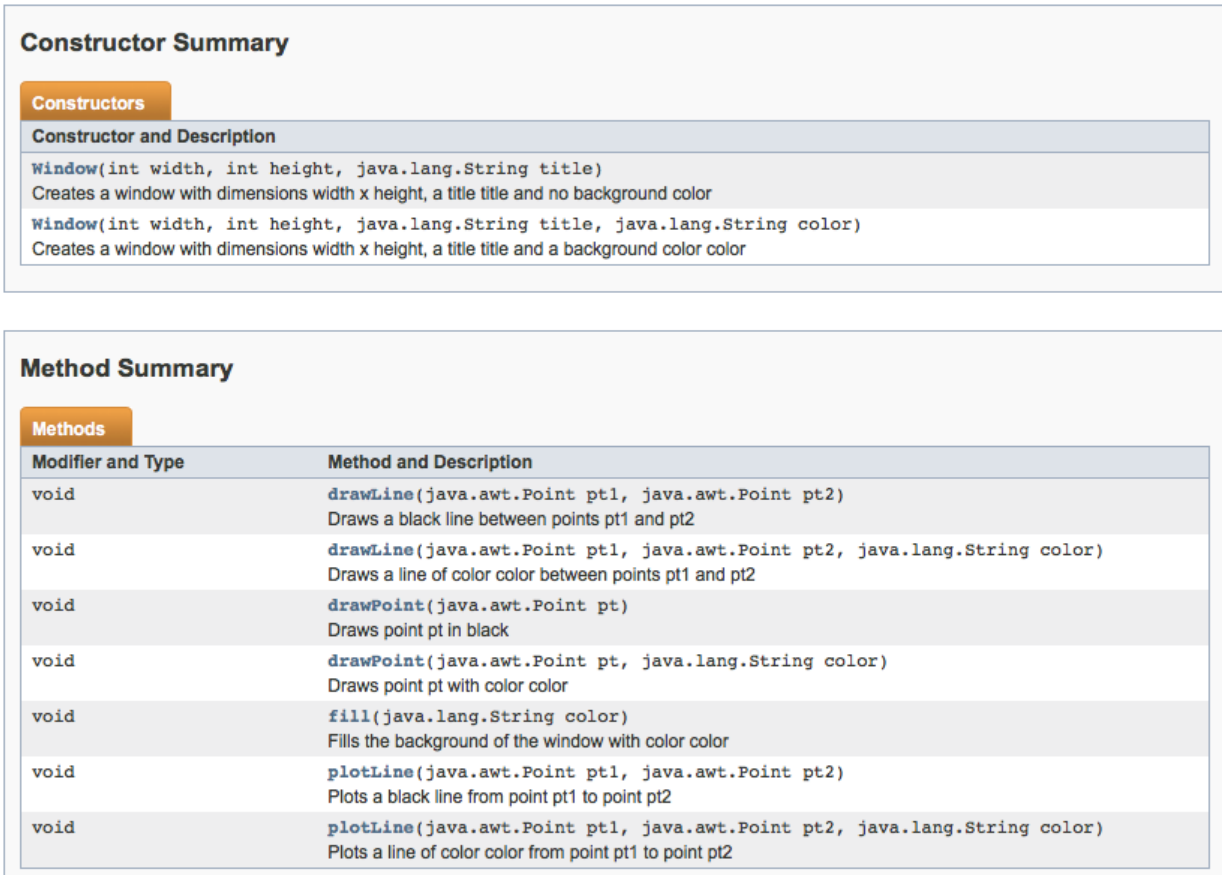
# Thème 2 - TME

## Exercices

Lors de chaque TME, vous devrez créer un répertoire pour chaque exercice, y mettre les fichiers s'y rapportant et soumettre ce répertoire à la fin du TME.

## Exercice 2 – Créer des threads, visualiser le parallélisme

Les applications graphiques permettent de mettre en évidence l'exécution concurrente de tâches. Pour ces applications, nous utiliserons un paquetage `graphic` minimal basé sur Swing, qui permet d'afficher dans une fenêtre des points et des lignes. Les caractéristiques de la classe `Window` du paquetage `graphic` sont données dans la Figure 2.



**Constructor Summary**

Constructors

| Constructor and Description |
|---|
| `Window(int width, int height, java.lang.String title)`<br>Creates a window with dimensions width x height, a title title and no background color |
| `Window(int width, int height, java.lang.String title, java.lang.String color)`<br>Creates a window with dimensions width x height, a title title and a background color color |

**Method Summary**

Methods

| Modifier and Type | Method and Description |
|---|---|
| void | `drawLine(java.awt.Point pt1, java.awt.Point pt2)`<br>Draws a black line between points pt1 and pt2 |
| void | `drawLine(java.awt.Point pt1, java.awt.Point pt2, java.lang.String color)`<br>Draws a line of color color between points pt1 and pt2 |
| void | `drawPoint(java.awt.Point pt)`<br>Draws point pt in black |
| void | `drawPoint(java.awt.Point pt, java.lang.String color)`<br>Draws point pt with color color |
| void | `fill(java.lang.String color)`<br>Fills the background of the window with color color |
| void | `plotLine(java.awt.Point pt1, java.awt.Point pt2)`<br>Plots a black line from point pt1 to point pt2 |
| void | `plotLine(java.awt.Point pt1, java.awt.Point pt2, java.lang.String color)`<br>Plots a line of color color from point pt1 to point pt2 |

FIGURE 2 – La classe `Window`

Le paquetage `graphic` est localisé dans le répertoire ci-dessous :

```
/Infos/lmd/2018/licence/ue/3I001-2018oct/TME
```

Vous devez ajouter ce répertoire à la variable `CLASSPATH`. Pour cela, ajoutez les commandes ci-dessous au fichier `.bashrc` qui se trouve dans votre répertoire de connexion (créez-le s'il n'existe pas) :

```
CLASSPATH=$CLASSPATH:/Infos/lmd/2018/licence/ue/3I001-2018oct/TME
export CLASSPATH
```

Il faut ensuite exécuter les commandes de `.bashrc` à partir d'un terminal :

```
source .bashrc
```

**Question 1**
Pour vous familiariser avec la bibliothèque graphique, écrivez un programme qui ouvre une fenêtre graphique et trace dedans un triangle. Vous tracerez les côtés du triangle en appelant la méthode `plotLine`.

Nous souhaitons maintenant paralléliser l'exécution du dessin en faisant tracer chaque côté du triangle par un `thread` différent. Dans un premier temps, nous construisons une solution basée sur une sous-classe de `Thread`.

**Question 2**
Créez une classe `DessineLigne` qui est une sous-classe de `Thread`. Une instance de cette classe doit permettre de tracer une ligne entre deux points dont les coordonnées sont des variables d'instance.

Écrivez une fonction `main` qui réalise le tracé du triangle en créant un `thread` pour dessiner chacun des côtés. Vérifiez que les différents tracés s'exécutent bien de manière parallèle.

L'autre manière de créer un `thread` est de passer au constructeur de la classe `Thread` un objet qui implémente l'interface `Runnable`.

**Question 3**
Proposez une nouvelle réalisation du tracé du triangle dans laquelle le tracé d'un côté est exécuté par un objet appartenant à la classe `DessineLigne` qui implémente l'interface `Runnable`.

## Exercice 3 – Produit matriciel parallèle

Nous souhaitons compléter notre classe `MatriceEntiere` de manière à pouvoir paralléliser le produit de matrices. Dans un produit de matrices, chaque élément de la matrice résultat est calculé en effectuant le produit d'une ligne de la première matrice par une colonne de la seconde matrice. Les deux matrices de données ne sont accédées qu'en lecture, il n'y a donc pas de risque d'interférence entre les différents calculs et on peut faire calculer chaque élément de la matrice résultat par un `thread` différent.

**Question 1**
Complétez la classe `MatriceEntiere` en lui ajoutant les méthodes suivantes :

- deux accesseurs `getNbLignes()` et `getNbColonnes()`, qui renvoient le nombre de lignes (resp. de colonnes) de la matrice ;
- une méthode *statique*
  `produitLigneColonne(MatriceEntiere m1, int i, MatriceEntiere m2, int j)`
  qui renvoie l'*entier* résultat du produit de la ligne `i` de `m1` par la colonne `j` de `m2`. Cette méthode devra lever une exception `TaillesNonConcordantesException` si les dimensions des deux matrices ne permettent pas de réaliser l'opération.

**Question 2**
Écrivez une classe `CalculElem` qui implémente l'interface `Runnable` (ou qui est définie comme une sous-classe de `Thread`). La méthode `run()` de cette classe doit écrire en position `(i, j)` d'une matrice `m` le résultat du produit de la ligne `i` d'une matrice `m1` par la colonne `j` d'une matrice `m2`.

**Question 3**
Écrivez une classe `TestProduitParallele` qui calcule une matrice `m`, résultat du produit d'une matrice `m1` par une matrice `m2`, en utilisant un thread différent pour calculer chacun des éléments de `m`.