Les tests unitaires – TP3
Ecriture d'une classe de tests unitaires

Objectif

L'objectif du TP est d'écrire une classe de tests unitaires afin de tester les méthodes de la classe ***OutilHoraire***. Cette classe contient différents traitements qui s'appliquent à des horaires de la journée, exprimés en heures et minutes. Pour que l'horaire soit valide, le nombre d'heures doit être compris entre 0 et 23 et le nombre de minutes entre 0 et 59.

Si un tel horaire est exprimé en tant que chaîne de caractères, celle-ci aura obligatoirement le format suivant : cchcc, soit 2 chiffres, puis la lettre 'h', puis 2 chiffres.

Vous serez amenés à écrire des tests unitaires pour les méthodes suivantes :

- ***estChiffre***
- ***estHoraireValide***
- transformer en chaîne de caractères un nombre entier supposé être la conversion d'un horaire en minutes
- effectuer l'opération inverse de la précédente.

Fichiers à récupérer sur Moodle :

- la classe ***OutilHoraire*** qui contient notamment les 4 méthodes que vous devrez tester
- les classes ***OutilHoraireEr1*** et ***OutilHoraireEr2*** qui contient les 4 méthodes, mais avec des erreurs de programmation
- un fichier ***OutilHoraireTestACompleter*** qui constitue le début de la classe de tests unitaires que vous devrez programmer dans ce TP

PREALABLE

Récupérer les différents fichiers et les adapter de manière à ce que la fonction ***main*** de la classe ***OutilHoraireTest*** s'exécute.

ETAPE 1 – Test unitaire de la méthode *estChiffre*

La méthode ***estChiffre*** prend en paramètre un caractère. Elle renvoie vrai ou faux selon que ce caractère est un chiffre (donc un caractère compris entre '0' et '9' ou pas). L'objectif est d'écrire une méthode pour tester le bon fonctionnement de la méthode ***estChiffre***.

Globalement, il faudra définir un jeu de tests sous la forme de 2 tableaux contenant des caractères. Le premier contiendra des caractères pour lesquels la méthode est supposée renvoyer vrai comme résultat, et le deuxième des valeurs pour lesquelles la méthode doit renvoyer faux.

Comme, le nombre de valeurs pour lesquelles la méthode renvoie vrai est petit (10 valeurs, pour les caractères compris entre '0' à '9'), et que de plus le test sera automatisé, on peut se permettre d'effectuer un test pour chacune de ces valeurs.

Il n'est en pas de même pour les caractères qui ne sont pas des chiffres et pour lesquels la méthode doit renvoyer la valeur faux. On testera seulement quelques valeurs.

Si la méthode ***estChiffre*** est bien programmée, on souhaite que la méthode de test affiche le résultat suivant :

```
-----------------------------------------------
     TESTS DE LA CLASSE  OUTIL HORAIRE
-----------------------------------------------


Test de la méthode estChiffre
------------------------------
   ==> a) Tests avec des chiffres :
10 test(s) ont réussi sur un total de 10 tests réalisés.
   ==>  Tous les tests sont OK


   ==> b) Tests autres caractères :
11 test(s) ont réussi sur un total de 11 tests réalisés.
   ==>  Tous les tests sont OK
```

Supposons qu'il y ait des erreurs de programmation, l'affichage souhaité est le suivant :

```
Test de la méthode estChiffre
------------------------------
   ==> a) Tests avec des chiffres :
Erreur : la méthode a détecté que 0 n'est pas un chiffre.
Erreur : la méthode a détecté que 9 n'est pas un chiffre.
8 test(s) ont réussi sur un total de 10 tests réalisés.
   ==>  2 tests ont échoué.


   ==> b) Tests autres caractères :
11 test(s) ont réussi sur un total de 11 tests réalisés.
   ==>  Tous les tests sont OK
```

1) Définir dans la classe ***OutilChaineTest*** un tableau constant contenant tous les chiffres :

```
/** jeu de test pour la méthode estChiffre : valeurs correctes */
private static final char[] CHIFFRE = { '0', '1', '2', '3', '4', '5',
                                        '6', '7', '8', '9'};
```

2) Définir dans la classe ***OutilChaineTest*** un tableau constant contenant des caractères qui ne sont pas des chiffres. Il faudrait trouver une dizaine de valeurs à tester et choisir certaines d'entre elles de manière pertinente (peut-être des valeurs proches de '0' ou '9' dans l'ordre des caractères ? pour effectuer des tests aux limites).

```
/** jeu de test pour la méthode estChiffre : valeurs incorrectes */
private static final char[] NON_CHIFFRE = {  . . . . . .};
```

3) Coder la méthode qui effectue le test unitaire de ***estChiffre***. Veillez à correctement choisir le nom de cette méthode.

4) Exécuter la classe de test de manière à afficher le rapport de test de la méthode ***estChiffre*** (voir exemple ci-dessus).

5) Pour vérifier que les affichages de votre méthode de test sont corrects y compris si la méthode ***estChiffre*** comporte des erreurs, exécuter la classe de tests unitaires avec la méthode ***estChiffre*** des classes ***OutilChaineEr1*** et ***OutilChaineEr2***. Si besoin corriger votre méthode de test unitaire.

Il n'est pas demandé de trouver les erreurs de programmation dans les 2 classes en erreur. Toutefois, vous pourrez éventuellement essayer de localiser les erreurs, à la fin du TP, s'il vous reste du temps.

ETAPE 2 – Test unitaire de la méthode *estHoraireValide*

La méthode ***estHoraireValide*** prend en paramètre une chaîne de caractères et détermine si celle-ci contient un horaire valide. Pour que l'horaire soit valide, le nombre d'heures doit être compris entre 0 et 23 et le nombre de minutes entre 0 et 59. De plus, la chaîne doit être dans le format : cchcc.

Comme dans l'étape précédente, il faudra définir un jeu de tests sous la forme de 2 tableaux contenant des valeurs à tester, donc ici des chaînes de caractères. Le premier contiendra des horaires pour lesquels la méthode est supposée renvoyer vrai comme résultat, et le deuxième des valeurs pour lesquelles la méthode doit renvoyer faux.

Il faudra choisir de manière pertinente ce jeu de tests.

Si la méthode ***estHoraireValide*** est bien programmée, on souhaite que la méthode de test affiche le résultat suivant :

```
Test de la méthode de estHoraireValide
---------------------------------------

   ==> a) Tests avec des horaires valides :
10 test(s) ont réussi sur un total de 10 tests réalisés.
   ==>  Tous les tests sont OK


   ==> b) Tests avec des chaînes invalides :
30 test(s) ont réussi sur un total de 30 tests réalisés.
   ==>  Tous les tests sont OK
```

Supposons qu'il y ait des erreurs de programmation, l'affichage souhaité est le suivant :

```
Test de la méthode de estHoraireValide
---------------------------------------

   ==> a) Tests avec des horaires valides :
Résultat inattendu : 00h00 a été considérée comme invalide.
Résultat inattendu : 23h59 a été considérée comme invalide.
Résultat inattendu : 08h02 a été considérée comme invalide.
Résultat inattendu : 10h58 a été considérée comme invalide.
Résultat inattendu : 11h59 a été considérée comme invalide.
Résultat inattendu : 10h00 a été considérée comme invalide.
Résultat inattendu : 07h02 a été considérée comme invalide.
Résultat inattendu : 15h01 a été considérée comme invalide.
2 test(s) ont réussi sur un total de 10 tests réalisés.
   ==>  8 tests ont échoué.


   ==> b) Tests avec des chaînes invalides :
Résultat inattendu : 11h23  a été considérée comme valide.
Résultat inattendu : 12h345 a été considérée comme valide.
28 test(s) ont réussi sur un total de 30 tests réalisés.
   ==>  2 tests ont échoué.
```

1) Définir dans la classe ***OutilChaineTest*** un tableau constant contenant une dizaine d'horaires valides, choisis pour certains d'entre eux de manière pertinente (prendre des valeurs aux limites et des valeurs médianes) :

```
/** jeu de test avec des chaînes valides (pour un horaire) */
private static final String[] HORAIRE_VALIDE = {"00h00", . . . .} ;
```

2) Définir dans la classe ***OutilChaineTest*** un tableau constant ***HORAIRE_INVALIDE*** contenant des horaires invalides, choisis de manière pertinente (prendre des valeurs aux limites et des valeurs médianes). Il faudrait trouver de 20 à 30 valeurs à tester.

3) Coder la méthode qui effectue le test unitaire de ***estHoraireValide***. Veillez à correctement choisir le nom de cette méthode.

4) Exécuter la classe de test de manière à afficher le rapport de test de la méthode ***estHoraireValide*** (voir exemple ci-dessus).

5) Pour vérifier que les affichages de votre méthode de test sont corrects y compris si la méthode ***estHoraireValide*** comporte des erreurs, exécuter la classe de tests unitaires avec la méthode ***estHoraireValide*** des classes ***OutilChaineEr1*** et ***OutilChaineEr2***.

ETAPE 3 – Test unitaire de la méthode *convertir* des minutes en chaîne

La première méthode ***convertir*** prend en paramètre un entier et le convertit en chaîne de caractères contenant un horaire dans le format cchcc. L'entier argument est supposé être un nombre de minutes, plus précisément la conversion en minutes d'un horaire de la journée. Par exemple 10 h 15 minutes est égal à 615 minutes. Donc si la valeur 615 est donnée en argument de la méthode, celle-ci renvoie comme résultat « 10h15 ».

Si les minutes données en argument ne sont pas valides, donc négatives ou trop grandes (supérieures à 1439 qui est la conversion de 23h59), la méthode doit renvoyer la constante RESULTAT_ERREUR.

Comme dans l'étape précédente, il faudra définir 2 jeux de tests, l'un pour avec des valeurs valides pour lesquelles la méthode est supposée renvoyer un horaire. L'autre avec des valeurs invalides pour les quelles la méthode est supposée renvoyer la constante RESULTAT_ERREUR.

Pour les tests avec des valeurs valides, on s'appuiera sur 2 tableaux :
- un tableau contenant des entiers à convertir en horaires chaîne de caractères
- un tableau contenant les horaires (en chaîne) résultat de la conversion. Pour ce deuxième tableau, vous pouvez réutiliser le tableau de l'étape 2. Le tableau contenant les minutes sera donc défini en adéquation avec celui-ci.

Pour les tests avec des valeurs invalides, le résultat renvoyé par la méthode étant toujours le même, donc la constante RESULTAT_ERREUR, un seul tableau suffira : un tableau contenant des minutes (donc des entiers) qui ne peuvent pas être converties en horaire valide. Il faudra choisir de manière pertinente ce jeu de tests.

Si la méthode ***convertir*** est bien programmée, on souhaite que la méthode de test affiche le résultat suivant :

```
Test de la méthode convertir minutes en horaire (entier -> chaîne)
```

```
------------------------------------------------------------------
   ==> a) Tests avec des horaires en minutes valides :
10 test(s) ont réussi sur un total de 10 tests réalisés.
   ==>  Tous les tests sont OK


   ==> b) Tests avec des horaire en minutes invalides :
8 test(s) ont réussi sur un total de 8 tests réalisés.
   ==>  Tous les tests sont OK
```

Supposons qu'il y ait des erreurs de programmation, l'affichage souhaité est le suivant :

```
Test de la méthode convertir minutes en horaire (entier -> chaîne)
------------------------------------------------------------------
   ==> a) Tests avec des horaires en minutes valides :
Résultat inattendu : 1439 a été converti en ERREUR format au lieu de 23h59
9 test(s) ont réussi sur un total de 10 tests réalisés.
   ==>  1 tests ont échoué.


   ==> b) Tests avec des horaire en minutes invalides :
8 test(s) ont réussi sur un total de 8 tests réalisés.
   ==>  Tous les tests sont OK
```

1) Définir dans la classe ***OutilChaineTest*** un tableau constant ***CONVERSION*** contenant la conversion en minutes des horaires contenus dans le tableau ***HORAIRE_VALIDE*** .

2) Définir également un tableau constant ***CONVERSION_IMPOSSIBLE*** contenant des entiers qui ne sont pas convertibles en horaires valides. Choisir des valeurs pertinentes.

3) Coder la méthode qui effectue le test unitaire de ***convertir***.

4) Exécuter la classe de test de manière à afficher le rapport de test de la méthode ***convertir***.

5) Pour vérifier que les affichages de votre méthode de test sont corrects y compris si la méthode ***convertir*** comporte des erreurs, exécuter la classe de tests unitaires avec la méthode ***convertir*** des classes ***OutilChaineEr1*** et ***OutilChaineEr2***.

ETAPE 4 – Test unitaire de la méthode *convertir* un horaire en chaîne en minutes

Suivre la même démarche que dans l'étape précédente pour tester la deuxième méthode ***convertir***, celle qui effectue la conversion inverse de la précédente. Donc à partir d'un horaire exprimé en chaîne de caractères, la méthode renvoie sa conversion en minutes. Si l'horaire argument n'est pas valide, la méthode doit renvoie la constante ***CODE_ERREUR***.

ETAPE 5 – BONUS (l'intérêt de cette étape réside dans le choix des jeux de test)

Les autres méthodes de la classe ***OutilHoraire*** effectuent des saisies au clavier : ***saisirEntierIntervalle***, ***saisirHoraire***, ***saisir2HorairesOrdonnes***. Il faudrait donc les tester manuellement.

1) Pour chacune d'elles choisir des jeux de tests pertinents.
2) Les méthodes de test interactives sont déjà programmées dans le fichier nommé ***en_vrac.txt***. Intégrer ce code à la classe ***OutilHoraireTest***. Exécuter ces méthodes pour tester le bon fonctionnement des méthodes de saisie