TP Noté de Java SE
2011 – 2012

Conditions d'examen :

- Tous les documents sont autorisés, y-compris Internet.
- Le travail se fera par binôme
- Le devoir se fait sur machine et se rendra sur AREL : matière Programmation Java.

Note Importante : Votre rendu se fera sur **AREL dans le travail prévu à cet effet** et ne doit comporter qu'**un dossier src contenant vos sources Java ainsi qu'un fichier Ant,** le tout zippé. Tout autre type de rendu (zip du projet Eclipse, …) entrainera automatiquement une note nulle.

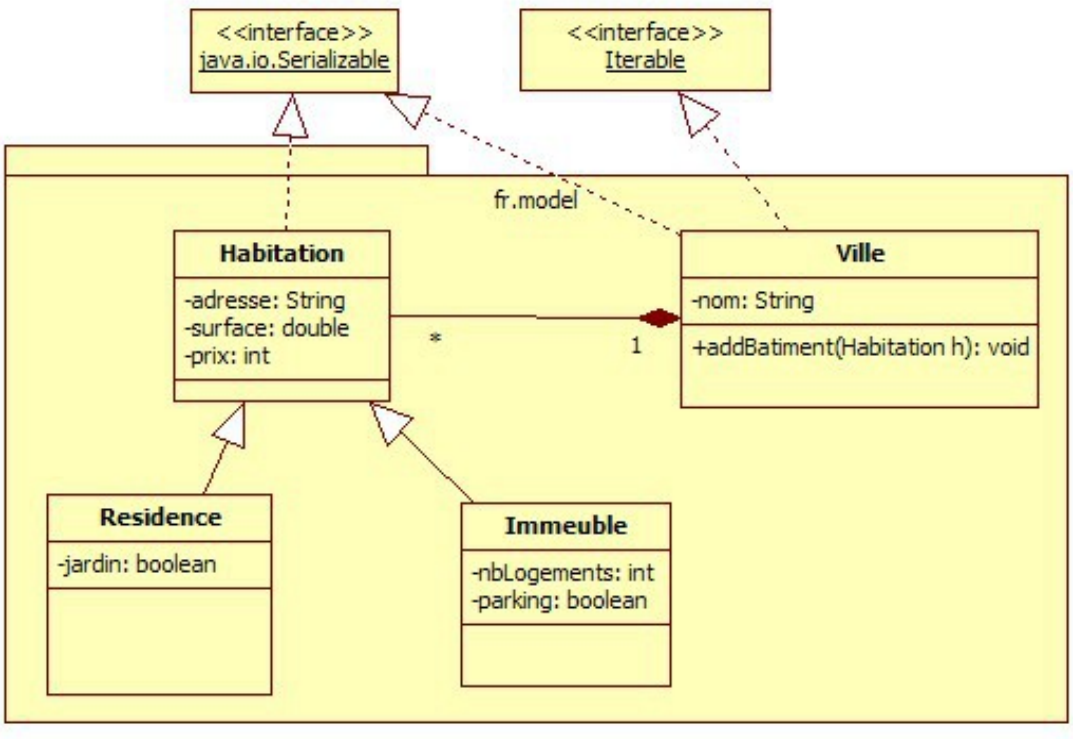
L'architecture de votre projet est imposé de la manière suivante (seules les entrées en gras font parties du rendu, le reste sera généré par le fichier Ant):

- **src (répertoire des sources Java)**
- bin (répertoire qui contiendra les classes)
- lib
  - jar (répertoire qui contiendra un Jar exécutable)
- **build.xml (le fichier Ant)**

Gestion Immobilière

Un client a besoin d'un outil de gestion de son parc immobilier. Il veut notamment avoir une fonctionnalité lui permettant d'extraire le logement ayant le meilleur rapport qualité/prix, ainsi qu'une fonctionnalité de sauvegarde binaire (et non textuelle comme c'est le cas actuellement).

Un équipe a déjà réalisé un diagramme de classe correspondant au modèle de l'outil, sur lequel vous devrez vous appuyez (exactement) pour réaliser l'application demandée.



Dans le diagramme de classe, on vous dit qu'une Ville représente en fait un ensemble d'habitations en vente, ces habitations pouvant être des Immeubles (avec plusieurs logements) ou des Résidences.

La méthode addBatiment permet d'ajouter une Habitation dans la Ville.

Le client vous fournit un jeu de données de test (**data.txt**) afin que vous puissiez réaliser votre application. Ce fichier contient les données pour une seule ville et est défini ainsi :

```
NomVille
TypeHabitation
Adresse
Surface
Prix
NbLogements // Uniquement si le type d'habitation est un Immeuble
Jardin ou Parking // En fonction du type d'habitation : Immeuble ou Résidence
TypeHabitation
Adresse
etc....
```

Question 1 [2 points] : Implémentez les classes correspondant au diagramme de classe, en négligeant dans un premier temps les 2 interfaces. Vous devez écrire les accesseurs en lecture mais aucun accesseur en écriture, ainsi que les constructeurs.

Question 2 [0.5 point] : Ecrivez la méthode toString de la classe Habitation, de façon à l'afficher de cette manière :

```
Adresse[Surface m²] : Prix €
```

Question 3 [0.5 points] : Faites de même pour les classes Immeuble et Résidence :

```
Résidence
Adresse[Surface m²] : Prix € (Avec Jardin)

Immeuble
Adresse[Surface m²] : Prix € (Sans Parking)
```

Question 4 : Créez un package “fr.controller“ contenant une classe Base avec :

- **Un attribut**

  ```
  public static Ville v;
  ```

  représentant la ville du fichier chargé (unique)
- **Une méthode [3 points]**

  ```
  public static void charger(String nomFic)
  ```

  permettant de créer l'objet static v à partir d'un fichier texte de données.
  => Vous ferez attention à gérer les exceptions pouvant survenir dans le cas d'un fichier erroné
- **Une méthode [3 points]**

  ```
  public static void serialiserDonnees(String nomFic)
  ```

  permettant de sérialiser l'objet static v dans un fichier binaire.
- **Une méthode [3 points]**

  ```
  public static void deserialiserDonnees(String nomFic)
  ```

  permettant de récupérer un objet sérialisé dans un fichier binaire et de l'affecter à l'objet static v.
- **Une méthode [2 points]**

  ```
  public static Habitation getRapport()
  ```

  permettant de récupérer l'Habitation la plus intéressante du jeu de données par rapport surface/prix.

=> Attention néanmoins pour les Immeubles, pour obtenir une estimation correcte de ce rapport il faut diviser la surface par le nombre de logements.

Question 5 : Créez un package “fr.view“ contenant une classe Menu avec :

- **Une méthode [2 points]**

```
private int menu()
```

Qui affiche le menu console suivant :

```
1 : Charger données texte
2 : Charger données binaires
3 : Afficher meilleur rapport qualite/prix
4 : Sauvegarder données en binaire
5 : Quitter
```

demande son choix à l'utilisateur et renvoie une valeur entière entre 1 et 5 (vous redemanderez à l'utilisateur d'entrer cette valeur si sa saisie est erronée.

- **Une méthode [2 points]**

```
public void show()
```

qui appelle la méthode menu(), regarde la réponse donnée par l'utilisateur, et appelle la méthode correspondante de la classe Base (éventuellement en demandant la saisie préalable par l'utilisateur de certaines informations).
Cette méthode boucle tant que le choix de l'utilisateur n'est pas égal à 5 (c'est à dire s'il veut quitter).

Question 6 [2 points] : Faites en sorte que la classe Ville implémente l'interface *Iterator*. Quel est l'intérêt? Modifiez votre code pour que cette modification soit utile.
=> Répondre sous forme de commentaire dans la classe Ville

Annexes

1. On vous donne la classe Executable finale (sans prendre en compte vos tests intermédiaires)**:**

```
public class Executable
{
	public static void main(String[] args)
	{
		Menu m = new Menu();
		m.show();
	}
}
```

2. Sérialisation

Vous pouvez regarder le cours pour comprendre le principe de la Sérialisation. Nous le résumons ici.

Sérialiser un objet signifie que l'on veut le stocker dans un fichier binaire, directement, tel quel

```
Object o;
ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("toto.obj"));
oos.writeObject(o); // Je sérialise l'objet o
oos.close();
```

Le récupérer est également simple :

```
ObjectInputStream ois = new ObjectInputStream(new FileInputStream("toto.obj"));
Object o = ois.readObject(); // Je récupère l'objet o stocké dans le fichier
ois.close();
```

Cependant, pour qu'une classe puisse être sérialisée dans un fichier, elle doit implémenter l'interface **Serializable.**

Une classe sérialisable contient alors un identifiant propre lui permettant d'être différencié des autres versions d'une classe portant le même nom :

```
private static final long serialVersionUID = 123L;
```

Si une classe est *Serializable*, elle peut être sérialisée via la méthode montrée. Cependant la sérialisation ne permet pas directement de stocker tous les attributs de la classe avec elle dans le fichier. Seuls les attributs ayant un type primitif (int, double, float, …) ou étant une classe elle-même sérialisable, seront également sérialisés avec l'objet. Les autres prendront automatiquement la valeur `null`.