ECOLE SUPERIEURE D'INFORMATIQUE DE GESTION

ÉPREUVE REGROUPÉE

Janvier 2018

BRANCHE : **ALGORITHMIQUE ET PROGRAMMATION OBJET (APO)**

CLASSE : **ESIG2 Classe A (Groupes 1, 2 et 3)**

DATE : 17 janvier 2018

NOM : .....

PRÉNOM : .....

PROFESSEUR : Eric Batard

N° du poste de travail : ESIG-PB.....

N° de clé USB : .....

Modalités

- Durée : 240 minutes.
- Travail individuel.
- Documentation personnelle (livres, papiers) : Autorisée.
  Documentation électronique (disquettes, CD, clé USB, …) : Autorisée si elle a été recopiée dans un répertoire sur `C:\ESIGUsers` ***avant*** le début de l'épreuve.
- Tout partage de ressources de votre poste de travail avec le réseau ainsi que toute tentative de communication seront considérés comme fraude et sanctionnés comme tels par la note minimale.

Démarrage

- Connectez-vous au réseau sur le poste de travail qui vous a été attribué.
- Copiez dans `C:\ESIGUsers\APO-JAN-18` le *contenu* du dossier réseau qui vous sera indiqué au début de l'épreuve, normalement
  `G:\ESIG Distribution\2017-2018\ ESIG2\APO\Eléments ER APO 17-01-18`
- Les éléments fournis, à l'exception de cet énoncé, se trouvent dans deux répertoires (`Lotoo-JC` et `Lotoo-IJ`) qui permettent, à choix, de travailler avec JCreator ou IntelliJ IDEa. Vous rendrez le répertoire dans lequel vous avez travaillé.

Travail à faire

- Lisez tous les documents fournis.
- Complétez les procédures/fonctions/méthodes ou classes demandées de manière à ce qu'elles répondent aux spécifications de l'énoncé.
- Vous rendrez la copie comportant les réponses écrites demandées ou les fichiers correspondants.

C'est *à vous* de vérifier que le répertoire rendu contient bien
la dernière version de votre travail

Si JCreator, le nom de chaque fichier rendu modifié doit être préfixé par vos initiales (p. ex. `EB_MonFichier.java`)

Si IntelliJ IDEa, le nom du *répertoire de projet* doit être préfixé par vos initiales

----

A ajouter au plus tard juste avant la reddition

Quand les deux cas se présentent, le masculin est utilisé dans cet énoncé de façon générique.

Tirer le gros lot en APO avec Loto-O

Contexte

Nous allons jouer avec au Loto-O ou Loto-Objet.

D'emblée voici l'interface du programme à obtenir suivie de quelques explications sur la terminologie. Ces explications se limiteront à ce qui est nécessaire pour cette épreuve sachant que divers aspects seront simplifiés, voire ignorés.

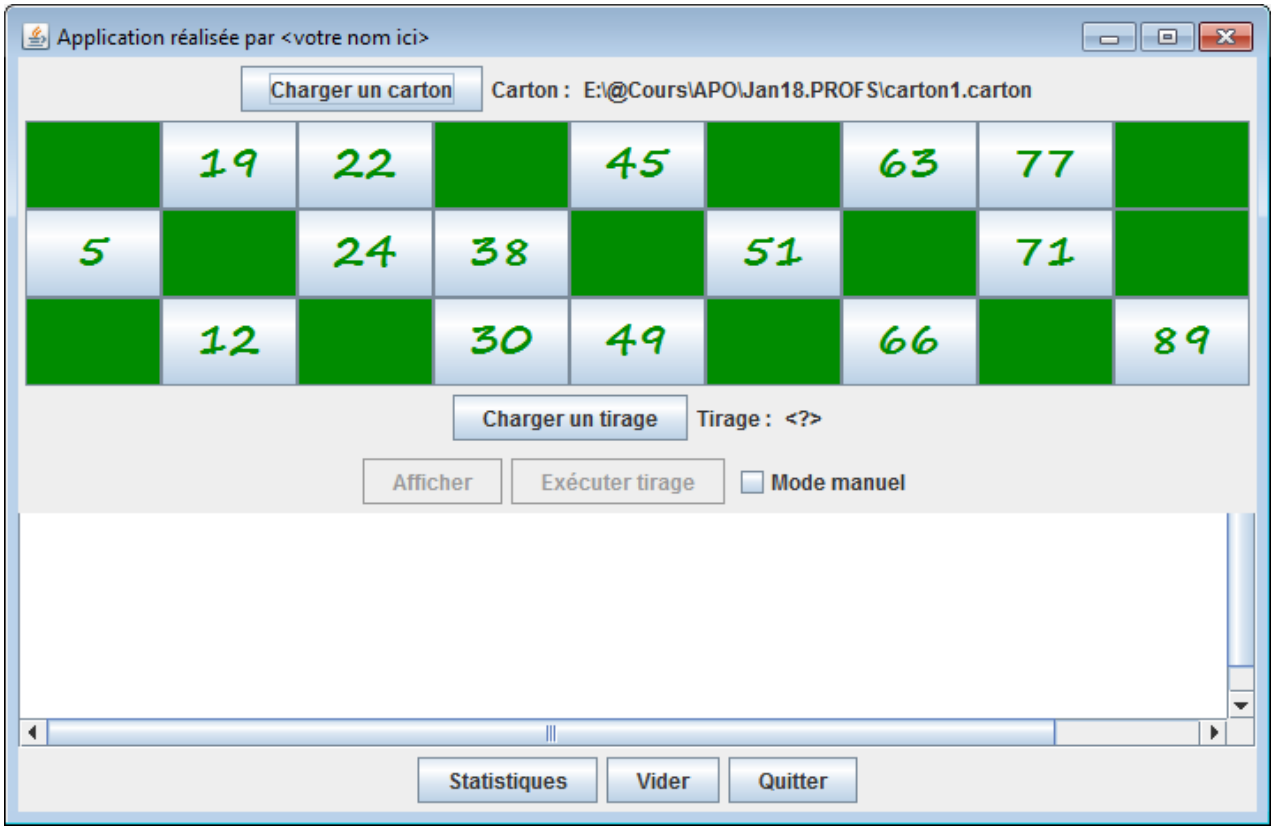

Figure 1

La zone de 3 lignes sur 9 colonnes qui contient des chiffres s'appelle un *carton*. Il y a toujours, sur un carton, exactement 5 nombres par ligne, soit 15 chiffres, valant entre 1 et 90. Ci-dessous l'image d'un vrai carton :

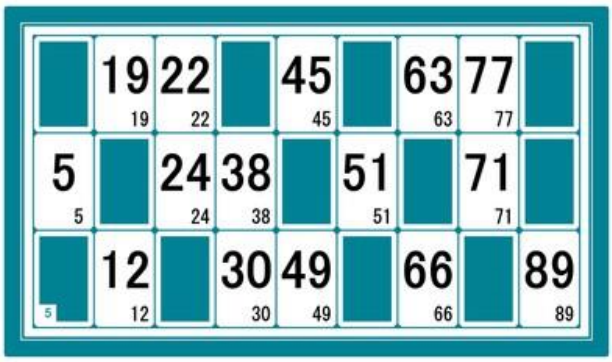

Figure 2

Nous travaillerons sur un seul carton à la fois mais dans un vrai loto, on peut distribuer un carton différent à des dizaines de joueurs.

Après la distribution des cartons, on procède au *tirage* des nombres, un par un. Chacun joueur note sur son carton si le nombre tiré apparaît sur son carton. On verra plus loin des exemples de tirage. Ce sera un changement de couleur du texte dans votre programme.

Par le jeu du hasard, il peut arriver que tous les nombres d'une même ligne soient tirés. Dans le vrai jeu, le joueur doit crier « Quine » et votre programme se contentera de l'afficher. De même si les dix nombres des deux lignes sont tirés, c'est « Double quine ». Et l'apothéose est le tirage des 15 nombres sur le carton qui donne « Carton plein ».

Dans le vrai jeu, dès qu'un joueur a obtenu le quine ou le double quine, on « remet » le carton, c'est-à-dire qu'on enlève toutes les marques du tirage précédent. Votre programme fera de même pour traiter correctement plusieurs tirages successifs.

Mise en place de l'interface

Compléter la classe fournie `LotooFen`. Pensez d'abord à mettre votre nom dans le titre de la fenêtre.

Dans cette fenêtre, il y a plusieurs zones à considérer, zones qu'il faut empiler d'une façon ou d'une autre. La gestion d'événements est détaillée dans la section suivante.

— *Zone du choix du carton :*
  — Un bouton toujours actif
  — Un label « Carton : »
  — Un autre label qui indique au démarrage « `<?>` » et qui contiendra la spécification du carton chargé.

  Les deux labels peuvent être fusionnés en un seul si vous voulez.

— *Zone de l'affichage du carton :*
  — 27 boutons disposés en 3 lignes.

  Utilisez la classe `LotooButton` qui est fournie et qui donnera instantanément la présentation souhaitée. Le texte sera initialement « `<?>` ».

— *Zone du choix du tirage :*
  — Un bouton qui ne sera actif qu'après le chargement d'un carton. Donc inactif au départ.
  — Un label « Tirage : »
  — Un autre label qui indique au démarrage « `<?>` » et qui contiendra la spécification du tirage chargé.

  Les deux labels peuvent être fusionnés en un seul si vous voulez.

— *Zone de traitement du tirage :*
  — Deux boutons qui ne seront actifs qu'après le chargement d'un tirage. Donc inactifs au départ.
  — Une case à cocher « Mode manuel », non cochée au départ et inactive au départ aussi.

— *Un* `JTextArea` pour les affichages, non modifiable et avec barres de défilement.
  Taille : 20 lignes par 40 colonnes.

— *Zone de commande :*
  — Un bouton « Statistiques » toujours actif (pour une partie faisable à part)
  — Un bouton « Vider » toujours actif (pour vider le contenu du `JTextArea`)
  — Un bouton « Quitter » toujours actif.

Voici ce que donne l'application au démarrage :

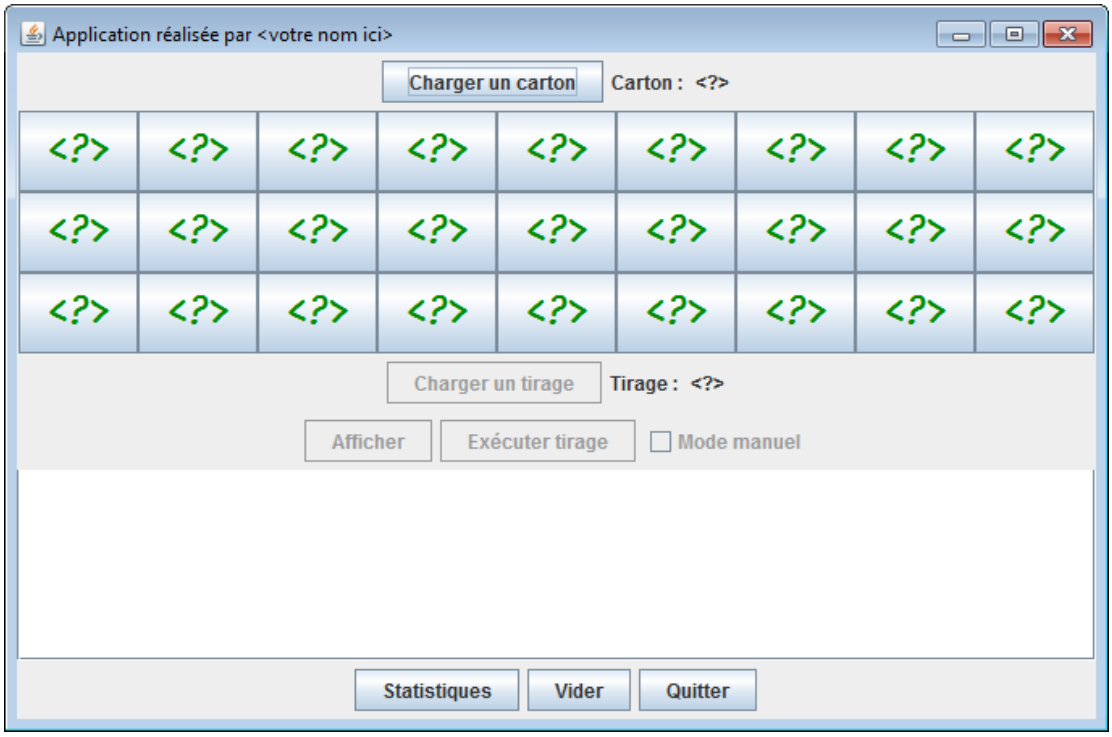

Figure 3

Classes fournies : tout ce qui est déjà fait et donc pas à faire !

Avant de détailler la gestion des événements, il faut présenter les classes mises à disposition. A une exception près (la classe `LotooButton`), vous devrez compléter ces classes en ajoutant des méthodes et des champs/propriétés/attributs. Vous pouvez aussi choisir de renommer (avec précaution !) certains éléments fournis (sauf les classes elles-mêmes). Mais à part cela il est ***fortement déconseillé*** de modifier le code fourni.

— *Classe* `LotooButton`

Cette classe est à utiliser telle quelle et vous n'aurez pas à la modifier. Cette sous-classe de `JButton` va servir à plusieurs choses (toutes ***déjà*** programmées) :

- remplacer la classe `JButton` pour l'affichage du carton,
- gérer automatiquement le changement de la valeur affichée (en vert),
- gérer automatiquement le cas de la valeur à 0 (case sans nombre, tout en vert),
- stocker la valeur associée à la case,
- stocker l'information selon laquelle la valeur a été tirée et même
- gérer automatiquement le changement de couleur quand la valeur est sortie (tirée).

C'est pourquoi la classe `Carton` (ci-dessous) utilise uniquement des instances de `LotooButton` et vous êtes encouragés à faire de même !

Les méthodes utiles sont :

- `setValeur()` et `getValeur()` qui permettent respectivement de fixer la valeur de la case et de la connaître.
- `setValSortie()` qui permettent respectivement de fixer la valeur, `true` ou `false`, de l'attribut `valSortie` qui indique si le nombre de la case est sorti (= a été tiré) la case et `isValSortie()` de savoir si c'est le cas ou pas.
- `setNumLigne()` et `getNumLigne()` qui permettent respectivement d'indiquer dans quelle ligne se trouve la valeur de la case et de connaître de ce numéro de ligne.

— *Classe* `Carton`

Cette classe devra être complétée pour répondre à diverses questions. Cette classe définit plusieurs informations :

- 2 constantes donnant le nombre de lignes (`NB_LIGNES`) et de colonnes (`NB_COLONNES`),
- une constante `NB_COLONNES_A_REMPLIR` qui donne le nombre de cases qui doivent être remplies (= valeurs tirées), c'est-à-dire 5, pour qu'une ligne soit remplie (sert pour le calcul des quines),
- un tableau à deux dimensions de `LotooButton` appelé `listeCases`, qui servira pour l'affichage du carton, cf. ci-dessous petit rappel sur les tableaux à deux dimensions.
- Et trois méthodes qui permettent de passer d'une vision linéaire (nombres de 0 à 26) à des lignes et colonnes d'un tableau 3x9 à deux dimensions, et inversement. Normalement vous n'en avez pas besoin.

— *Classe* `LotooMain`

Inutile d'insister sur le fait qu'il faudra compléter la méthode `main()` de cette classe !

— *Classe* `LotooFen`

Cette classe sera la classe de fenêtre. Elle contient deux constantes :

- le répertoire de départ, à ajuster selon votre choix d'environnement de développement, et à utiliser par la suite.
- la police utilisée pour le `JTextArea` (appel de la méthode `setFont(POLICE_JTA)`) mais c'est facultatif.

Et deux méthodes plus pratiques à utiliser :

- `afficher()` qui prend deux paramètres : une chaine et un `JTextArea` et qui affiche sur une ligne la chaine dans le `JTextArea`.
- `findExtension()` qui prend en paramètre un `File` et qui renvoie sous la forme d'une chaîne l'extension de ce fichier.

Rappel sur les tableaux à deux dimensions

Le tableau est déjà déclaré. Vous n'aurez donc besoin que de :

— accéder à un élément du tableau : la syntaxe est `listeCases[numLigne][numColonne]`
Notez la double paire de crochets (pas de virgule !).
Ne perdez pas de vue que pour accéder à un attribut (non statique) d'une classe en dehors de cette classe, il faut mettre un nom d'instance avant. Ici ce sera une instance de la classe `Carton`. Et n'oubliez pas que ce tableau contient des éléments qui sont, chacun, des instances de `LotooButton`.
Mais tout cela dépend de comment vous organisez votre programme.

— parcourir tous les éléments du tableau : une bonne double boucle imbriquée avec une boucle sur les lignes et l'autre sur les colonnes fera l'affaire ! Utilisez les constantes pour les bornes.

Gestion d'événements et cinématique de l'interface

— ***Avant le démarrage (cf. Figure 3 ci-dessus)***

Les boutons de votre `listeCases` devront être instanciés avec `<?>`.

— ***Le mode manuel (accessible après chargement)***

Si la case à cocher est cochée, un clic sur un `LotooButton` du carton doit provoquer le changement d'état de l'attribut `valSortie` : le texte passe en rouge. Cf. plus loin pour le visuel.

— ***Chargement d'un carton (cf. Figure 1 au début) :***

Dans cette épreuve, un carton prend la forme d'un simple fichier texte, dont l'extension est `.carton`. Les données sont dans l'ordre des lignes, comme les cartons, et 0 signifie pas de chiffre dans cette case. On supposera que les fichiers contiennent toujours exactement 27 données correctes.

La classe fournie `LotooButton` comporte ce qu'il faut pour afficher joliment le carton.

Ce qui est imposé

- Après clic sur le bouton, ouverture d'un `JFileChooser` qui permet de choisir un fichier dans le répertoire de départ
- Lecture du fichier texte correspondant
- Affichage des valeurs dans les cases correspondantes
- Affichage de la spécification du fichier et du carton correspondant
- Activation du bouton pour les tirages et de la case à cocher.

Ce qui est proposé

- Pendant la lecture, comme les boutons sont déjà instanciés et posées sur l'interface, il suffira d'affecter le numéro de ligne, qui est stocké dans chaque `LotooButton` et utiliser `setText()` pour affecter la valeur. L'affichage viendra alors automatiquement.

— ***Chargement d'un tirage :***

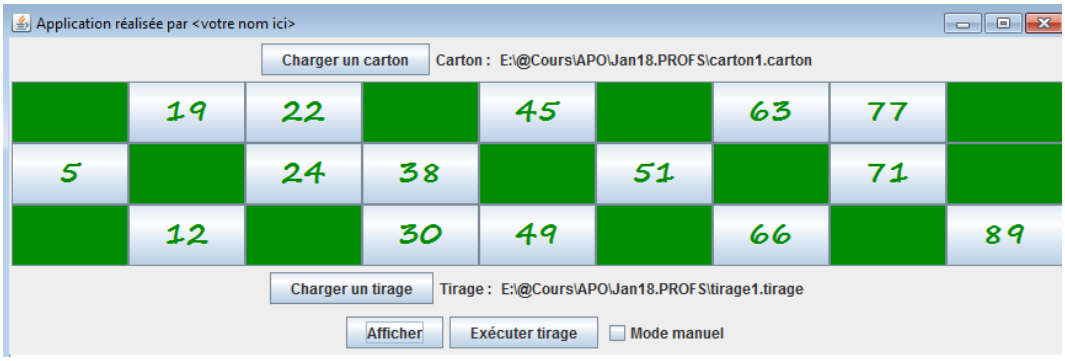

Figure 4

Dans cette épreuve, un tirage prend (aussi) la forme d'un simple fichier texte, dont l'extension est `.tirage`. Il faut considérer que les données sont en nombre quelconque et il faut les stocker dans une structure de données adéquate.

On supposera que les fichiers contiennent toujours des données correctes.

Ce qui est imposé

- Après clic sur le bouton, ouverture d'un `JFileChooser` qui permet de choisir un fichier dans le répertoire de départ
- Lecture du fichier texte correspondant
- Stockage (sans affichage) des valeurs
- Affichage de la spécification du fichier et du carton correspondant
- Activation des boutons d'affichage et d'exécution.
- Remise à l'état « non tirée » des cases du carton (cf. plus bas).

— ***Affichage d'un tirage :***

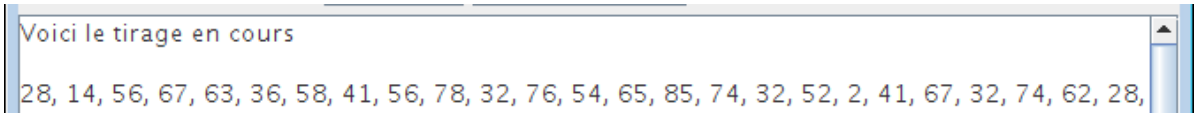

Figure 5

La liste des données apparaît dans la zone d'affichage centrale.

— ***Exécuter un tirage :***

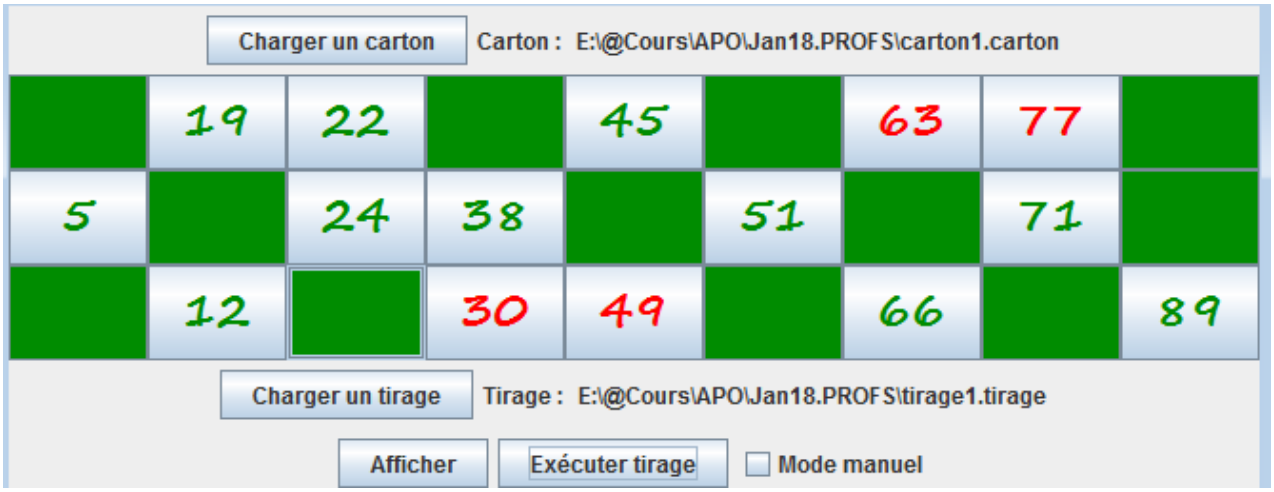

Figure 6

L'exécution d'un tirage revient à parcourir tous les nombres d'un tirage et regarder s'ils se trouvent dans le `listeCases` du carton en cours.

Si le nombre est trouvé, il faut changer le statut (`setValSortie(true)`) du `LotooButton` qui le contient, ce qui aura pour effet de changer sa couleur automatiquement.

Il faut donc établir un lien entre le nombre et la case qui le contient. Pour cela, aucune méthode n'est imposée. On peut utiliser une boucle (imbriquée) de recherche ou préparer le travail avec un `HashMap` (par exemple).

Une fois tous les nombres du tirage traités, il faudra vérifier si on a une seule ligne remplie, deux lignes remplies ou les 3 et afficher le message adéquat (Quine, Double quine ou Carton plein) et pas de message s'il n'y a aucune ligne remplie !

Exemple de quine :

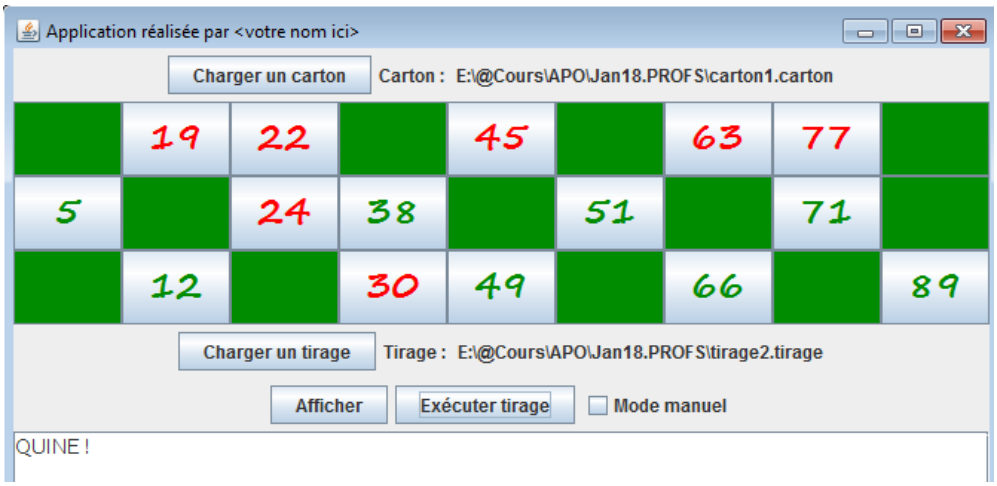

Figure 7

Recommandation/suggestion : il sera peut-être pratique de définir un tableau qui contient le nombre de valeurs tirées par ligne et une fonction qui compte combien de lignes sont remplies à partir de ce tableau.

— ***Vider et Quitter :***

Respectivement vide la zone d'affichage centrale et quitte proprement l'application.

— ***Statistiques : TOTALEMENT INDEPENDANT DES CARTONS ET TIRAGES***

Si vous n'arrivez pas à faire l'interface graphique, les résultats peuvent être affichés par `System.out.println()`

Un répertoire `Data` est fourni avec divers fichiers (vides) et répertoires

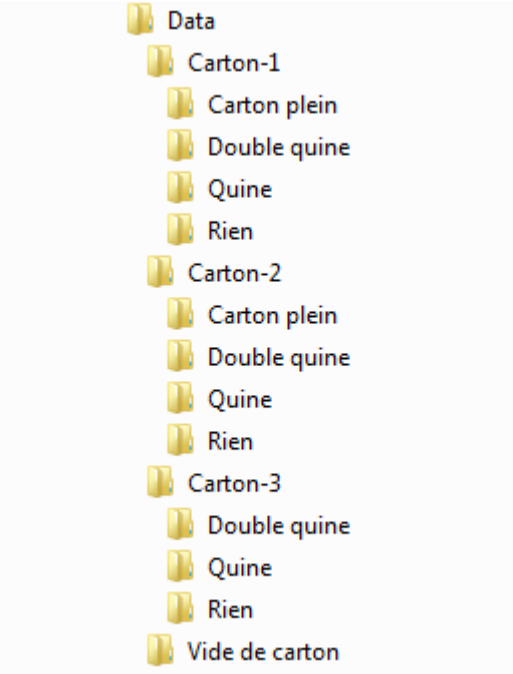

Figure 8

L'objectif de cette question est d'obtenir les informations suivantes :

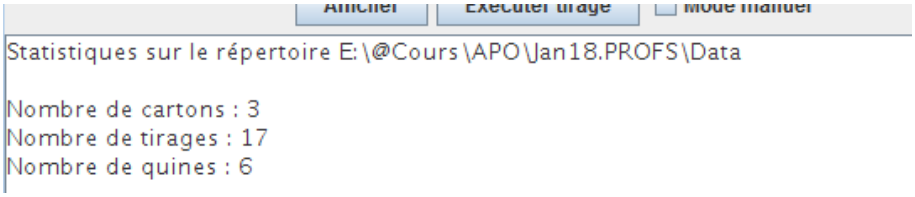

Figure 9

— Le nombre de cartons est le nombre de fichiers dont l'extension est `carton`.
— Le nombre de tirages est le nombre de fichiers dont l'extension est `tirage`.
— Le nombre de quines est le nombre total de fichiers qui se trouvent dans des répertoires dont le nom est `Quine`.

Il est demandé 3 fonctions distinctes (c'est plus simple).

Pour répondre à chacune de ces questions, faites un parcours récursif de `Data` et compter les fichiers ayant les caractéristiques voulues.