LABO 7 – REVISION SUR LES CLASSES
Informatique 1

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

But du laboratoire (2 périodes)

1. Dans ce laboratoire, vous allez réviser les notions de base des classes vues au cours ainsi qu'implémenter différents mécanismes liés aux classes. Ce labo possède deux parties distinctes.
   Dans la première, il s'agit de réaliser sur ordinateur la classe *Rectangle* vue au cours. Dans un deuxième temps, vous allez réaliser une petite application pour gérer un garage contenant différentes voitures.
2. La durée estimée pour réaliser ce laboratoire est de **deux périodes (1 heure 30 minutes)**.
3. Vous pouvez trouver cette donnée sous forme électronique se trouve sur le site web du cours (http://inf1.begincoding.net). Vous y trouverez également le corrigé de ce labo la semaine prochaine.

Partie 1 – Implémentation de la classe *Rectangle* / 45 minutes

Ce premier exercice est destiné aux personnes n'ayant jamais travaillé avec des classes et/ou pour qui les classes restent encore un sujet opaque. Nous allons ainsi voir pas-à-pas comment implémenter une classe simple comme vu au cours afin de pouvoir s'habituer progressivement à ces nouveaux concepts.

Tâche 1

1) Après avoir lancé *Eclipse*, créez une nouvelle classe *Rectangle* dans un nouveau fichier. Utilisez la définition suivante : un rectangle possède deux attributs de type double, *width* et *height*.
2) Créez ensuite une nouvelle classe, nommée *Task1Runner,* qui contiendra la méthode *main*. Dans ce *main*, déclarez les objets suivants de la classe *Rectangle* :
   a) r1, taille 10x5.
   b) r2, taille 3x4.
   c) r3, taille 100x100.
3) Remplacez ensuite le constructeur par défaut de la classe *Rectangle* par votre propre constructeur définissant directement les attributs *width* et *height*.
4) Modifiez la déclaration de l'objet r1 afin qu'il prenne en compte ce nouveau constructeur.
5) Ajoutez une méthode permettant de calculer la surface du rectangle et utilisez-là sur r1 afin de déterminer sa surface. Affichez le résultat sur la console afin de vérifier le fonctionnement correct du programme.

Tâche 2

1) Cachez maintenant les attributs de la classe rectangle afin qu'on ne puisse plus les modifier depuis l'extérieur de la classe. Comment peut-on désormais y accéder ? Quel est l'avantage de faire cela ? Ces questions seront posées en classe. Préparez votre réponse et ajoutez les méthodes nécessaires à la classe *Rectangle*.
2) De manière générale, tous les objets en *Java* sont censés implémenter une méthode `public String toString()` qui retourne la représentation textuelle de l'objet sous forme de *String*. Cette méthode est utilisée par exemple lorsque l'on essaie d'utiliser l'objet en tant que *String*, par exemple dans le cas de `System.out.println`. Ainsi, chaque classe peut donc proposer une version « texte » de ce qu'elle est. Implémentez cette méthode `toString()` pour la classe *Rectangle* de manière à ce que l'instruction suivante :

   ```
   System.out.println(r1);
   ```

   affiche le texte suivant sur la console :

   ```
   Rectangle size : 10.0 x 5.0
   ```

Tâche 3

1) Ajoutez à la classe un attribut `color` de la classe `Color` qui est fournie par le langage Java (ajoutez `import java.awt.Color;` tout au début de votre fichier pour pouvoir l'utiliser). Modifiez le constructeur de manière adéquate pour que tous les rectangles soient rouges par défaut.
2) Ajoutez une méthode pour pouvoir changer la couleur d'un rectangle.
3) Ajoutez à la représentation textuelle la couleur du rectangle.

Partie 2 : Application garage / 45 minutes

Dans cette partie, vous allez réaliser une petite application de gestion de garage. Pour ce faire, on vous demande tout d'abord d'implémenter une classe *Auto*. Cette classe comporte les attributs suivants :

- Nom (*String*).
- Nombre de chevaux (*double*).
- Vitesse maximale (*double*).

Tâche 4

1) Implémentez cette classe en redéfinissant le constructeur par défaut de manière à ce que tous les paramètres soient définis dans le constructeur.
2) Créez une nouvelle classe nommée *Task2Runner* qui contiendra la méthode *main* pour cette deuxième partie. Dans cette méthode, créez les instances de voiture suivantes :
   - Une Audi TT, 135 cv, 250 km/h max
   - Une VW Golf, 100 cv, 180 km/h max
   - Une Lancia Y, 60 cv, 150 km/h max
   - Une Porsche 911, 400 cv, 312 km/h max
3) Ajoutez une méthode `toString()` pour que l'on obtienne pour la voiture 1, « `Audi TT, 135.0 hp, 250.0 km/h` »
4) Ajoutez encore une méthode `isVeryFast()` retournant `true` si la voiture peut aller à plus de 200 km/h et `false` autrement. Testez votre méthode sur la Porsche et sur la Lancia en affichant le résultat à la console.

Tâche 5

Vous allez maintenant ajouter une classe *Garage* qui modélise le garage qui stocke différentes autos. Pour notre programme, un garage a une capacité de 100 autos maximum qui seront stockées dans un tableau. Notez que dans ce garage, il est possible d'ajouter des objets de la classe voiture grâce à la méthode `addCar(Auto a)`. Toutefois, il n'est pas possible d'en enlever !

Il est également possible à tout moment de connaître le nombre de voitures dans le garage grâce à la méthode `getNumberOfCars()`.

La représentation UML de cette classe est la suivante :

Garage
`Auto stock[100]`

`void addCar(Auto a)`
`int getNumberOfCars()`
`String toString()`

1) Implémentez la classe *Garage.* La méthode `toString()` retourne un *String* contenant la représentation textuelle de toutes les voitures présentes dans le garage comme suit :

```
Garage content :
      Car 0 : Audi TT, 135.0 hp, 250.0 km/h
      Car 1 : VW Golf, 100.0 hp, 180.0 km/h
      …
```

   Dans cette méthode, vous devez utiliser la méthode `toString()` des objets *Auto* afficher la partie sombre.
2) Ajoutez les quatre voitures créées ci-dessus dans le garage en vous assurant que le nombre d'auto est correct à la fin de votre programme en l'affichant sur la console.

Tâche 6 / Exercice optionnel

Ajoutez au garage la méthode nommée `displayFastCars()` qui affiche sur la console toutes les voitures rapides (càd qui retournent `true` à la méthode `isVeryFast()`). Sur notre garage, on doit obtenir :

```
Garage fast cars :
       Car 0 : Audi TT, 135.0 hp, 250.0 km/h
       Car 3 : Porsche 911, 420.0 hp, 312.0 km/h
```

Testez la méthode `displayFastCars()` qui dans ce cas doit afficher l'Audi et la Porsche.

Tâche 7 / Exercice optionnel

Le garagiste qui utilise votre application se rend compte que les clients demandent souvent des informations sur la consommation des voitures. Il voudrait ainsi pouvoir ajouter des informations sur celle-ci.

Etant donné que les consommations varient en fonction des conditions dans lesquelles la voiture est utilisée, nous vous proposons de modéliser une consommation grâce à une classe. Créez la classe *Consommation* contenant les trois informations suivantes, stockées sous forme de *double* :

- Consommation / 100 km en régime urbain (`urban`)
- Consommation / 100 km en régime autoroutier (`highway`)
- Consommation / 100 km en régime mixte (`combined`).

1) Ajoutez un attribut *Consommation* à la classe *Auto*.
2) Ajoutez ensuite dans la classe *Garage* une méthode qui affiche le nom de toutes les voitures dont la consommation mixte est inférieure à 8 litres / 100 km et également la consommation urbaine est inférieure à 10 litres / 100 km.