Programmation Objet en Java - Feuille de TP 7

Exercice 1.

a) Écrire une classe `City` pour représenter des villes. Une `City` aura un champ `name` pour indiquer son nom, un champ `populationNumber` pour le nombre d'habitants et un champ `country` pour indiquer le pays. Le champ `name` sera non mutable.

b) Écrire une classe `CityTest` dans laquelle on crée deux villes et on les affiche.

c) Une ville ne doit jamais avoir une population inférieure à 1000. Comment faire pour être sûr que cela n'arrive jamais ? Ajouter dans le code ce qu'il faut pour empêcher la construction d'une ville de moins de 1000 habitants.

d) On voudrait aussi indiquer dans le message d'erreur le nombre d'habitants qui a posé problème. Comment faire ?

e) Faire en sorte qu'on ne puisse pas créer une ville dont le nom a moins de 3 caractères. L'exception levée devra contenir le message : "The city *name* has less than 3 characters".

f) Ajouter une méthode `cityRank` qui renvoie un réel positif calculé comme la somme de la longueur du nom du pays et du troisième caractère (on peut additionner des `char`) du nom de la ville, le tout multiplié par le logarithme (`Math.log`) de `populationNumber/1000`.

g) Ajouter un accesseur pour la population ainsi que 2 méthodes `increasePopulation` et `decreasePopulation` qui augmente et diminue respectivement la population de 1000. Les deux méthodes doivent renvoyer le nombre d'habitants actuel (après l'opération).

h) On aimerait bien utiliser une exception spécifique au problème du nom de la ville. Écrire une classe `CityNameException` dérivant de `RuntimeException` et l'exception précédente par celle-ci dans le constructeur de `City`. Que doit-on faire dans `CityNameException`, si on veut pouvoir construire une telle exception avec un message ?

i) Essayer de faire dériver `CityNameException` de `Exception` au lieu de `RuntimeException`. Pourquoi a-t-on un problème au niveau du constructeur désormais ? Pourquoi n'est-ce pas arrivé avant ? *Remarque : c'est une bonne raison pour que l'on essaie au maximum d'utiliser les exceptions déjà existantes.*

j) Pour corriger le problème, faire en sorte que le constructeur *déclare* l'exception qui risque d'être lancée.

k) Que se passe-t-il dans la classe de test maintenant? Pour chaque création de ville, capturer l'exception qui pose problème dans un bloc `try/catch`. Dans le bloc `catch`, on se contentera d'afficher que la ville n'a pas pu être créée.

l) Pour chaque ville, capturer également l'exception liée à la population, et faire en sorte d'afficher la *stack trace*, mais sans interrompre le programme. Tester avec toutes les combinaisons de problèmes possibles pour un ville.

m) **Capturer une exception qui n'a pas besoin de l'être** (une `RuntimeException` ou dérivée) **pour éviter une erreur est une très mauvaise pratique.** Dans la cas présent, c'est à l'utilisateur du constructeur de faire attention à ne pas provoquer d'`IllegalArgumentException`. Et pour cela, il faut qu'il sache que le constructeur peut lever cette exception et pourquoi, en lisant sa documentation.

   (a) Écrire la `javadoc` du constructeur en spécifiant les exceptions potentielles. Il est possible d'auto-générer le bloc de commentaires qui permet de faire ça grâce au raccourci `alt + shift + J`.

   (b) Trouver dans les menus comment générer la `javadoc`. Vous devriez avoir un résultat qui ressemble à ça :

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	decreasePopulation()	Decrease the population of this city by 1000
void	increasePopulation()	Increase the population of this city by 1000
int	populationNumber()	The population of this city
String	toString()	

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Details

City

```
public City(String name,
            int populationNumber)
     throws fr.dut.city.CityNameException
```

Constructs a new City

Parameters:

name - the name of the city

populationNumber - the number of inhabitants

Throws:

NullPointerException - if the name is null

CitNameException - if the city name is less than 3 characters

IllegalArgumentException - if populationNumber is under 1000

Exercice 2. On considère une l'interface suivante pour manipuler une pile (*Last In First Out*) d'entiers **positifs** (des `int`). Il s'agit d'une structure dans laquelle on peut empiler et dépiler des éléments. On peut également consulter le sommet de la pile, tester si la pile est vide ou pleine. On traite les cas où les opérations sont impossibles avec des exceptions.

```java
public interface Stack {
  /**
   * Tests if this stack is empty.
   * @return true if and only if this stack contains no items; false otherwise.
   */
  boolean isEmpty();

  /**
   * Tests if this stack is full.
   * @return true if and only if this stack can not contain more items; false otherwise.
   */
  boolean isFull();

  /**
   * Pushes an item (int) onto the top of this stack.
   * @param item the element to push
   * @throws IllegalStateException if this stack is full.
   */
  void push(int item);

  /**
   * Looks at the int at the top of this stack without removing it from the stack.
   * @return the int at the top of the stack.
   * @throws IllegalStateException if this stack is empty.
   */
  int peek();

  /**
   * Removes the int at the top of this stack and returns that int.
   * @return the int at the top of the stack.
   * @throws IllegalStateException if this stack is empty.
   */
  int pop();
}
```

a) Écrire une classe `ListStack` qui implémente l'interface `Stack`. La classe `ListStack` contiendra une `ArrayList` et la pile ne sera jamais pleine.

b) Écrire une classe pour tester votre implantation de l'interface `Stack`.

c) (optionnel) Utiliser la pile pour écrire une méthode `isWellFormed` qui vérifie qu'un mot pouvant contenir des parenthèses et des crochets est bien formé (c'est à dire que chaque symbole ouvrant est bien associé au symbol fermant correspondant et que les imbrications sont correctes). Par exemple, `"abc"`, `"(abc)"`, `"ab[cd]ef"`, `"a[b]c(d)e"`, `"a((b)c)d"` et `"a(b[c()e]f)g"` sont bien formés, alors que `"("`, `"abc)"`, `"ab)c"`, `"a(b]c"`, `"ab(c)d)e"` et `"a(b[c)d]e"` ne le sont pas.