (moitié de la note)
- warning (annotation @deprecated interdite) : -2 (un 10ème)
- norme de programmation Java non respectée : -5 (un quart)
- pas de commentaires : -5 (un quart)

NB: dans les exercices suivants, il n'est à aucun moment demandé un affichage optimisé d'une donnée de type couleur. On pourra utiliser la méthode toString() par défaut qui affiche les composantes rgb de la couleur.

Exercice 1 [Cours] – 1.5 points

Soient les types Java suivant :

a- class Tutu { ... }
b- abstract class Titi { ... }
c- interface Toto { ... }
d- enum Tata { ... }

1/ Est-ce qu'on peut déclarer une référence de chacun de ces types ?
a- oui/non
b- oui/non
c- oui/non
d- oui/non

2/ Est-ce qu'on peut avoir un constructeur dans chacun de ces types ?
a- oui/non
b- oui/non
c- oui/non
d- oui/non

3/ Est-ce qu'on peut faire un new sur chacun de ces types ?

a- oui/non
b- oui/non
c- oui/non
d- oui/non

Exercice 2 – Des Voitures – 5 points

2.1 [Java] - 1 point

Ecrire la classe **Voiture** avec les champs privés suivants :

- couleur : Color
- marque : String
- modele : String

La classe Color est une classe de l'API Java. Ajouter un constructeur initialisant tous les champs avec les paramètres et les accesseurs en lecture et écriture des différents champs. La méthode **accelerer** se contentera d'afficher la trace suivante : “accélération d'une voiture normale” sur la console.

2.2 [Java] – 1 point

Ecrire la classe **VoitureSport** qui spécialise la classe **Voiture** avec :

- 1 champ privé **nombreChevaux** : int
- la méthode **accelerer** redéfinie

La voiture de sport doit disposer d'un constructeur initialisant tous ses champs à partir des paramètres et des différents accesseurs (lecture/écriture) pour ses champs. La méthode **accelerer** affiche la trace “accélération d'une voiture de sport” sur la console.

2.3 [Java] – 1 point

Ecrire une application **JeuVoiture** qui crée une voiture de couleur noire de marque Citronault et de modèle méga209 ainsi qu'une voiture de sport rouge de marque Ferrasche, de modèle F40 et 600 chevaux.

2.4 [Cours] – 2 points

Pour chacun des extraits de code suivants vous préciserez :

a) s'ils sont acceptés par le compilateur
b) si a) OK si l'exécution se passe bien
c) ci b) OK quelle est la trace affichée

Extrait 1:

```
Voiture v;
Voiture vs;
v = new Voiture(.....);
vs = v;
v.accelerer();
vs.accelerer();
```

Extrait 2:

```
Voiture v;
Voiture vs;
```

```
vs = new VoitureSport(.....);
v = vs;
v.accelerer();
vs.accelerer();
```

Extrait 3:

```
Voiture v;
Voiture vs;
v = new Voiture(.....);
vs = (VoitureSport) v;
v.accelerer();
vs.accelerer();
```

Extrait 4:

```
Voiture v;
Voiture vs;
vs = new VoitureSport(.....);
v = (Voiture) vs;
v.accelerer();
vs.accelerer();
```

Extrait 5:

```
Voiture v;
Voiture vs;
v = new VoitureSport(.....);
vs = v;
v.accelerer();
vs.accelerer();
```

Extrait 6:

```
Voiture v;
Voiture vs;
vs = new Voiture(.....);
v = vs;
v.accelerer();
vs.accelerer();
```

Extrait 7:

```
Voiture v;
Voiture vs;
v = new VoitureSport(.....);
vs = (VoitureSport) v;
v.accelerer();
vs.accelerer();
```

Extrait 8:

```
Voiture v;
Voiture vs;
vs = new Voiture(.....);
v = (Voiture) vs;
v.accelerer();
vs.accelerer();
```

Exercice 3 – Des Voitures à sérialiser – 5 points

3.1 [Java] – 3 points

Ecrire une application **SauveVoiture** avec :

- 1 méthode **static Voiture saisieVoiture()** qui demande à l'utilisateur de saisir la couleur, la marque et le modèle de la voiture et renvoie une nouvelle voiture possédant ses caractéristiques. Pour la couleur on demandera 4 entiers pour utiliser le constructeur **public Color(int r, int g, int b, int a)**. Les problèmes liés à la saisie d'une chaîne de caractère ne correspondant pas à un entier ou liés à un entier de valeur incorrecte provoqueront une nouvelle saisie de la part de l'utilisateur. Tout autre problème devra être propagé en dehors de la méthode sous la forme d'une exception SaisieVoitureException (à définir). Cette exception contiendra 2 informations : 1 message avec votre interprétation du problème survenu et l'exception Java à la source du problème.
- 1 méthode **static void sauverVoiture(String nomFichier,Voiture voiture, boolean append)** qui sérialise la voiture passé en paramètre dans le fichier **nomFichier**. Si **append** est vrai, la voiture est ajoutée en fin de fichier sinon le fichier est écrasé. Tout problème éventuel sera propagé en dehors de la méthode.
- 1 méthode principale qui demande la saisie de 2 voitures et les sauve dans le fichier voitures.obj. Tout problème éventuel devra être capté et un message d'erreur devra prévenir l'utilisateur de la nature du problème intervenu.

3.2 [Java] – 2 points

Ecrire une application **LireVoiture** avec :

- 1 méthode **static List<Voiture> lireVoiture(String nomFichier, nombreVoiture)** qui lit nombreVoiture voiture(s) sérialisées dans le fichier nomFichier et les renvoie sous la forme d'une liste. Tout problème éventuel de lecture sera propagé en dehors de la méthode.
- 1 méthode principale qui lit **n** voitures à partir d'un **nom de fichier**, ces 2 paramètres étant fournis en ligne de commande. Tout problème éventuel devra être capté et un message d'erreur devra prévenir l'utilisateur de la nature du problème intervenu.

Exercice 4 – Un concessionnaire de voitures – 6.5 points

4.1 [Java] – 5.5 points

Ecrire une classe **Concessionaire** qui aggrège des voitures dans une liste d'association faisant correspondre à une marque et un modèle (concaténation des 2 chaînes avec le séparateur '/') la liste des voitures disponibles pour ce concessionaires. Prévoir :

- 1 constructeur qui initialise la liste d'association
- 1 méthode **void ajouterVoiture(Voiture voiture)** qui permet d'ajouter une voiture à la liste d'association. Cette méthode doit vérifier l'existence de la clé correspondant à la voiture ajoutée. Si la clé est présente, la voiture est ajoutée dans la liste existante sinon on ajoute cette nouvelle clé associée à une nouvelle liste contenant uniquement la voiture ajoutée.
- 1 méthode **List<Voiture> rechercherVoitureMarqueModele(String marque, String modele)** qui renvoie la liste des voitures correspondant à la marque et au modèle passé en paramètre. Une liste vide sera renvoyée si la recherche est infructueuse.
- 1 méthode **List<Voiture> rechercherVoitureMarqueModeleCouleur(String marque, String modele, Color couleur)** qui renvoie la liste des voitures correspondant à la marque, au modèle et à la couleur recherchée. Une liste vide sera renvoyée si la recherche est infructueuse.
- 1 méthode **List<Voiture> rechercherVoitureCouleur(Color couleur)** qui renvoie la liste des

voitures correspondant à la couleur recherchée toute marque et couleur confondue. Une liste vide sera renvoyée si la recherche est infructueuse.

4.2 [Java] – 1 point

Ecrire une classe **TestConcessionaire** qui crée un concessionaire lui ajoute une dixaine de voitures (prévoir des marques+modèles à un exemplaire et d'autres à plusieurs exemplaires ; prendre 2 ou 3 couleurs différentes). Tester vos 3 fonctions de recherches avec un affichage sommaire du résultat des recherches.

Exercice 5 – Gestion du projet Voitures – 2 points

5.1 [Ant] Ecrire un fichier **build.xml** qui permet de compiler avec la règle par défaut l'ensemble de vos sources compilables (si vous avez une classe incomplète, l'exclure des fichiers à compiler). Vous ajouterez une règle d'exécution par application demandée (run2.3, run3.1, run3.2, run4.2), une règle produisant la javadoc (javadoc) et une règle de nettoyage (clean) supprimant le bytecode produit et la javadoc.

NB: pour passer des paramètres en ligne de commande via ant ajouter une sous-balise à la balise java comme suit :

```
<java .....>
        <arg line="mes paramètres"/>
</java>
```