Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

DEV2 – JAVL – Laboratoire Java

TD 2 – L'orienté objet

Partie 1

Résumé

Ce TD est le premier consacré à l'orienté objet. Vous allez coder une classe implémentant le concept de *moment dans la journée*.

Mise en place

- ▷ Vous allez utiliser le dépôt créé lors du 1er TD.
- ▷ Vous allez coder dans le **même projet**, dans un package dédié (`g12345.dev2.td02`).
- ▷ Vérifiez que la version du dépôt sur le PC est synchronisée avec le dépôt sur GITLAB [a] : faites un *clone* s'il n'est pas encore présent sur le PC ou un *pull* si votre PC contient une vieille version du dépôt.

a. Dans le cas contraire, il vous sera difficile d'y sauver votre travail plus tard.

1 La classe Moment

Nous vous demandons de coder, pas à pas, une classe introduisant la notion de *moment dans la journée* (représenté à la seconde près). Veillez à bien tester régulièrement ce que vous écrivez.

1.1 Attributs et constructeur

1. Créez une classe `Moment` avec les trois attributs (privés) suivants : `hour`, `minute` et `second`, tous des entiers.
2. Ajoutez à cette classe un constructeur qui reçoit les valeurs des attributs. Nous ne vous demandons pas encore de vérifier la validité des paramètres.
3. Créez une classe `MainMoment` avec une méthode `main` qui crée trois moments : `moment1`, `moment2` et `moment3`.
4. Ajoutez l'instruction `System.out.println(moment1);`. Qu'affiche-t-elle ?

Faites un *commit* (message : `"td02/ex1.1: attributs et constructeur"`) avant de passer à l'étape suivante.

Version du document
11 février 2021

1.2 Accesseurs

1. Revenez à la classe `Moment` et ajoutez les accesseurs pour chacun des attributs.
2. Modifiez votre méthode `main` pour afficher les valeurs des attributs des 3 moments.

> Dernier rappel : Faites un **commit** après chaque étape.

1.3 Documentation

Vous avez peut-être codé la classe sans écrire de documentation. Ce n'est pas bien ! Il **faut** écrire la documentation en même temps que le code !

Prenez le temps d'écrire une JAVADOC complète du code déjà existant :

1. On vous demande de l'écrire en anglais.
2. N'oubliez pas la documentation de la classe même.
   - ▷ À placer juste avant la classe.
   - ▷ C'est la première chose que va lire un développeur souhaitant comprendre/utiliser votre classe. Elle doit expliquer ce que représente une instance de cette classe (ici un moment de la journée) en donnant un maximum de détails utiles.
   - ▷ Il faut par contre (comme pour toute javadoc d'ailleurs) passer sous silence tout ce qui est privé : quelqu'un qui lit la javadoc doit la comprendre sans lire le code correspondant !
3. Les documentations des accesseurs peuvent parfois sembler inutiles et redondantes, c'est vrai [1].

> Vous pouvez vous inspirer d'une javadoc officielle. Consultez par exemple la documentation de la classe `LocalTime` qui est la classe standard la plus proche de ce que vous êtes en train de construire.

1.4 Affichage

1. Ajoutez à la classe `Moment` la méthode `toString()` qui retourne la chaine représentant le moment en suivant ce modèle : `13:01:20`
2. N'oubliez pas de documenter cette méthode.
3. Qu'affiche maintenant `System.out.println(moment1);` ?

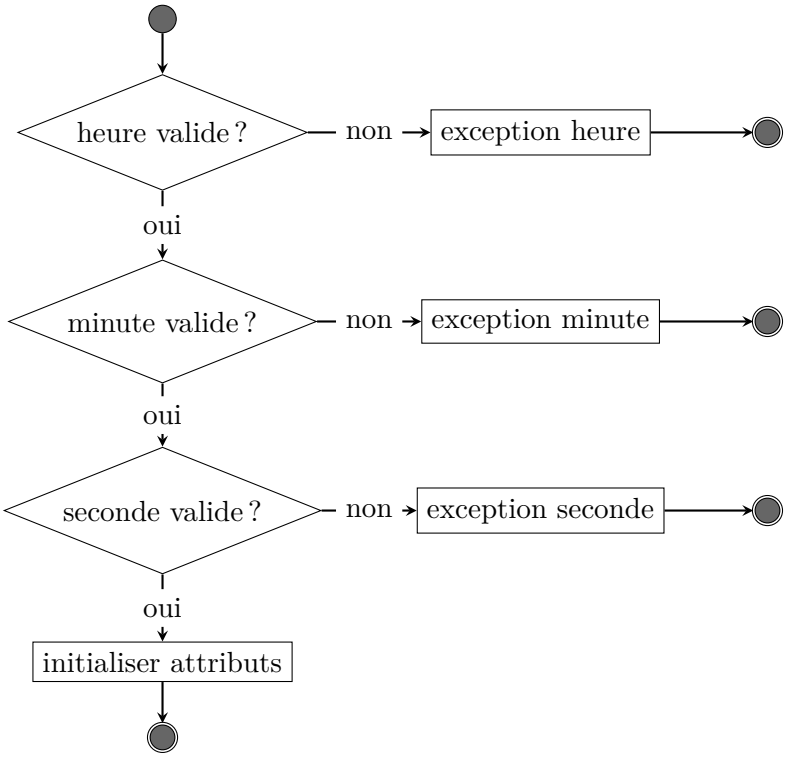
1.5 Validation

Pour l'instant, il n'y a aucun test dans le constructeur. Il est, dès lors, possible de créer des moments qui n'ont pas de sens (par exemple : `32:-04:87`). Remédions-y !

1. Ajoutez dans le constructeur du code pour valider les paramètres fournis. La valeur de l'heure doit être comprise entre 0 et 23 inclus, la valeur des minutes et des secondes entre 0 et 59 inclus. Si un des paramètres fournis n'est pas valide, vous lancerez une `IllegalArgumentException`.

   Voici un diagramme d'activité [2] qui illustre les étapes de l'algorithme.

1. C'est un sujet de discussion récurrent dans la communauté JAVA.
2. Vous apprendrez à lire/écrire des diagrammes d'activités dans vos cours d'analyse. C'est très proche de l'ordinogramme dont on vous a parlé en DEV1.



2. Préparez un plan de tests et montrez-le à votre professeur pour validation.
3. Écrivez des tests unitaires JUnit pour tester la validité de votre constructeur. Pensez à tester des cas corrects, les cas limites corrects et incorrects (qui provoquent des lancements d'exception).

1.6 Égalité

Que se passe-t'il si on teste l'égalité de deux moments ? Faite l'expérience en utilisant le code suivant dans votre méthode principale.

```
Moment moment1 = new Moment(8,15,0);
Moment moment2 = new Moment(8,15,0);
Moment moment3 = moment1;
Moment moment4 = new Moment(10,15,0);
stem.out.println(moment1 + "==" + moment2 + ": " + (moment1==moment2));
stem.out.println(moment1 + "==" + moment3 + ": " + (moment1==moment3));
stem.out.println(moment1 + "==" + moment4 + ": " + (moment1==moment4));
System.out.println(moment1 + "equals" + moment2 + ": " + moment1.equals(moment2));
System.out.println(moment1 + "equals" + moment3 + ": " + moment1.equals(moment3));
System.out.println(moment1 + "equals" + moment4 + ": " + moment1.equals(moment4));
```

Quel est le résultat ? Comprenez-vous pourquoi ? Quel résultat aurions-nous aimé avoir à la place ?

Schéma mémoire
Faites un schéma mémoire des variables `moment1` à `4` pour justifier les résultats obtenus. Apprendre à faire de tels schémas mémoire vous sera utile pour comprendre d'autres comportements complexes. Ce sera aussi important pour préparer l'oral.

Écrivons une méthode `equals()` afin que `moment1` soit *equals* à `moment2`.

1. Ajoutez à votre classe une méthode[3] `public boolean equals(Moment other)` permettant de vérifier si deux moments sont égaux ou non. Deux moments sont égaux s'ils ont la même valeur pour l'heure, les minutes et les secondes.
2. Relancez votre méthode `main` afin de vérifier que le résultat est bien celui attendu cette fois.

1.7 Surcharge de constructeur

1. Revenez à la classe `Moment` et ajoutez-y un constructeur sans paramètre qui initialise le moment à `00:00:00` (Revoyez les explications du mot clé `this` dans le slide 38 du cours si nécessaire).
2. Modifiez la méthode principale pour créer et afficher un moment représentant minuit.

1.8 Comparaison

1. Ajoutez à la classe `Moment` une méthode sans paramètre `int toSeconds()` qui convertit le moment en secondes.
2. Dans cette même classe, ajoutez une méthode `int compareTo(Moment other)` qui retourne :
   - ▷ 0 si `other` est égal au moment courant ;
   - ▷ un entier négatif si le moment courant est antérieur à l'`other` ;
   - ▷ un entier positif si le moment courant est postérieur à l'`other`.

   Une solution simple utilise `toSeconds`.
3. Modifiez la méthode principale afin d'afficher si le premier moment vient avant le second ou non.
4. Planifiez puis écrivez les tests unitaires pour vous assurer de la validité de votre méthode `compareTo`.

Le développement de la classe `Moment` est à présent terminé. N'oubliez pas de faire un dernier commit et de déposer votre travail sur le serveur GitLab.

2 Préparation à l'oral

Dans le cadre de DEV_2, vous aurez un examen oral sur la théorie. Lors de cet examen, vous allez tirer deux mots au hasard (parmi une liste de mots connue et publiée bien avant l'oral). Voici ceux qui sont en lien avec ce TD : `Encapsulation`, `toString`, `equals` (une partie seulement).

Pour chacun de ces mots, réfléchissez à comment vous les présenteriez lors d'un oral[4] : explication du concept, exemple pour l'illustrer.

3. La signature recommandée est `public boolean equals(Object other)`. Nous y reviendrons mais simplifions pour l'instant.
4. Vous pouvez vous créer des fiches qu'il vous suffira de relire/mémoriser en session.