L02D\_Modificateurs de non-accès

Table des matières

[1 Questions 2](#_Toc142135884)

[2 Code 2](#_Toc142135885)

[2.1 Outils de validation pour un formulaire 2](#_Toc142135886)

[2.2 Compteur et numéro séquentiel 3](#_Toc142135887)

[2.3 Gestion de facturation 4](#_Toc142135888)

# Questions

1. Revisitez le laboratoire L02C et identifiez quels champs ou méthodes seraient des bons candidats pour un modificateur **static** et expliquez pourquoi.

|  |
| --- |
|  |

# Code

## Outils de validation pour un formulaire

1. On désire réunir dans une classe des outils pour valider les champs dans les formulaires d’une application (lors de la création d’un compte client par exemple).
2. Créez un nouveau projet **L02D-ModificateursNonAcces**
3. Créez un nouveau nouveau package **client.acme\_cvm.ca**
4. Créez une classe **Client** avec les champs suivants:
   1. nomComplet (type String)
   2. nas (type String)
   3. age (type int)
   4. codePostal (type String)
5. Dans la classe client, créez une méthode **setClient** :
   1. Cette méthode ne retourne rien et accepte 4 paramètres : nom complet, nas, age et codePostal, de même types que les champs
   2. La méthode assigne aux champs la valeur correspondante aux paramètres reçus
6. Créez une classe **OutilsValidation** avec :

* Aucun champ
* Une méthode **static** **longueurNomComplet** :
  1. Accepte en paramètre 2 String : **nom** et **prenom**
  2. Retourne un booléen, vrai si le nombre de lettre du nom + nombre de lettres du prénom +1 (espace entre prénom et nom) ne dépasse pas 40.
* Une méthode **static** **ageLegal** :
  1. Accepte 1 paramètre **age**
  2. Retourne un booléen, vrai si age est entre 18 et 100 ans, inclusivement
* Une méthode **static** **nasValide** :
  1. Accepte 1 paramètre **nas** de type String
  2. Retourne 1 booléen, vrai si :
     1. Le paramètre a 9 caractères
     2. Tous les caractères sont numériques
* Une méthode **codePostalValide** :
  1. Accepte 1 paramètre **codePostal** de type String
  2. Retourne 1 booléen, vrai si :
     1. Le paramètre a 6 caractères
     2. Les caractères 1, 3 et 5 sont des lettres
     3. Les caractères 2, 4 et 6 sont des chiffres
     4. **Truc** : les méhodes static Character.isDigit() et Character.isLetter() acceptent un caractères en paramètre et retournent un booléen ai le caractère est respectivement un chiffre et une lettre.

1. Créez une classe **ApplicationClient** avec une méthode main
   1. Créez des variables locales pour : prenom, nom, age, nas et codePostal et assignez des valeurs de test à ces variables.
   2. Affichez à l’écran si chacun des champs sont valides ou nom en utilisant les méthodes de la classe **OutilsValidation** avec la forme **OutilsValidation.*nomDeMethode(.* . .)**, sans créer d’objet.
   3. Essayez différentes valeurs pour tester vos méthodes.
   4. Note : les classes d’outils sont une des applications typiques de **static**.

Note : avez-vous assigné les modificateurs d’accès selon les meilleures pratiques?

## Compteur et numéro séquentiel

On modélise une application conçue pour gérer les invités dans un événement (concert, événement sportif, parc d’attractions etc.).

On désire entre autres savoir combien d’invités sont présents et leur assigner un numéro unique qui leur servira de passe pour les différents services.

Continuez le travail dans le même projet **L02D-ModificateursNonAcces**.

1. Créez un nouveau package **evenement.acme\_cvm.ca**
2. Créez une classe **Invite** avec les champs suivants:
   1. static nbreInvites (type int)
   2. static sequenceInvite (type int)
   3. codeInvite (type String)
   4. nomComplet (type String)
3. Dans la classe **Invite**, créez des méthodes d’accès pour les 4 champs
4. Créez une méthode de mutation pour les champs **nbreInvites** et **sequenceInvite** (pour initialiser leurs valeurs au départ)
5. Dans la classe **Invite**, créez une méthode **setInvite** :
   1. Ne retourne pas de valeur
   2. Accepte 1 paramètre, soit le nom complet de l’invité
   3. Assigne la valeur du paramètre au champ **nomComplet**
   4. Assigne une valeur au champ **codeInvite** suivant le modèle : 3 premières lettres du nom + valeur du champ **sequenceInvite** (convertit en texte)
   5. Incrémente **sequenceInvite** de 1
   6. Incrémente **nbreInvites** de 1
6. Créez une classe **ApplicationEvenement** avec une méthode main
   1. Initialisez la valeur de **nbreInvites** à 0 (pas besoin de créer un objet)
   2. Initialisez la valeur de **sequenceInvite** à 12300000 (pas besoin de créer un objet)
   3. Créez un objet de la classe Invite, puis initialisez-le avec la métode setInvite() en lui donnant un nom de votre choix.
   4. Répétez l’opération 2 autres fois pour obtenir en tout 3 objets Invite avec des noms différents
   5. Affichez à l’écran :
      1. Le nom et code pour chacun des invités
      2. Le nombre total d’invités, sans utiliser d’objet
      3. La prochaine valeur de sequenceInvite qui sera utilisée, sans utiliser d’objet

## Gestion de facturation

Dans une application de vente pour un magasin, on modélise la partie qui permet de gérer des factures.

L’application est lancée au début de la journée et s’exécute jusqu’à la fermeture du magasin, le soir.

Pour cette application on commence avec 2 classes qui ont les champs suivants (lisez tout le texte avant de commencer à coder) :

Classe **Magasin**

* nomEntreprise : "ACME CVM", le nom complet de l’entreprise, apparaitra sur toutes les factures
* adresseEntreprise : "1 rue principale, Montréal, Québec H0H 0H0" adresse de l’entreprise, sur une seule ligne, apparaitra sur toutes les factures
* tauxTaxeFed : 5%, le taux de taxes fédéral, appliqué au sous-total des factures
* tauxTaxeProv : 9,975%, le taux de taxes provincial, appliqué au sous-total des factures
* noTPS : "123456789 RT 0001", le numéro de taxes de l’entreprise au gouvernement fédéral, inclus sur toutes les factures
* noTVQ : "1234567890 TQ 0001", le numéro de taxes de l’entreprise au gouvernement provincial, inclus sur toutes les factures

Classe **Facture**

* nbreClients : le nombre de clients (1 par facture) courant pour la journée, initialisé à 0
* ventesCourantes : le montant total des ventes, courant pour la journée, initialisé à 0
* noFactureCourant : le numéro de facture actuel, commence à 0 au début de la journée et s’incrémente de 1 à chaque facture
* noFacture : numéro unique (numérique) pour une facture dans la journée
* sousTotal : le montant des achats fait pour cette facture, avant les taxes
* taxeFed : le montant de la taxe fédéral = sousTotal \* tauxTaxeFed
* taxeProv : le montant de la taxe provinciale = sousTotal \* tauxTaxeProv
* Total : le montant total à payer = sousTotal + taxeFed + taxeProv

Note : la liste des items achetés avec leur prix et quantités respectives seraient programmés dans une autre classe, pas demandé dans cet exercice.

Pour chaque champ des 2 classes ci-dessus, identifiez : le type approprié, si le champ devrait être **static** et si le champ devrait être **final**, et expliquez pourquoi.

Remplissez le tableau qui suit, puis comparez avec la réponse sur la page d’après.

Classe **Magasin**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Champ** | **Type** | **Static**  **(Oui/Non)** | **Final**  **(Oui/Non)** | **Explication** |
| nomEntreprise |  |  |  |  |
| adresseEntreprise |  |  |  |  |
| noTPS |  |  |  |  |
| noTVQ |  |  |  |  |
| tauxTaxeFed |  |  |  |  |
| tauxTaxeProv |  |  |  |  |

Classe **Facture**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Champ** | **Type** | **Static**  **(Oui/Non)** | **Final**  **(Oui/Non)** | **Explication** |
| nbreClients |  |  |  |  |
| ventesCourantes |  |  |  |  |
| noFactureCourant |  |  |  |  |
| noFacture |  |  |  |  |
| sousTotal |  |  |  |  |
| taxeFed |  |  |  |  |
| taxeProv |  |  |  |  |
| total |  |  |  |  |

La clé pour la prise de décision des modificateurs de non-accès ici, est la durée de vie de l’application : 1 journée. Toute donnée qui ne changera pas au typiquement au cours d’une journée peut être considéré comme une constante.

Classe **Magasin**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Champ** | **Type** | **Static**  **(Oui/Non)** | **Final**  **(Oui/Non)** | **Explication** |
| nomEntreprise | String | Oui | Oui | Static pour y avoir accès sans créer d’objet  Final parce que ne changera pas au cours de la durée de l’application (1 journée) |
| adresseEntreprise | String | Oui | Oui | Static pour y avoir accès sans créer d’objet  Final parce que ne changera pas au cours de la durée de l’application (1 journée) |
| noTPS | String | Oui | Oui | Static pour y avoir accès sans créer d’objet  Final parce que ne changera pas au cours de la durée de l’application (1 journée) |
| noTVQ | String | Oui | Oui | Static pour y avoir accès sans créer d’objet  Final parce que ne changera pas au cours de la durée de l’application (1 journée) |
| tauxTaxeFed | Double | Oui | Oui | Static pour y avoir accès sans créer d’objet  Final parce que ne changera pas au cours de la durée de l’application (1 journée) |
| tauxTaxeProv | Double | Oui | Oui | Static pour y avoir accès sans créer d’objet  Final parce que ne changera pas au cours de la durée de l’application (1 journée) |

Normalement, dans le main on créerait un unique objet de type magasin en initialisant le nom, l’adresse et les 2 numéros de taxes. Dans ce contexte, ces champs n’auraient pas besoin d’être static, puisqu’il n’y a qu’un seul objet pour le magasin de toute façon.

Les 2 taux de taxe sont des parfaits exemples de constantes dans une telle application, donc static et final. Un exemple où les taux de taxes ne seraient pas des constantes serait une plateforme de e-commerce internationale, où les taxes changent dépendant du pays/région.

Classe **Facture**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Champ** | **Type** | **Static**  **(Oui/Non)** | **Final**  **(Oui/Non)** | **Explication** |
| nbreClients | int | Oui | Non | Static pour y avoir accès sans créer d’objet, compteur qui appartient à la classe et pas à une seule facture  Pas final parce qu’il change |
| ventesCourantes | double | Oui | Non | Static pour y avoir accès sans créer d’objet, total cumulé qui appartient à la classe et pas à une seule facture  Pas final parce qu’il change |
| noFactureCourant | int | Oui | Non | Static pour y avoir accès sans créer d’objet, compteur qui appartient à la classe et pas à une seule facture  Pas final parce qu’il change |
| noFacture | int | Non | Non | Spécifique à une facture et montant change d’une facture à l’autre |
| sousTotal | double | Non | Non | Spécifique à une facture et montant change d’une facture à l’autre |
| taxeFed | double | Non | Non | Spécifique à une facture et montant change d’une facture à l’autre |
| taxeProv | double | Non | Non | Spécifique à une facture et montant change d’une facture à l’autre |
| total | double | Non | Non | Spécifique à une facture et montant change d’une facture à l’autre |

1. Continuez le travail dans le même projet **L02D-ModificateursNonAcces**.
2. Créez un package ventes.acme\_cvm.ca
3. Dans le package **ventes.acme\_cvm.ca**, créez les 2 classes avec leurs champs.
4. Modifiez au besoin la notation des champs qui sont final et considérés des constantes (majuscules, mots séparés par un \_ )
5. Dans la classe **Magasin**, ajoutez :

* 1 méthode d’accès pour chacun des champs
* Essayez de faire une méthode de mutation pour le champ **NOM\_ENTREPRISE** : ça ne fonctionne pas, le champ de modificateur final ne peut pas être modifié.
* Initialisez les valeurs de tous les champs de **Magasin** directement dans la déclaration des champs, en prenant les valeurs au début de l’énoncé de cet exercice.

1. Dans la classe Facture ajoutez :

* 1 méthode d’accès pour chacun des champs
* méthodes de mutation pour les champs : nbreClients, ventesCourantes, noFactureCourant
* 1 méthode de mutation **setFacture** pour assigner les valeurs à 1 facture en respectant les consignes suivantes :
  + Reçoit 1 paramètre **sousTotal**, le sous-total
  + Assigne la valeur du paramètre sousTotal au champ sousTotal
  + Calcule et assigne les valeurs pour : **taxeFed**, **taxeProv** et **total**
  + Assigne la prochaine valeur de **noFactureCourant** à **noFacture**
  + Incrémente **nbreClients** de 1
  + Ajoute le **total** de la facture à **ventesCourantes**
  + Incrémente le **noFactureCourant** de 1

1. Dans le package **ventes.acme\_cvm.ca**, créez une classe **AppVente** avec une méthode main et dans cette classe :

* Créez un objet **Magasin**
* Initialisez **nbreClients** à 0
* Initialisez **ventesCourantes** à 0
* Initialisez **noFactureCourant** à 1
* Créez successivement 3 factures, avec différents sous-total
* Affichez la troisième facture avec les informations pour tous les champs
* Affichez à l’écran les valeurs de **nbreClients**, **ventesCourantes**, **noFactureCourant**
* Affichez à l’écran la valeur de **noFactureCourant** pour la première facture puis pour la troisième facture. Que remarquez-vous?

NOTES :

1. En plus des modificateurs de non-accès, avez-vous porté attention aux modificateurs d’accès?
2. Dans le programme principal, avez-vous appelé les méthodes de classes pour les informations static?