TP23 (noté !) : Algorithmique – Programmation FIP (ING39)

S. Rosmorduc

Ce Tp réutilise ce que vous avez fait dans les deux séries de Tps précédents, création de composants et MVC (surtout JTable).

Le but de ce TP est de pouvoir afficher des données numériques à la fois dans une colonne d'une JTable *et* dans un composant graphique histogramme.

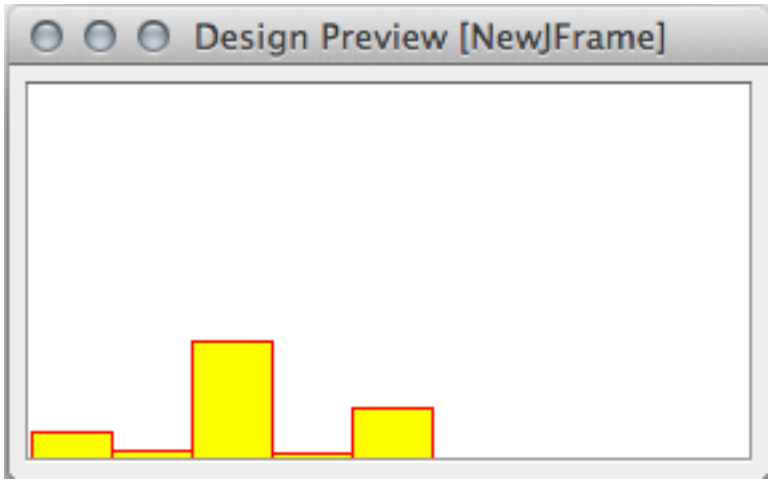
On procèdera en plusieurs étapes, en créant d'abord une version simplifiée du composant (exercice 1), puis en le modifiant pour qu'il puisse travailler facilement sur des données provenant du modèle d'une JTable (exercice 2).

Respectez l'organisation du projet qui vous est proposée, même si des classes se répètent entre la question 1 et la question 2. Cela nous permettra de noter séparément les deux questions.

Dans tout ce devoir, on supposera que les tableaux d'entiers passés en argument ne contiennent que des entiers positifs.

Exercice 1

On désire créer le composant `tp23Swing.exo1.JHistogramme1` pour afficher un histogramme comme ci-dessous :



Le composant affichera des données stockées dans un tableau d'int (voir la méthode `setDonnees`)
Créer et tester le composant graphique.

— d'afficher l'histogramme à l'échelle 1 unité = 1 pixel ;
— pour l'instant, la seule méthode qui modifie le contenu des données de l'histogramme est `setDonnees` ; on ne demande pas de programmer une architecture MVC pour cet exercice ;
— rendez actif dans l'application le bouton « changer pour une valeur aleatoire » ; il doit appeler la méthode `fixerAuHasard`.

Indication : Si on veut faire comme si le repère était orienté vers le haut, et que le point (0,0) était en bas à gauche, il faut utiliser la formule suivante pour passer d'un y calculé dans le repère mathématique vers un y dans le repère standard de Swing, orienté vers le bas :

```
int ySwing= getHeight() - yMath;
```

Ainsi, 0 dans le repère mathématique devient getHeight() dans le repère de Swing, et getHeight() dans le repère mathématique devient 0 dans le repère de Swing.

Exercice 2

Chaque question correspond à un package ; seule la classe JHistogramme2 est partagée entre tous.

Dans cette exercice, on va créer un nouveau composant histogramme, capable d'afficher les données prises dans une *colonne* d'une table. Le modèle de ce composant sera donc un `TableModel,` (plus, implicitement, un entier qui sera le numéro de la colonne dont les données sont extraites).

En toute logique, ce composant sera un peu trop limité - dans la réalité, l'histogramme aurait son propre modèle, et on utiliserait probablement le pattern *adaptateur* pour obtenir un « modèle d'histogramme prenant ses données dans une table ». Mais c'est un peu complexe pour le début de ce TP noté.

Question 1

Complétez la classe JHistogramme2 pour qu'elle affiche correctement les données de la table telle qu'elle est créée dans MainQ1.

Question 2

Dans le package Question2, complétez le code pour que cette fois-ci, votre histogramme soit correctement mis à jour quand le contenu du tableau est modifié.

Suggestion : envisagez d'implémenter un `TableModelListener` (voir la doc).

Attention, l'histogramme est placé dans un `JScrollPane`, et on désire que si les valeurs à afficher sont grandes, on puisse voir la totalité des barres, quitte à scroller avec les ascenseurs.

Question 3

On ajoute maintenant un bouton et un champ texte pour permettre d'ajouter une valeur au tableau.

Pour cette question, contrairement aux précédentes, on vous laisse modifier – si vous le jugez nécessaire – le code de Modele3.

Question 4

Écrivez dans le package `question4` un programme où
— le tableau comporte deux colonnes d'entier ;
— on affiche le tableau sous forme de `JTable` (dont les cases sont modifiables) ;
— on affiche trois histogrammes, un pour chaque colonne ;
— si on modifie le contenu d'une case, ça se répercute sur l'histogramme correspondant.

Exercice 3

Cet exercice n'est pas noté.

En réalité, au lieu d'utiliser directement `TableModel` comme modèle d'un histogramme, on aurait plutôt tendance à utiliser un modèle comme :

```
public interface HistogrammeModele {
  int getNombreDeBarres();
  int getValeur(int index);
  void ajouterHistogrammeModelListener(HistogrammeModeleListener
listener);
  void supprimerHistogrammeModelListener(HistogrammeModeleListener
listener);
}

public interface HistogrammeModeleListener {
  void histogrammeModifie();
}
```

À noter : il n'y a pas de méthode `setValeur,` dans l'interface, mais on peut imaginer que les données de l'histogramme changent à cause de causes externes.

- Créez une classe JHistogramme3 qui utilise un modèle de ce type, et une implémentation basique de ce modèle.
- Créez un *adaptateur* : comme l'interface TableModel ne peut convenir comme modèle d'un JHistogramme3, vous écrirez `TableColumnHistogrammeModel,` un `HistogrammeModele` qui a comme représentation interne un `TableColumn` et un numéro de colonne.
- Vous pourrez alors essayer de reprendre (dans le package Exercice3), la question 2 de l'exercice 2, mais avec votre nouveau modèle.