LABORATOIRE 13 | *Informatique 1*
ECRITURE ET LECTURE DE STREAMS

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Buts du laboratoire (3 périodes)

1. Le but principal de ce laboratoire est de comprendre et d'appliquer les concepts de flux de données (en anglais, *streams*) dans une application. Ainsi, nous pourrons observer qu'un programme peut facilement interagir avec des fichiers externes en les écrivant ou en les lisant. Nous illustrerons ces lectures et écritures notamment en créant des images dans des formats courants utilisables par d'autres programmes. Votre programme ne sera ainsi plus seul mais interagira avec le "monde extérieur" !
2. Dans la première partie du laboratoire, vous allez écrire du code qui va générer des factures sous forme de fichier texte. La partie génération de facture est déjà totalement réalisée et votre tâche sera de mettre le texte de la facture dans un fichier. Vous apprendrez également comment gérer les erreurs de manière élégante grâce au mécanisme des *exceptions*.
3. Dans la deuxième partie, vous allez voir comment il est possible de lire dans un fichier en créant une méthode qui va lire dans un fichier texte des valeurs qui correspondent à des lignes que vous pourrez ensuite dessiner à l'écran. Une fois le dessin effectué à l'écran, vous aller sauvegarder celui-ci dans un format SVG (*Scalable Vector Graphics*) affichable dans *Firefox* par exemple et, pour la partie optionnelle de ce labo, dans le format BMP (*Bitmap*) standard sous *Windows*™.
4. La durée estimée pour réaliser ce laboratoire est de **trois périodes**. Si le temps imparti ne devait pas suffire, vous êtes invités à terminer le travail en dehors des heures de cours.
5. Vous pouvez trouver cette donnée sous forme électronique sur http://inf1.begincoding.net dans la rubrique laboratoires. Vous y trouverez également le corrigé de ce labo la semaine prochaine.

Partie 1 – Génération de facture en mode texte

Nous vous fournissons une classe `GarageManager` qui est une application permettant de gérer différents types de travaux sur des voitures et de créer des factures à l'écran. Toutefois, cette application est incomplète car le garagiste désire pouvoir créer un fichier texte correspondant à la facture afin de l'imprimer. Votre tâche sera donc de prendre ce qui est affiché à l'écran et de créer un fichier texte contenant la même information.

1) Démarrez *Eclipse* et créez un nouveau projet nommé *Lab13.*
2) Téléchargez les données du laboratoire depuis le site web du cours et copier les fichiers source dans le répertoire *src* du projet que vous venez de créer.
3) Dans la classe `GarageManager`, observez la méthode `generateBill(…)` et comprenez comment **l'utiliser** uniquement (son fonctionnement est assez compliqué). Cette méthode retourne un `String` correspondant à la facture.
4) Vous devez maintenant sauvegarder dans un fichier, que vous nommerez `bill.txt`, ce que la méthode `generateBill()` vous retourne. Pour ouvrir un fichier et écrire à l'intérieur vous devez vous référer à vos notes prises lors des explications des classes `PrintWriter` et `FileOutputStream`. N'oubliez pas le block `try` / `catch` pour gérer les exceptions et de refermer le fichier après y avoir écrit.
5) Vérifiez que le fichier généré est bien correct en l'ouvrant à l'aide d'un éditeur de texte comme par exemple *Notepad* (sous Windows).

Partie 2 – Fichiers graphiques

Après avoir exercé les écritures dans les fichiers, dans cette partie nous allons lire (on dit *parser* en anglais) un fichier texte qui contient des coordonnées correspondant à des lignes à dessiner dans une certaine couleur.

Le format du fichier CSV est le suivant :

```
  (1)     (2)       (3)
100;100;200;100;250;000;000
300;100;250;100;000;100;200
500;200;200;200;100;255;000
050;200;100;100;100;110;100
```

Les éléments sont, dans l'ordre, comme suit :

(1) Les coordonnées x et y du premier point

(2) Les coordonnées x et y du deuxième point
(3) Les trois valeurs des composantes rouge, vert et bleu de la couleur (Valeurs entre 0 et 255)

Tâche 1 – Lecture du fichier CSV

1) Copier le fichier *drawingTest.csv* dans le répertoire de votre projet. Il contient un exemple de dessin CSV valide pour votre application.
2) Implémentez la partie manquante de la méthode `parseFile` de la classe `ReadFileApplication`. Cette méthode devra faire les choses suivantes :
   - Lire chaque ligne du fichier et, pour chaque ligne :
     - Séparer la ligne en fonction des ';'. Vous pouvez utiliser la méthode `split` de la classe `String`.
     - Stocker les valeurs numériques lues (utilisez la méthode `Integer.parseInt(String s)` pour tenter de convertir un `String` en `int`).
     - Créer une nouvelle instance de `Line` avec ces valeurs.
     - Ajouter cette instance dans un `Vector` qui sera retourné à la fin de la méthode et qui sera dessinée par la classe `ReadFileApplication`.

   L'ouverture et la fermeture du fichier sont déjà gérées par le constructeur et la fonction `close` de la classe `CSVReader`.
3) Vérifier en utilisant le *debugger* que le fichier est correctement lu et que le `Vector` de `Line` est correctement créé.
4) Vérifier aussi que les lignes soient correctement affichées.

Tâche 2 – Enregistrement de la figure au format SVG

Le format SVG

Le format SVG est un format de graphique portable pour le Web[1]. Ce format vectoriel est reconnu par tous les browsers excepté *Internet Explorer*™. La spécificité d'un format vectoriel permet de stocker des images sous formes de primitives graphiques (comme des lignes, des courbes etc…) ce qui permet d'avoir un meilleur rendu et une plus faible taille de fichier car les graphiques peuvent être redimensionnés sans problème (d'où le nom *Scalable Vector Graphics* (SVG)[2]). Un format bitmap quant à lui stocke une image sous forme de pixels (points) comme illustré sur l'image ci-dessous :

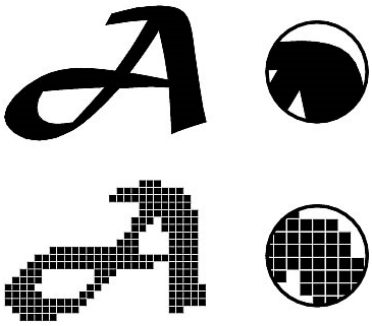

Figure 1 : Tiré de thephotoshopbible.blogspot.com

Durant cette partie, nous allons convertir le dessin affiché par `SimpleGraphics` en un fichier SVG afin de pouvoir le voir dans un browser web. Le graphique que nous avons créé dans la tâche 1 ne contenant que des lignes, nous allons devoir exprimer les lignes dans un fichier SVG[3]. En plus de cette description des lignes, un fichier SVG complet contient également un entête particulier (*header*) ainsi qu'une fin de fichier spéciale (*footer*) mais qui sont déjà implémentés dans la classe `SVGWriter`. Afin d'être dessinée correctement, une ligne se présente sous la forme suivante dans un fichier SVG :

```
<line x1="100" y1="100" x2="300" y2="200" stroke="rgb(250,0,0)"/>
```

[1] Ce format est basé sur XML et développé par le W3C (le comité de normalisation du Web).

[2] http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[3] http://www.w3.org/TR/SVG/shapes.html#LineElement

Les valeurs `x1`, `x2`, `y1` et `y2` correspondent aux positions des extrémités de la ligne. Les valeurs qui suivent le terme `rgb` correspondent aux composantes rouges, vertes et bleues de la couleur de la ligne.

1) Implémentez la méthode `writeLine` de la classe `SVGWriter`. Vous pouvez vous inspirer de la fonction `writeHeader` pour observer comment la classe `PrintStream` fonctionne. Vous pouvez aussi vous référer à l'API *Java*[4] qui reste toujours une source de très bonne qualité (alternativement, vous pouvez pressez sur la touche F2 dans *Eclipse* lorsque votre souris est sur une classe pour afficher sa documentation).
2) Le fichier SVG créé possède par défaut le même nom que le fichier CSV et est enregistré dans le même répertoire que celui-ci. Vous pouvez cependant rajouter un deuxième argument à l'exécution (comme dans la tâche 1 afin de spécifier le nom et l'emplacement du fichier à sauvegarder).
3) Vérifier que le fichier SVG est correctement créé en l'ouvrant avec *Firefox*.
4) Trouver d'autres formes dans la norme SVG et essayez de faire un dessin avec (cercles, …)

Tâche 3 – Écriture facture au format HTML

Dans cette tâche on vous demande de rajouter une nouvelle méthode dans la classe `GarageManager` ajoutant la possibilité de faire des factures sous forme de fichier HTML (c'est-à-dire un fichier affichable par un navigateur Web). Le principe des fichiers HTML est simple : un fichier HTML commence par <HTML> et se termine par </HTML>. Des tags (indiqués par les caractères <>) permettent d'indiquer la mise en page. Notez qu'il faut ouvrir les tags et les refermer (à l'aide du caractère '/'), ceci afin d'indiquer la fin de la validité du tag.

Ainsi si l'on écrit dans un fichier HTML : `<html><b> Hello </b></html>`

Le texte **Hello** sera affiché en gras lorsqu'il sera affiché dans un navigateur web. Voici une petite liste des tags existants et que vous pouvez utiliser directement partout où de l'HTML est utilisable :

Tag	Name	Example	Output
****	Bold	Hello	**Hello**
** **	Center	Hello	Hello
****	Emphasis	This is an ****Example****	This is an *Example*
** **	Line break	Hello World	Hello World

Vous trouverez sur Internet des listes plus complètes, par exemple pour réaliser des listes numérotées etc… (par exemple http://www.web-source.net/html_codes_chart.htm).

À l'aide de ces informations, réalisez une facture élégante utilisant les tags ci-dessus.

Partie 3 – Exercice complémentaire

Tâche 4 – Enregistrement de la figure au format BMP (*optionnel*)

Le format BMP est un format binaire : contrairement à ce que nous avons fait jusqu'à maintenant, le contenu du fichier n'est pas sous forme ASCII ou UTF-8. Il n'est donc pas possible de lire ce fichier avec un éditeur de texte.

Le format BMP est ce que l'on nomme un *bitmap* (c'est-à-dire une carte de bits) qui est principalement utilisé sous *Windows™* car il a été créé à l'origine par *Microsoft©*. A l'inverse des formats vectoriels, il est plutôt recommandé pour les photos car ce format correspond en fait à un grand tableau dans lequel la couleur de chaque pixel est décrite. Ce format d'image est en général non-compressé et prend donc beaucoup plus de place qu'un JPEG (qui est un format utilisant des méthodes de compression afin de réduire la taille de l'image aux dépens de la qualité de celle-ci).

Durant cette partie, nous allons écrire la valeur des pixels de `SimpleGraphics` dans un fichier BMP. Le format de fichier d'image BMP est très simple. Il contient tout d'abord un *header* qui est décrit ici[5]. Ce *header* est ensuite suivi par

[4] http://java.sun.com/j2se/1.5.0/docs/api/java/io/PrintStream.html

[5] http://www.fastgraph.com/help/bmp_header_format.html

les données brutes sans aucun formatage particulier (1 byte pour le rouge, 1 byte pour le vert et 1 byte pour le bleu). Il y a cependant des particularités. La première est que les valeurs dans le header sont stockées en format *Little-endian*[6] ce qui a pour conséquence que les bytes sont stockés dans l'ordre inversé.

La deuxième particularité est le *padding* (la bourre ou remplissage en français). Chaque ligne de données doit ainsi contenir un nombre de bytes qui est un multiple de 4. Si par exemple l'image est en couleur avec 8 bits (1 byte) pour chacune des couleurs et qu'une ligne de l'image contient 30 pixels, une ligne correspondra donc à 90 bytes. 90 n'étant pas un multiple de 4, il faut donc rajouter 2 bytes vides à la fin de la ligne pour respecter le *padding*.

1) Implémentez la fonction `writeHeader` qui écrit le header du fichier BMP. Vous pouvez utiliser les fonctions `convertToLED()` pour convertir les nombres en *Little-Endian* avant de les écrire.
2) Implémentez les méthodes `writePixel` et `writeImage` qui écrit les valeurs des pixels dans le fichier. Vous pouvez vous inspirer de la méthode `fillImage`.
3) Testez les fichiers générés en les ouvrant directement à l'aide d'un outil standard comme par exemple GIMP.

[6] http://en.wikipedia.org/wiki/Endianness