TD Simulation d'afficheurs lumineux

Le but de cet exercice est de simuler en Java les afficheurs lumineux qu'on voit un peu partout et qui font circuler un texte en boucle.

Exercice 1 : Les afficheurs lumineux

Un *afficheur lumineux* (*led displayer*) est un objet en charge de l'affichage d'un message d'une longueur non nulle l. Ce message s'affiche sur un écran d'une largeur ($\neq 0$) qui peut être plus petite ou plus grande que l. L'écran n'affiche donc éventuellement qu'une portion du message.
Un afficheur est initialement défini par la largeur de son écran, le message étant vide.

Il est possible de modifier le texte de l'afficheur (celui qui appparaît sur l'écran) en décalant (*shift*) d'une position vers la gauche le texte du message qui apparaît à l'écran. Quand le dernier caractère du message vient d'entrer dans la partie affichée, au prochain décalage on reprend le message au début et c'est le premier caractère du message qui y entre à son tour, et ainsi de suite.

Le code de la classe `LedDisplayer` pourrait donc ressembler à :

```
package displayer;
public class LedDisplayer {
  ...
  /** get the (full) message of this displayer
   * return the (full) message of this displayer*/
  public String getMessage()   { ... }
  /** set the new message to display, a call to textOnScreen() is now blank until next shift()
   * @param message the new message */
  public void setMessage(String message) { ...  }
  /** @return the screen width */
  public int getScreenWidth() { ... }
  /**  shift message by one character  */
  public void shift() { ... }
  /** return the text that appears on the screen,
   *           it is always composed of getScreenWidth() characters
   * @return the text that appears on the screen */
  public String textOnScreen() { ... }

}
```

On utilise un objet de la classe `DisplayerController` pour et cadencer les décalages d'un `LedDisplayer` : à chaque « seconde » le contrôleur fait un appel à la méthode `shift` de l'affficheur. Le message défile ainsi en boucle.
Pour fixer les idées, une classe `DisplayerController` est fournie ci-dessous, l'invocation

```
new DisplayerController(new LedDisplayer(5)).tryIt("Abcd",9);
```

produit alors la trace sur la droite.

```
package displayer;
public class DisplayerController {
   private LedDisplayer displayer;

   public DisplayerController(LedDisplayer disp) { this.displayer = disp; }

   public void tryIt(String message, int nbSeconds) {
      this.displayer.setMessage(message);
      for (int i = 0; i < nbSeconds; i++) {
         this.displayer.shift();
         System.out.println("|" + displayer.textOnScreen() + "|");
      }
   }
}
```

```
|    A|
|   Ab|
|  Abc|
| Abcd|
|AbcdA|
|bcdAb|
|bdAbc|
|dAbcd|
|AbcdA|
```

Les afficheurs « simples »

Q 1 . Complétez le code de la classe `LedDisplayer`.

Les afficheurs avec latence

On remarque que pour les afficheurs de l'exercice précédent il est difficile de voir où se termine le message. Pour éviter ce problème, on veut une nouvelle classe d'afficheurs pour lesquels on pourra spécifier lors de leur création un « *temps de latence* » entre l'entrée du dernier et celle du premier caractère. Ce temps de latence sera exprimé par un nombre positif ou nul d'espaces à insérer entre ces deux caractères. On doit pouvoir accéder à la valeur de la latence.
Appelons `DisplayerWithLatency` cette nouvelle classe d'afficheurs.

On voudrait pouvoir utiliser les mêmes contrôleurs que précédemment et faire en sorte que l'invocation

```
new DisplayerController(new DisplayerWithLatency(5,3)).tryIt("Abcd",8);
```

crée un afficheur avec une latence de trois espaces et produit l'affichage ci-contre.

```
|    A|
|   Ab|
|  Abc|
| Abcd|
|Abcd |
|bcd  |
|cd   |
|d   A|
```

Q 2 . Définissez la classe `DisplayerWithLatency`.

Q 3 . Quelles modifications faut-il apporter à la classe `DisplayerController` ?

Les afficheurs avec latence et vitesse paramétrable

A chaque top d'horloge, les afficheurs précédents font un seul décalage. On voudrait une nouvelle sorte d'afficheur dont on pourrait fixer le nombre de décalages effectués à chaque top. Ce nombre sera un entier positif ou nul.
Appelons `SpeedDisplayer` cette nouvelle classe d'afficheurs.
L'appel « `new DisplayerController(new SpeedDisplayer(5,3,2)).tryIt("Abcd",8);` » utilise un afficheur avec une latence de 3 et une vitesse de 2, ce qui doit produire les affichages successifs montrer ci contre.

```
|   Ab|
| Abcd|
|bcd  |
|d   A|
|  Abc|
|Abcd |
|cd   |
|   Ab|
```

Q 4 . Définissez la classe `SpeedDisplayer`.

Affichage de droite à gauche

Le succès grandissant de nos afficheurs ouvre la porte à de nouveaux marchés à l'international. Nous avons désormais des demandes pour fournir des afficheurs lumineux « simples » qui affichent et font défiler le message de gauche à droite.
Appelons `LeftRightDisplayer` ces afficheurs.
En utilisant un `DisplayerController` pour piloter un tel afficheur, on veut donc obtenir une trace telle que celle montrée ci-contre.

```
|A    |
|bA   |
|cbA  |
|dcbA |
|AdcbA|
|bAdcb|
|cbAdc|
|dcbAd|
```

Q 5 . Définissez la classe `LeftRightDisplayer`.

Hors sujet – Question plus difficile : *comment pourrait-on faire (autrement) pour pouvoir également disposer d'afficheur de gauche à droite, avec latence et avec avec vitesse paramétrable ?*
L'idée étant de ne pas avoir à redéfinir autant de nouvelles classes.