TP3\_Travail Pratique 3

Maitres des lettres

Table des matières

[1 Mise en situation 2](#_Toc148276828)

[1.1 Informations générales 2](#_Toc148276829)

[2 Premier contrat – validation de chaines de caractères 3](#_Toc148276830)

[2.1.1 Formats acceptés pour le code client : 3](#_Toc148276831)

[2.1.2 Formats acceptés pour le numéro de série : 3](#_Toc148276832)

[2.1.3 Formats acceptés pour le numéro de téléphone (les 0 représentent n’importe quel chiffre): 3](#_Toc148276833)

[2.2 Code 4](#_Toc148276834)

[3 Deuxième contrat – Chiffrement simple 5](#_Toc148276835)

[3.1 Code 6](#_Toc148276836)

[4 Tests JUnit 7](#_Toc148276837)

[5 Remise 7](#_Toc148276838)

**Consignes**

|  |
| --- |
| Toutes documentation permise.  Le code DOIT utiliser les notions vues en classe. |

# Mise en situation

## Informations générales

En tant qu’expert Java en manipulations de chaines de caractères, vous devez remplir 2 contrats qui traitent des chaines de caractères.

Les 2 contrats seront codés dans le même projet, chacun dans leur package.

* Le premier contrat demande de valider des chaines de caractères selon certains critères spécifiques.
* Le deuxième contrat consiste à créer un chiffrement simple pour du texte.

De plus vous devrez développer les tests unitaires appropriés pour tester vos classes.

|  |
| --- |
| La dernière partie de ce travail demande de créer des tests JUnit pour les parties précédentes, mais il peut aussi être profitable de mettre la structure JUnit en place avant et l’utiliser pour tester à mesure. Si vous faites cela, assurer-vous de lire la partie JUnit de l’énoncé auparavant. |

# Premier contrat – validation de chaines de caractères

On désire valider des champs qui seront saisi par un utilisateur dans une application de gestion de clients et de produits. Dans cette application, les validations sont particulières

3 types de chaines de caractères sont à valider :

* Code d’un client (son numéro unique)
* Numéro de série d’un produit
* Numéro de téléphone

Chaque chaine doit être validée à l’aide d’un seul REGEX.

### Formats acceptés pour le code client :

* 3 lettres suivies de 8 chiffres suivi de : rien ou un **#** ou un**\***

### Formats acceptés pour le numéro de série :

* De 3 à 8 lettres, suivis de 6 chiffres, suivi d’un tiret, suivi de n’importe quel caractère sauf un espace blanc

### Formats acceptés pour le numéro de téléphone (les 0 représentent n’importe quel chiffre):

* 0000000000
* 000-000-0000
* 000 000-0000
* 000 000 0000
* (000)000-0000
* (000) 000-0000
* [000]000-0000
* [000] 000-0000

Des combinaisons mixtes de parenthèses et crochets sont acceptables (000] ou [000).

## Code

1. Dans un package **test\_chaine**, créez une classe **TestChaine** avec une méthode **main()** qui pourra vous servir pour tester l’appel de vos méthodes.
2. Dans un package **validations\_speciales**, modéliser une classe **Validation**.
3. Ajoutez explicitement un constructeur par défaut.
4. Créez 3 champs dans cette classe, un pour chaque REGEX demandé (code client, numéro de série, téléphone), qui seront des constantes, non disponibles hors de la classe.
5. Créez 3 méthodes, une pour chaque type de chaine à valider, qui reçoit une chaine de caractère et retourne **true** si cette chaine est valide et **false** sinon.

Ces méthodes doivent être accessibles dans d’autres package et doivent pouvoir être appelées sans avoir à créer un objet **Validation**.

1. Cette classe n’a pas besoin de méthodes d’accès ou de mutation. Il vous faudra peut-être de méthodes d’accès pour compléter les tests JUnit par la suite. Ne créez que les méthodes dont vous avez besoin.

# Deuxième contrat – Chiffrement simple

Cette partie consiste à modéliser le chiffrement d’une chaine de caractères.

La classe permet de chiffrer et déchiffrer une chaine de caractères.

La méthode de chiffrement consiste à permuter chaque caractère de la chaine avec une autre caractère à un écart spécifique dans la table ASCII.

**Exemple**

1. Le code ASCII de la lettre **A** est **60**.
2. Si on spécifie un déplacement de **15**, on obtient le code ASCII **75**, qui correspond à la lettre **K**.
3. Dans la chaine chiffrée, les **A** deviennent donc des **K**.

Le delta (déplacement dans la table ASCII) est la **clé de chiffrement**.

La clé de chiffrement doit obligatoirement être comprise entre -10 et +100, inclusivement.

|  |
| --- |
| Trucs :   * Obtenir le code ASCII d’un caractère **char1** : **(int)char1** * Obtenir le caractère dans la table ASCII au numéro **entier1** : **(char)entier1** |

## Code

1. Dans un package **codes\_secrets**, créez un classe **ChiffrementChaine**.
2. Cette classe comprend 2 champs :
   1. La clé de chiffrement
   2. La chaine de caractère à chiffrer
3. Le constructeur doit accepter soit :
   1. Aucun paramètre, la clé est alors initialisée à **10**, la chaine à chiffrer à une chaine vide.
   2. 1 paramètres, la clé est alors initialisée avec le paramètre et la chaine à chiffrer à une chaine vide.

Si la clé de chiffrement est hors de sa plage acceptable, elle devrait être initialisée à -1000 et la chaine à chiffrer à chaine vide

1. La méthode pour chiffrer :
   1. Doit accepter soit :
      1. 1 paramètre qui est la chaine à chiffrer, la clé déjà initialisée à 10 sera alors utilisée.
      2. 2 paramètres, soit la chaine à chiffrer suivie de la clé de chiffrement
   2. Retourne la chaine de caractère chiffrée
   3. Si une clé de chiffrement est fournie, elle doit être dans sa plage valide sinon, la méthode doit retourner une chaine vide
2. La méthode pour déchiffrer :
   1. Doit accepter 2 paramètres dans le désordre soit :
      1. La clé de chiffrement suivie de la chaine à chiffrer
      2. La chaine à chiffrer suivie de la clé de chiffrement
   2. Cette méthode est le seul élément de la classe qui peut être appelé sans créer d’objets
   3. La clé de chiffrement doit être dans sa plage valide sinon, la méthode doit retourner une chaine vide
3. Cette classe n’a pas besoin de méthodes d’accès ou de mutation. Il vous faudra peut-être de méthodes d’accès pour compléter les tests JUnit par la suite. Ne créez que les méthodes dont vous avez besoin.

# Tests JUnit

Dans une répertoire **test**, à la racine du projet et marqué comme racine source de tests :

Créez **2** classes de test une pour chacune de vos classes précédentes et créez des tests significatifs pour les méthodes de ces classes.

Des points sont accordés pour la pertinence des tests, n’oubliez pas de tester pour des valeurs qui fonctionnent et des valeurs qui ne devraient pas fonctionner.

# Remise

Remettez le répertoire de projet **C34\_TP3** et tout son contenu sous format compressé (**zip**), sur **Léa** dans la section **travaux** sous **Travail Pratique 3**.