Série1 : Morane Gruenpeter

Bref résumé de la série 1 :

Première série du cours pendant laquelle on travaille sur l'objet DateUser et sur la classe client qui sera la classe de l'utilisateur pour tester les objets et les méthodes. En plus, première initiation avec la classe **Calendar** de la bibliothèque Java. Ensuite cette série est un début de projet dans lequel l'utilisation des dates sera fréquente et essentielle.

Class DateUser

La classe DateUser représente un objet date, qui désigne un jour spécifique en contenant trois entiers le jour, le mois et l'année de la date.

Field Detail

lesJours

`static int[] lesJours`

lesJours est un tableau d'entiers qui nous donne la valeur par défaut de nombre des jours par mois. Sachant que la case 0 est pour Janvier et la case 11 est pour Décembre.

Constructor Detail

DateUser

`public DateUser()`

Constructeur n°1- constructeur par défaut. Initialise la date à la date d'aujourd'hui.
exo1.2- le constructeur par défaut qui crée l'objet Calendar et initialise la date par défaut étant la date du jour

DateUser

`public DateUser(int j, int m, int a)`

Constructeur n°2- constructeur avec paramètres.
`int` - j pour le jour
`int` - m pour le mois
`int` - a pour l'année

Method Detail

getAnnee

`public int getAnnee()`

Retourne la valeur de l'année de l'objet courant **Returns:** int annee

getJour

`public int` **`getJour`**`()`

Retourne la valeur du jour de l'objet courant **Returns:** int jour

getMois

`public int` **`getMois`**`()`

Retourne la valeur du mois de l'objet courant **Returns:** int mois

setAnnee

`public void` **`setAnnee`**`(int a)`

Setter de la valeur année de l'objet courant avec l'entier en paramètre

Parameters: `int` - a- nouvelle valeur de l'année

setJour

`public void` **`setJour`**`(int j)`

Setter de la valeur jour de l'objet courant avec l'entier en paramètre

Parameters: `int` - j- nouvelle valeur de jour

setMois

`public void` **`setMois`**`(int m)`

Setter de la valeur mois de l'objet courant avec l'entier en paramètre

Parameters: `int` - m- nouvelle valeur de mois

toString

`public java.lang.String` **`toString`**`()`

Retourne une chaine de caractères de l'objet date courant

Overrides: `toString` in class `java.lang.Object` **Returns:** String

anneeBissextile

`private static boolean` **`anneeBissextile`**`()`

Retourne vrai si l'année est bissextile et faux sinon. **Returns :** boolean si bissextile

validerDate

`public static boolean` **`validerDate`**`(int j, int m, int a)`

Retourne vrai si les entiers en paramètres sont acceptable comme date en retrouvant tous les cas possible des mois de l'année

Paramètres:

`j-` - pour jour

`m-` - pour mois

`a-` - pour annee

Returns:

booléen si la date est correcte

hier

`public` `DateUser` **`hier`**`()`

Retourne une nouvelle date d'hier de l'objet courant en un nouvel objet date

Après avoir créé une nouvel instance DateUser avec les paramètres de l'objet courant j'applique la méthode hierChange() sur cette nouvelle instance.
Returns: DateUser hier

hierChange

`public void **hierChange**()`

Change l'objet date courant avec la valeur de la date d'hier après avoir vérifié si ce n'est pas le premier du mois, si oui, elle vérifie quel est le mois de la dates courante pour en utilisant le tableau static lesJours et la méthode anneeBissextile() pour le mois de Février on peut définir quel est la date qui la précède.

lendemain

`public DateUser **lendemain**()`

Retourne une nouvelle date du lendemain de l'objet courant en un nouvel objet date
Returns:
DateUser lendemain

lendemainChange

`public void **lendemainChange**()`

Change l'objet date courant avec la valeur de la date du lendemain après avoir vérifié si ce n'est le dernier du mois, en utilisant le tableau static lesJours et la méthode anneeBissextile() pour le mois de Février; on peut définir quel est la date qui la succède.

age

`public int **age**()`

Retourne l'entier qui désigne l'âge par rapport à la date (de naissance) pointée en créant un objet Calendar.getInstance() qui me retourne la date d'aujourd'hui et en faisant une soustraction de l'année de l'objet courant et l'année d'aujourd'hui. Prenant en considération le mois et le jour de la naissance avec des tests de 'if'.
Returns:
int age

avant

`public boolean **avant**(DateUser d)`

Retourne vrai si l'objet courent est avant l'objet en paramètre et faux sinon.
En comparant premièrement l'année, après le mois et après le jour si nécessaire.
Parameters:
`DateUser` - d , la date avec laquelle on compare
Returns: boolean estAvant

Class ClientJavaDateUser

Le scénario du ClienJava commence par une sortie de la date du jour. Ensuite, une demande de saisie, « le jour ? », « le mois ? » et « l'année ? ». La méthode saisieEntier contrôle les intervalles dans lesquels l'utilisateur choisi son entier. Si l'entier est hors limite, la boucle dans saisieEntier lui demande de nouveau d'entrer un entier. Après, cette date est testé ; si elle est une date correcte (qui existe), sur cette date s'effectue un jeu de tests des méthode : hier(),

hierChange(), lendemain(), lendemainChange(). A chaque test une sortie du résultat du test. Si la date est incorrecte, s'affiche une seule sortie qui dit que cette date est incorrecte. Fin du scénario ClientJavaDateUser.

```
Test 1: la création de la date d
Test 2: la méthode hier() en créant un nouvel objet d2
Test 3: que date d, est toujours la même
Test 4: méthode hierChange() qui change la date d avec la date d'hier
Test 5: la méthode lendemainChange qui change l'objet d avec la date du
lendemain
Test 6: création de nouvel objet date qui aura la date du lendemain de
dernier d
Test 7: que date d, est toujours la même
```

Class ClientJavaDateUser2

Le scénario du ClienJava2 commence de la même manière que le premier scénario, sauf que parmi les tests du jeu de tests sont sur les méthode age() et avant().

```
Test 1: la création de la date d
Test 2: de la méthode age
Test 3: de la méthode avant en créant en main une autre date qui sera
toujours celle comparable
Test 3: de la méthode avant, qui vérifie si la date entrée est avant la
date d'aujourd'hui
```

Mon apport par rapport au cours

Pendant le travail de cette série j'ai utilisé deux majeures techniques vues en cours. La première étant le **switch** et la deuxième est l'utilisation du **scanner** pour la saisie d'un entier. En plus l'écriture la plus concise et suffisante sans trop de ligne de code.

La classe **Calendar** et son utilisation était un défi intéressant car je ne la connaissais pas et en regardant sa documentation j'ai vu qu'elle avait des méthodes de comparaison qui pouvait peut être m'être utile pour la méthode avant(). Or je n'ai pas réussi à utiliser la méthode before() car la classe Calendar est abstrait, donc j'ai écrit les classes age() et avant() avec mes propre technique n'utilisant que Calendar.getInstance() de l'énoncé.

Conclusion

Avant tout j'ai trouvé la classe DateUser très intuitive pour la première série et cela a facilité le travail sur cette classe. La méthode de saisieEntier() dans la classe Client était moins intuitive mais me semble après le travail beaucoup plus efficace.

Ensuite, les tests des classes dans la classe Client me paraient pour le moment long et pas optimiser par rapport à une création de fichier JUnit pour le contrôle des dates et des différentes méthodes. Pour l'instant la classe Client ne ressemble pas à une classe utilisateur, mais j'ai vu que pour la série2 on travaillera sur la notion de scénario de la classe Client. J'ai

rencontré des problèmes plutôt en effectuant des tests dans la classe Client, que j'ai corrigé au fur et à mesure, or en le faisant manuellement cela m'a pris trop de temps en espérant de n'avoir oublié aucun cas.

De plus, les nouvelles petites technique que j'ai apprises m'ont aidée pour l'optimisation de l'écriture du programme et ceci est très important.

Finalement je prends avec grande volonté tous ce que j'ai appris pendant le travail. Pour les prochaines série je préfèrerais d'effectuer aussi des classes de JUnit pour tester mon travail de manière régulière et contrôlée.