IUT Belfort Montbéliard
Département Informatique

Programmation Orientée Objet - DUT S2
TP - Premiers objets simples

L'objectif de ce TP est de vous apprendre à rédiger une classe, en y positionnant les attributs et les méthodes. Vous créerez ensuite un *objet* à partir de cette classe en l'instanciant, et vous manipulerez l'objet en affectant ses attributs et en invoquant ses méthodes. Un autre objectif est de découvrir la méthode `toString()`.

1 Coder une classe `Personne`

On considère une classe décrivant une personne, conformément au modèle suivant :

`Personne`

`nom : chaine`
`prenom : chaine`
`jourNaiss : entier`
`moisNaiss : entier`
`anneeNaiss : entier`

`setPrenom(n_prenom : chaine) : rien`
`setNom(n_nom : chaine) : rien`
`setDateNaiss(j : entier, m : entier, a : entier) : rien`
`ageEn2014() : entier`

Les méthodes `setNom()` et `setPrenom()` permettent respectivement de fixer les attributs `nom` et `prenom` à des valeurs fournies en paramètre.

La méthode `setDateNaiss()` permet de fixer les attributs `jourNaiss`, `moisNaiss` et `anneeNaiss` à des valeurs fournies en paramètre.

La méthode `ageEn2014()` calcule et renvoie l'âge de la personne par rapport à l'année 2014.

Question 1. Codez cette classe en Java (attention : n'ajoutez pas de méthode `main()` à cette classe).

Question 2. Compilez cette classe. Remarque : comme la classe `Personne` ne possède pas de méthode `main()`, vous ne pouvez pas *exécuter* la classe pour l'instant.

1.1 Instancier un objet de la classe `Personne`

Puisque la classe `Personne` ne contient pas de méthode `main()`, elle ne constitue pas un programme Java exécutable, mais elle est utile uniquement à décrire des objets du type `Personne`.

Vous allez maintenant écrire parallèlement à la classe `Personne` une seconde classe, qui elle contiendra la méthode `main()`, et constituera donc le programme à proprement parler. Ce programme a pour but de créer (instancier) un objet du type `Personne` et de le manipuler.

En Java, on utilise l'opérateur `new` pour instancier un objet après l'avoir déclaré. L'instruction suivante permet de déclarer *et* d'instancier un objet de la classe `Personne` :

```
Personne p1 = new Personne();
```

La partie `Personne p1` déclare un nouvel objet nommé `p1` et de type objet `Personne`, tandis que `p1 = new Personne()` permet d'instancier cet objet, c'est à dire de le créer physiquement en mémoire.

N.B. En répondant aux questions, n'oubliez de compiler régulièrement votre code pour le corriger au fur et à mesure. Vous devez pour compiler indiquer toutes les classes auxquelles vous faites appel.

Question 3. Écrivez une classe nommée `GoPersonne` avec uniquement la méthode `main()`.

Question 4. Déclarez un objet `p1` de type `Personne` dans cette méthode `main()` et instanciez le.

Question 5. Demandez à l'utilisateur de saisir un nom et un prénom, et attribuez ce nom et ce prénom à `p1` en invoquant sur lui la méthode appropriée.

Question 6. Même chose pour le jour, le mois et l'année de naissance.

Question 7. Réalisez avec des `System.out.println()` les affichages successifs de tous les attributs de `p1`.

Question 8. Déclarez et instanciez un nouvel objet `p2` de type `Personne`, puis attribuez lui un nom et un prénom fournis par l'utilisateur.

Question 9. Recopiez dans `p2` les mêmes valeurs de date de naissance que `p1` (pas de saisie utilisateur).

N.B. Nous verrons plus tard qu'il est fortement déconseillé de faire ce qu'il vous est demandé de faire ici : affecter directement les attributs d'un objet sans passer par une méthode. Nous le faisons pour l'instant pour que vous vous familiarisiez avec la syntaxe objet.

Question 10. Comme pour `p1`, réalisez avec des `System.out.println()` les affichages successifs de tous les attributs de `p2`.

1.2 La méthode `toString()`

Il est pénible d'avoir à afficher un à un tous les attributs d'un objet pour connaître son état. Il serait plus commode de pouvoir écrire directement `System.out.println(p1);`
pour pouvoir afficher en une seule instruction l'état de l'objet `p1`.

Cela est possible grâce à une méthode particulière en Java nommée `toString()`. Sa signature est obligatoirement `public String toString()`
et elle doit être ajoutée à la classe dont on souhaite visualiser l'état (ici la classe `Personne`).

Le code de la méthode `toString()` doit construire une chaîne de caractères représentant l'état de l'objet, puis la renvoyer.

ATTENTION. La méthode ne réalise pas elle même l'affichage par `System.out.println()`, mais se contente de *construire* la chaîne de caractère qui sera affichée plus tard par un `System.out.println()`.

Par exemple, la méthode `toSTring()` qui permet de représenter une personne de prénom `Jason` et de nom `Lytle` sous la forme `[Lytle, Jason]` est la suivante :

```
public String toString() {
      return "[" + nom + ", " + prenom + "]";
}
```

Question 11. Dotez la classe `Personne` d'une méthode `toString()` permettant de représenter une personne par une chaîne de la forme `nom, prenom (jj/mm/aaaa)`, où `nom` est remplacé par le nom de la personne, `prenom` par son prénom, `jj` par son jour de naissance, `mm` par son mois de naissance et `aaaa` par son année de naissance. Par exemple, la chaîne représentant une personne de prénom `Jason`, de nom `Lytle`, née le 26 mars 1969, sera `Lytle, Jason (26/03/1969)`.

Question 12. Vérifiez l'effet de cette méthode dans le `main()` en demandant l'affichage de `p1` et `p2` par

```
System.out.println(p1);
System.out.println(p2);
```

2 Un objet « Reveil matin »

Le réveil sera modélisé de la manière suivante.

2.1 État du réveil

L'état du reveil est caractérisé par l'horaire qu'il possède (heure, minute, seconde), par un horaire d'alarme (heure, minute, seconde), et par le fait que l'alarme est enclenchée ou non.

On définira donc 6 attributs entiers (heure courante, minute courante, seconde courante, heure alarme, minute alarme, seconde alarme) et un attribut booléen (l'alarme) pour décrire l'état du reveil.

2.2 Comportement du réveil

Le comportement du réveil est le suivant
- il peut être ajusté à un horaire courant fourni par la donnée de 3 éléments (heure, minute, seconde) ;
- son alarme peut être réglée à un horaire fourni par la donnée de 3 éléments (heure, minute, seconde) ;
- son alarme peut être allumée, et elle peut également être éteinte ;
- on doit pouvoir savoir si l'horaire courant est égal à celui de l'alarme ou pas ; si c'est le cas, un signal doit être émis (sous la forme d'un message affiché sur la console) ;
- l'horaire courant doit pouvoir être comparé à celui d'un autre objet de type réveil ;
- la différence (en nombre de secondes) entre l'horaire courant et celui d'un autre réveil doit pouvoir être calculée.

L'objectif du TP n'est pas de réaliser un réveil qui fonctionne en temps réel, c'est à dire qui décompte effectivement le temps qui passe. Ici, le passage du temps est modélisé de manière abstraite par le fait que le réveil peut être incrémenté, c'est à dire que son horaire courant est augmenté d'une seconde. L'horaire résultant de cet ajout d'une seconde doit bien entendu être cohérent : l'heure courante est dans l'intervalle [0..23], et la minute et la seconde courantes dans l'intervalle [0..59].

De même l'ajustement à un nouvel horaire courant ou à un nouvel horaire d'alarme ne doit être possible que si cet horaire est cohérent. Dans le cas contraire, les horaires ne sont pas modifiés.

Ce comportement sera décrit au moyen de méthodes dont les noms seront respectivement `setHoraireCourant()`, `setHoraireAlarme()`, `allumerAlarme()`, `eteindreAlarme()`, `controlerAlarme()`, `comparerA()`, `differenceAvec()` et `incrementer()`.

2.3 Réalisation

Écrire la classe Reveil correspondant à cette description. Cette classe sera écrite dans un fichier nommé `Reveil.java`.

Dotez cette classe d'une méthode `toString()` appropriée.

3 Utilisation de l'objet Reveil

Créer une nouvelle classe, qui s'appelle `RunReveil`, et dont le seul but est d'utiliser des objets `Reveil` par le biais de leur méthodes. Cette classe sera écrite dans un autre fichier que `Reveil.java`. Elle doit être écrite dans un fichier intitulé `RunReveil.java`.

Cette classe contient la méthode `main()`.

Dans la méthode `main()`, déclarer et instancier 2 objets $r1$ et $r2$ de type `Reveil`.

Afficher l'état de l'un d'entre eux pour voir la valeur initiale des attributs.

Tester le bon comportement de la méthode `incrementer()` avec (par exemple) les horaires courants suivants avant l'appel à `incrementer()` : $(0, 0, 0)$, $(23, 59, 59)$, $(21, 59, 59)$, $(20, 18, 59)$, $(20, 15, 45)$, ... Vous devez bien sûr pour cela afficher l'heure courante avant et après l'incrémentation.

Ajuster à la même heure les réveils $r1$ et $r2$, puis vérifier le bon comportement des méthodes `comparerA()` et `differenceAvec()`. Même chose avec des valeurs différentes pour $r1$ et $r2$.

Dans le même ordre d'idées, faire fonctionner l'alarme d'un réveil suite à des appels répétés à la méthode `incrementer()`. Par exemple, fixer l'horaire courant du réveil $r1$ à $(6, 28, 30)$ (c'est à dire à 6h28min30s), et son horaire d'alarme à $(6, 30, 0)$ (6h30). Puis, incrémenter par une boucle 150 fois le réveil $r1$, en vérifiant à chaque pas d'itération s'il est l'heure de déclencher l'alarme (par le biais de la méthode `controlerAlarme()`). Ne pas oublier qu'en principe l'alarme ne doit « sonner » que si elle a été allumée.