Programmation Orientée Objet
M2 CCI 2008–2009
hoenen@icps.u-strasbg.fr

Université Louis Pasteur
Département d'informatique
17 décembre 2008

TP2 : faire ses classes

1 Organiser les classes

Les packages Il est parfois difficile de s'y retrouver parmi les différents *types* (classes, interfaces, etc...) mis à disposition par l'API Java et par d'autres programmeurs. Pour faciliter l'utilisation de ces différents types, on utilise des paquetages ou *packages*. Ces paquetages permettent d'éviter des conflits de nommage et d'effectuer du contrôle d'accès. Une classe appartient à un paquetage si la première ligne non-commentée du code source contient `package nom.du.package;` (les noms sont toujours en minuscules).

Par la suite, on pourra faire référence à cette classe en faisant précéder son nom par le nom du paquetage. La classe `MaClasse` du paquetage `mon.package` est référencée par `mon.package.MaClasse` dans les classes extérieurs au paquetage. De manière plus simple, si il n'y a pas de risques de conflit entre plusieurs classes de même nom, on peut utiliser directement la classe d'un paquetage dans une classe extérieure `ClasseExt` en ajoutant par exemple `import mon.package.MaClasse;` au début de `ClasseExt.java`. Les références se font alors directement par `MaClasse`. On peut également importer toutes les classes définies dans ce paquetage (`import mon.package.*;`).

Ranger les fichiers sources et le bytecode Bien que cela ne soit pas obligatoire dans les spécifications du langages Java, il est souvent recommandé d'organiser les fichiers sources des différentes classes dans une hiérarchie de répertoires qui correspondent aux paquetages auquels elles appartiennent. Ainsi le fichier `MaClasse` sera stocké dans l'arborescence `.../mon/package/MaClasse.java`.

De plus, on peut vouloir séparer les sources `.java` des fichiers `.class`, par exemple pour interdir l'accès au code source en restreignant l'accès au dossier dans lequel il est stocké. Pour cela on créé habituellement deux répertoires distincts `sources` et `classes`. Par exemple :

```
.../sources/mon/package/MaClasse.java
.../classes/mon/package/MaClasse.class
```

On peut ensuite préciser le chemin absolu jusqu'au répertoire `.../classes` dans la variable d'environnement `CLASSPATH`. Le compilateur Java et la machine virtuelle se chargent alors de rajouter le chemin correspondant à la hiérarchie des paquetages pour retrouver les fichiers `.class`.

1. Reprendre les sources du premier TP et les organiser en paquetage (par exemple `cci.tp1`). Par la suite, les classes de chaque TP seront définies dans un paquetage différent.
2. Renseigner la variable `CLASSPATH` et compiler tous les sources de manière à ce que le bytecode soit écrit directement au bon endroit.
3. Comment faire appel, depuis n'importe quel endroit dans le système, aux programmes du paquetage qui vient d'être défini ?

2 Des personnes-objets...

Contrôle d'accès Si aucun modificateur d'accès n'est ajouté avant un membre de classe, alors le membre est *package private* : il n'est accessible que pour l'ensemble des classes du paquetage courant. Si un modificateur est précisé, celui-ci peut être soit `public` (accessible par tout le monde), soit `protected` (accessible par les classes du package et les sous-classes), soit `private` (accessible uniquement par la même classe). Le tableau suivant résume l'accessibilité selon le modificateur :

Modificateur	classe	paquetage	sous-classes	le monde
`public`	O	O	O	O
`protected`	O	O	O	N
par défaut	O	O	N	N
`private`	O	N	N	N

Surcharge Java supporte la surchage des méthodes : deux méthodes peuvent avoir le même nom tant qu'elles ont une signature différente. Cette différence porte sur le nombre et/ou le type des paramètres de la méthode, mais ne doit pas porter uniquement sur le type de retour de la méthode.

On veut modéliser administrativement une personne. Répondre aux questions suivantes et essayer chaque fonctionnalité implantée dans un programme test.

1. Construire une classe `Personne` qui représente une personne par son nom, son prénom, son âge et son adresse. On portera une attention toute particulière sur le contrôle d'accès aux différents membres de la classe.
2. Ajouter les accesseurs et autres méthodes qu'impliquent ce contrôle d'accès, en sachant que seuls les noms, prénoms et âges sont obligatoires pour créer un objet de type `Personne`.
3. Modifier la classe pour permettre l'affichage de toutes ces informations par un simple :

```
Personne p = new Personne("Smithee","Alan",41);
System.out.println(p);
```

4. Écrire une nouvelle classe `Adresse` contenant le numéro, le nom de la rue, le code postal et le nom de la ville. Attention aux modificateurs, aux accesseurs et à l'affichage d'un objet de cette classe.
5. Les informations d'une adresse doivent-elles être modifiables ? Si ce n'est pas le cas, comment peut-on l'empêcher ?
6. Utiliser la classe `Adresse` dans la `Personne` en effectuant le moins de modifications possibles.

3 Immatriculation

On veut créer un système simplifié de gestion des immatriculations de véhicules. Chaque immatriculation possède un identifiant composé de caractères formant l'écriture en base 16 d'un nombre unique (en utilisant les caractères `0,1,...,9,a,...,f`), et possède également un propriétaire (une personne).

1. Écrire une classe `Immatriculation` qui contient toutes ces informations et les méthodes utiles à la gestion d'une telle immatriculation, et permet de modifier le propriétaire de l'immatriculation (par exemple lors de la vente du véhicule).
2. Faire en sorte que le constructeur de la classe ne prenne qu'une personne en argument : l'identifiant est donné automatiquement en fonction du nombre total d'immatriculations déjà attribuées.
3. Un contrôle technique est à effectué chaque année (la date importe peu). Ajouter une méthode qui dit si oui ou non il faut effectuer le contrôle technique, en sachant qu'un contrôle à été réalisé le jour de la création de l'immatriculation ou lors d'une revente. Ajouter une méthode permettant d'effectuer ce contrôle technique.

4 Un peu de géométrie

1. Écrire une classe `Point` qui représente un point à coordonnées entières dans l'espace cartésien.
2. Écrire ensuite une classe `Rectangle` permettant de représenter les objets géométriques rectangulaires. Les constructeurs devront permettre de définir un rectangle de plusieurs manières différentes.
3. Ajouter des méthodes pour calculer le périmètre, l'aire et le centre du rectangle.
4. Ajouter une méthode qui prend deux rectangles en arguments et renvoie le rectangle issu de leur union.