Développer des classes

Plan

Plan détaillé ¶ §

Nous allons maintenant nous focaliser sur l'écriture des classes et aborder:

- la visibilité (mots-clés `private`, `public`),
- le mot-clé `this`,
- le mot-clé `static`,
- le nommage (`package`, `import`).

Visibilité

Mots-clés `private`, `public`

Les classes sont plus qu'un simple assemblage de champs comme le sont les structures en C, mais sont composées de membres qui caractérisent à la fois l'état (par les champs) et le comportement (par les méthodes) d'une famille d'objets.

De plus, les mots-clés `private` et `public` permettent de régler la **visibilité**, c'est-à-dire, l'accès, par des objets tiers, aux membres des objets d'une classe.

Les membres déclarés privés d'un objet ne sont accessibles que par les objets de la même classe. Au contraire, les membres déclarés publics d'un objet sont accessibles par n'importe quel objet client.

Règles générales

On cache les champs quand

- on veut qu'un objet ne change pas au cours de l'exécution
- on veut garantir leur cohérence

Pour les méthodes

- on expose les services qu'un objet offre aux autres,
- on cache les tâches internes, qu'un objet fait pour lui-même au cours d'une opération.

Conception à la C

```java
class Point {
  public double x, y;
}
```

```java
class DemoPoint {
  public static void main(String[] args) {
    Point p = new Point();
    p.x = 2.0;
    p.y = 5.0;
    System.out.println("("+p.x+","+p.y+")");
  }
}
```

Si on considère qu'un point ne doit pas changer après sa création, exposer ses coordonnées n'est pas judicieux.

Conception typiquement objet

```java
class Point {
  private double x, y;
  public Point(double unX, double unY) {
    x = unX;
    y = unY;
  }
  public String toString() {
    return "("+x+","+y+")";
  }
}
```

```java
class DemoPoint {
  public static void main(String[] args) {
    Point p = new Point(2.0,5.0);
    System.out.println(p); //on lui demande de s'afficher
    //equivalent à: System.out.println(p.toString());
  }
}
```

Note: le mot-clé `this`

Pour adresser une requête à un objet, une référence vers cet objet doit être disponible. Dans une classe, pour adresser une requête à soi-même, on utilise **this**.

Sans ambiguité, il n'est pas nécessaire. Sinon, il distingue les champs, des paramètres ou variables locaux:

```java
public Point(double x, double y) {
   this.x = x;
   this.y = y;
}
```

Ex.1. Secteur angulaire (10 min)

Complétez le fichier **`Secteur.java`** décrivant la classe `Secteur` qui modélise un secteur angulaire, c'est-à-dire un intervalle connexe, borné par deux angles $[\theta_1, \theta_2[$, chacun dans $[0, 2\pi]$. Par convention, le secteur est celui qu'on parcourt en partant de θ_1 dans le sens trigonométrique.

Testez avec le fichier **`TestSecteur.java`**.

NB. `java.lang.Math.PI` donne la valeur de π et % est l'opérateur modulo permettant de garantir des angles dans $[0, 2\pi]$.

Membres statiques

Le mot-clé **static**

Les membres définis avec le mot-clé `static`, doivent être appelées, par une requête adressée directement à la classe (et non à une de ces instances). Cela est possible car les classes sont chargées en mémoire par la machine virtuelle (et vues comme les objets d'une meta-classe).

```
class Math {
  ...
  public static final double PI = 3.141592653589793;
  ...
  public static double cos( double a ) {
    ...
  }
}
```

```
double x = Math.cos( Math.PI / 2.0 );//on n'a pas créé d'objet!
```

Application du mot-clé

Le mot-clé **static** convient pour les

- méthodes indépendantes de tout objet, comme les fonctions sans état (`Math.cos()`).
- champs dont les valeurs sont partagés par tous les objets d'une même classe, comme les constantes (`Math.PI`). Dans ce cas on fait suivre le mot-clé **final** à `static`.

Ex.2. Secteur angulaire 2 (10 min)

Dans votre classe modélisant un secteur angulaire:

- ajoutez un champs statique constant pour la valeur 2π.
- ajoutez une méthode statique pour créer des secteurs particuliers:

- le secteur complet $[0; 2\pi[$
- le premier quadrant $[0; \pi/2[$
- Testez leur écart dans `TestSecteur`.

Composition d'objets

Relation de composition

Un objet peut en contenir d'autres via ses champs. La **composition** est la relation entre objets dans laquelle un objet est le champs du second.

Au sens fort, le champs est créé en même temps que l'objet, qui le possède.

Au sens faible (on parle dans ce cas plutôt d'**agrégation**), le champs réfère à un objet, indépendant de l'objet qui le connait.

Souvent les deux choix sont possibles et dépendent du contexte et des objectifs.

Ex.2. Cercle et arc de cercle (10 min)

- Copiez la classe `Point` précédente.
- Proposez une classe `Cercle` modélisant un cercle défini par un centre et un rayon, obligatoirement positif.
- Une classe `ArcCercle` modélisant un arc de cercle, défini par un cercle et un secteur angulaire.
- Chaque classe doit posséder une méthode `toString` qui retourne une description textuelle de l'objet courant.

NB. `java.lang.Math.abs()` retourne la valeur absolue du nombre donné.

Nommage

Package et nom complet

```
package tc.elp.java.geometry;   //nom du package
public class Point {   //nom strict de la classe
```

- Le mot-clé `package` permet d'ajouter une classe à un regroupement. Son nom est composé; chaque partie étant séparée par des points. Il doit être déclaré en haut du fichier source de la classe (un package anonyme est créé par défaut).
- Le nom **complet** de chaque classe est la concaténation du nom du package auquel elle appartient et de son nom strict.

```
tc.elp.java.geometry.Point p; //déclaration d'une variable
```

Recherche des classes

A la compilation (resp. à l'exécution), les fichiers `.java` (resp. `.class`) sont recherchés sur le système de fichiers. Le **classpath** (option `-classpath` ou `-cp`) indique le répertoire *à partir duquel* ils sont cherchés (le répertoire courant par défaut). Depuis ce répertoire, le chemin donné par le nom strict de la classe, en remplaçant les points par des *slash*, mène au repertoire où ils doivent se trouver.

Dans notre exemple, le compilateur cherchera:

classpath **/tc/elp/java/geometry/Point** `.java`

A l'exécution, la machine virtuelle cherchera:

classpath **/tc/elp/java/geometry/Point** `.class`

Alias

Ce mécanisme de nommage permet d'éviter les **conflits de noms**, mais les noms de classe sont déraisonnablement longs:

```
tc.elp.java.geometry.Point p =
  new tc.elp.java.geometry.Point(2.0,5.0);
```

Il est possible de raccourcir localement (dans un fichier source) le nom des classes en utilisant le mot-clef `import` au début des fichiers sources.

```
import tc.elp.java.geometry.Point;
//'Point', alias de 'tc.elp.java.geometry.Point' ci-dessous
...
Point p = new Point(2,5);
```

Ex.4. Package (10 min)

- Dans un répertoire appelé `ExempleNommage`, ajoutez la hiérarchie de répertoires `tc/elp/java/geometry`. Dans `geometry`, ajoutez les fichiers sources des classes `Point`, `Secteur`, `Cercle`, `ArcCercle`.
- Insérez les lignes commençant par le mot-clé `package` pour que ces classes appartiennent toutes au package `tc.elp.java.geometry`.
- Dans `ExempleNommage`, ajoutez la classe `DemoArcCercle` qui affiche le premier quadrant d'un cercle centré en l'origine et de rayon 5 (elle n'appartient pas au même package).
- Comment compiler et exécuter ?

Ex.4. suite: compilation/éxécution

- Dans `ExempleNommage`, créez un répertoire `build`, puis compilez avec `javac tc/elp/java/geometry/*.java DemoArcCercle.java -d build`. Que se passe-t-il ? La classe ArcCercle est-elle publique ? Sinon, précédez la déclaration de la classe par le mot-clé `public`.
- Exécutez avec `java -cp build/ DemoArcCercle`.

Retour sur la visibilité (O: oui, N: non)

- Visibilité des classes:

Mot-clé	Package	Tous
public	O	O
aucun	O	N

- Visibilité des membres:

Mot-clé	Classe	Package	Tous
public	O	O	O
aucun	O	O	N
private	O	N	N

Ce qu'il faut retenir

Le sens et l'intérêt des mots-clés (ou de l'absence des mot-clés):

- `this`,
- `private`, `public`,
- `static`,
- `package`, `import`.