UPMC SORBONNE UNIVERSITÉS

UFR d'Ingénierie

8 septembre 2015

Licence d'Informatique (2015-2016)

Programmation Concurrente (3I001)

TD/TME

Thèmes 1 à 5

Thème 1

Objectifs

- Retour sur les exceptions
- Retour sur les entrées/sorties

Exercices TD

Exercice 1 – Lecture de données dans un fichier ASCII

Nous souhaitons écrire une méthode **`public void`** `initFromFile(String fichier)` permettant de lire des valeurs entières dans un fichier ASCII, dont le contenu se présente sous la forme suivante :

```
5
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
```

Les deux premières lignes donnent respectivement le nombre de lignes de données dans le fichier et le nombre de valeurs qu'elles contiennent.

Voici les premières instructions que nous utilisons dans la méthode de lecture :

```
BufferedReader in = null ;
in = new BufferedReader (
         new FileReader ( new File(fichier) ));
```

Question 1
Expliquez les différents éléments de l'initialisation de la variable `in`.

Question 2
Comment s'effectue la fermeture de ce flux ?

Question 3
Nous souhaitons stocker dans un tableau l'ensemble des valeurs lues sur une ligne. Nous disposons pour cela des méthodes :

- `String readline()` dans la classe `BufferedReader`, qui lit une ligne de texte ;
- `String[] split(String regex)` dans la classe `String`, qui découpe la chaîne en utilisant *regex* comme séparateur.

Écrivez le code qui permet de stocker dans un tableau `vals` les données contenues dans la première ligne du fichier.

Question 4
Que manque-t-il pour pouvoir utiliser les données stockées dans le tableau afin de calculer, par exemple, une somme ?

Question 5
De quelle autre solution dispose-t-on pour lire les données du fichier comme une suite d'entiers ?

Exercice 2 – Division par zéro

Nous allons voir comment utiliser les exceptions pour gérer l'erreur provoquée par une division par zéro. Le principe de la gestion des exceptions en Java consiste à repérer un morceau de code (par exemple, une division) qui pourrait générer une exception (lorsque le diviseur est nul), à capturer l'exception correspondante et enfin à la traiter, c'est-à-dire déterminer ce qu'il convient de faire lorsque cette situation se produit (terminer le programme, afficher un message personnalisé et continuer l'exécution, ...).

Question 1

Que se passe-t-il si une division par zéro est effectuée par un programme Java ? Quelle utilité pourrait-il y avoir à rattraper une exception particulière (par exemple `java.lang.ArithmeticException`) dans ce cas ?

Question 2

Que doit-on ajouter au code pour empêcher l'arrêt de l'exécution du programme suite à la levée d'une exception ?

Question 3

Que faut-il faire si nous voulons obligatoirement effectuer une action, qu'une exception soit levée ou non (par exemple fermer systématiquement un fichier, clore une connexion à une base de données ou un socket, ...) ?

Question 4

Pour effectuer la division, nous disposons d'une classe `Arithmetique` qui implémente une méthode `division` prenant deux entiers en paramètres et retournant le résultat de la division entière. Cette méthode est appelée par la méthode `main` de notre classe principale ou par un autre objet. Qui doit lever l'exception et qui doit en assurer le traitement ?

Exercice 3 – Exceptions personnalisées - Gestion de comptes bancaires

Question 1

En considérant le programme suivant, nous voulons mettre en œuvre une exception afin d'interdire l'instanciation d'un objet `Compte` présentant un solde négatif. Comment peut-on procéder ?

```
public class Compte {
    private int numero = 0;
    private float solde;

    public Compte(float s){
        solde = s;
    }
}
```

Question 2

Où doit-on lever explicitement l'exception `SoldeNegatifException` ?

Question 3

Nous essayons maintenant de créer un compte avec un solde négatif à partir de la méthode `main()` de la classe principale. Que doit-on ajouter au code de création de ce compte dans la méthode `main()` ?

Exercices TME

Les indispensables

Le site de l'UE :
`https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2015/ue/3I001-2015oct/`

L'API Java 8 :
`http://docs.oracle.com/javase/8/docs/api/index.html`

La procédure de soumission

Pour chaque exercice, vous devrez télécharger un répertoire à partir du site de l'UE, compléter son contenu et le soumettre à la fin du TME. Sur le site de l'UE, sélectionnez la rubrique "Soumission des exercices". Le formulaire de la Figure 1 s'affiche.

FIGURE 1 – Formulaire de soumission

Le chemin que vous donnez est le chemin *absolu* d'accès au répertoire *sur les machines de la PPTI* : vous pouvez effectuer une soumission depuis chez vous, mais vous devez avoir recopié au préalable le répertoire à soumettre sur votre compte à la PPTI.

Vérifiez systématiquement les informations données dans le formulaire de réponse à votre soumission, en particulier la liste des fichiers chargés lors de la soumission. Cette liste doit inclure tous vos codes source.

Exercice 4 – La classe `MatriceEntiere`

Le but de cet exercice est d'écrire une classe `MatriceEntiere` offrant les fonctionnalités suivantes :
- la possibilité d'initialiser les éléments de la matrice à partir de données contenues dans un fichier ASCII ;
- la possibilité d'initialiser une matrice à zéro ;
- les opérations de base : somme de matrices, produit de matrices, produit d'une matrice par un scalaire ;
- le calcul de la matrice transposée ;
- une méthode `toString()` et une méthode d'affichage ;

L'ensemble des exceptions potentiellement levées par ces différentes opérations devra être géré soit en interne par la méthode qui lève l'exception, soit par le programme appelant, en choisissant la solution la mieux adaptée.

Question 1

Les attributs d'une matrice entière sont le nombre de lignes, le nombre de colonnes et un tableau à deux dimensions permettant de stocker ses éléments.
Construisez une classe `MatriceEntiere` avec :
- un constructeur `MatriceEntiere(int lignes, int colonnes)`
- un accesseur `int getElem(int i, int j)` qui renvoie la valeur de l'élément en ligne `i` et colonne `j`.
- un modificateur `void setElem(int i, int j, int val)` qui place la valeur `val` à l'intersection de la ligne `i` et de la colonne `j`.

Pour pouvoir tester notre classe, nous avons besoin de saisir les valeurs contenues dans une matrice. Cette opération étant fastidieuse, nous allons lire ces valeurs dans un fichier ASCII.

Question 2

Recherchez dans l'API Java la méthode permettant de transfomer une chaîne représentant un entier en la valeur correspondante.

Question 3

Écrivez une méthode *statique* **void** `initMatrixFromFile (MatriceEntiere m, String fichier)` permettant d'initialiser les éléments de la matrice à partir des données contenues dans le fichier ASCII `fichier`.

La première ligne du fichier donne le nombre de lignes de données contenues dans le fichier. La deuxième ligne du fichier donne le nombre de valeurs contenues sur une ligne de données. Les lignes suivantes contiennent les données.

Cette méthode doit lever une exception, avec un message d'erreur significatif lorsque :

- le nombre de lignes de données ne correspond pas au nombre de lignes de la matrice ;
- le nombre de valeurs contenues sur une ligne de données ne correspond pas au nombre de colonnes de la matrice ;

Question 4

Ajoutez à la classe `MatriceEntiere` une méthode `toString()` et une méthode d'affichage, et écrivez un programme permettant de construire une matrice à partir des données contenues dans un fichier et de l'afficher. Vous pourrez tester votre classe avec les fichiers de données fournis (`donnees_*`).

Question 5

Complétez la classe `MatriceEntiere` en lui ajoutant les méthodes permettant d'effectuer les opérations suivantes : initialisation à zéro, calcul de la transposée, somme de matrices, produit d'une matrice par un scalaire et produit de deux matrices.

Vous devrez vous assurer, lorsqu'une opération implique plusieurs matrices, que les dimensions de ces matrices sont compatibles avec l'opération demandée. Vous devrez pour cela créer votre propre classe `TaillesNonConcordantesException`.

Vous pourrez tester les méthodes en utilisant les fichiers de données fournis. Le fichier `resultats_tests` contient les valeurs attendues lorsqu'on somme les matrices initialisées à partir des valeurs contenues dans `donnees_somme1` et `donnees_somme2`, ainsi que le produit de la matrice initialisée à partir de `donnees_produit1` par la matrice initialisée à partir de `donnees_produit2`.

Exercice 5 – Gestion de comptes bancaires

Question 1

En considérant le programme suivant, nous voulons mettre en œuvre une exception afin d'interdire l'instanciation d'un objet Compte présentant un solde négatif. Proposez une implémentation.

```
public class Compte {
    private int numero = 0;
    private float solde;

    public Compte(float s){
        solde = s;
    }
}
```

Question 2

Doit-on inclure chaque instruction de création d'un compte dans un bloc **try ... catch...** ? Pourquoi ?

Question 3

Nous voulons implémenter des méthodes pour créditer et débiter le solde d'un compte. Certains comptes n'ont pas d'autorisation de découvert. Dans ce cas, quelle modification faudra-t-il apporter au programme pour lever une exception si le montant à débiter est plus élevé que le solde du compte ?

Complétez la classe `Compte` pour permettre la gestion de ces opérations et écrivez un programme de test.

Thème 2

Objectifs

- Création de thread
- Classes anonymes

Exercices TD

Exercice 6 – Création et lancement de threads : Hello World

Nous souhaitons écrire un programme très simple qui crée et lance l'exécution de plusieurs threads. La tâche affectée à chacun des threads est l'affichage de la chaîne de caractères *"Hello World"*.

Question 1
Quelle est la méthode qui contient le code exécuté par un thread ? Quelle est sa signature ?

Question 2
Proposez un programme qui lance trois threads affichant chacun *"Hello World"*, en passant par une sous-classe de `Thread`. Quel est l'inconvénient de cette solution ?

Question 3
Reprenez le programme précédent en utilisant cette fois une classe qui n'est pas une sous-classe de `Thread`.

Question 4
On veut mainteant individualiser les affichages. Chaque thread est identifié par un entier, et chacun salue une ville différente. On peut par exemple obtenir un affichage du type :

```
thread 2 salue Londres
thread 3 salue Tokyo
thread 1 salue Paris
```

Modifiez chacune des implémentations précédentes pour obtenir cet affichage.

Exercice 7 – Producteur/Consommateur : le producteur

Le schéma Producteur/Consommateur est un modèle classique de synchronisation. Un (ou plusieurs) processus producteur(s) dépose(nt) des données dans une zone de stockage, de taille limitée ou non. Un (ou plusieurs) processus consommateur(s) va (vont) récupérer les données dans cette même zone de stockage.

Nous nous intéressons dans un premier temps à un système très simple dans lequel un unique producteur dépose des données dans la zone de stockage. Cette zone de stockage est une instance d'une classe `Buffer` qui permet de gérer un tableau dans lequel les données déposées sont écrites. La donnée déposée est représentée par un entier naturel.

Question 1
Quelles sont les informations nécessaires pour pouvoir gérer la zone de stockage ? Proposez une implémentation de la classe `Buffer`.

Question 2
On s'intéresse au système le plus simple possible : un unique producteur effectue un nombre `nbDepots` de dépôts, où `nbDepots` correspond au nombre de cases dans la zone de stockage. La valeur déposée est un entier tiré aléatoirement.

Écrivez une classe `Producteur`, de manière à pouvoir lancer l'exécution du producteur dans un `thread` séparé.

Question 3
Écrivez un programme permettant de tester l'exécution du producteur. Ce programme crée un buffer de 5 cases, puis un producteur qui est exécuté dans un thread séparé. Chaque opération de dépôt donne lieu à un affichage.

Question 4
On souhaite aussi contrôler l'état du buffer, à la fin de l'exécution, pour vérifier que les dépôts ont bien été stockés.

Complétez si nécessaire votre classe `Buffer` en ajoutant une méthode `toString()` qui affiche l'état du stock. Appelez cette méthode à partir du thread principal. L'affichage de l'état du stock est-il correct ?

Question 5
Comment peut-on garantir que l'affichage n'a lieu qu'une fois que le producteur a effectué l'ensemble de ses dépôts ?

Exercices TME

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 8 – 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 `graphique` minimal basé sur Swing, qui permet d'afficher dans une fenêtre des points et des lignes. Les caractéristiques de la classe `Fenetre` du paquetage `graphique` sont données dans la Figure 2.
Le paquetage graphique est localisé dans le répertoire ci-dessous :

```
/Infos/lmd/2015/licence/ue
```

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/2015/licence/ue
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 utilisant une fonction `tracerLignePointAPoint`.

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.

Constructor Summary

Constructors

Constructor and Description
`Fenetre(int width, int height, java.lang.String title)` Cree une fenetre de dimensions width x height, avec un titre title et un fond blanc
`Fenetre(int width, int height, java.lang.String title, java.lang.String couleur)` Cree une fenetre de dimensions width x height, avec un titre title et une couleur de fond couleur

Method Summary

Methods

Modifier and Type	Method and Description
void	`remplir(java.lang.String couleur)` Remplit le fond de la fenetre avec la couleur couleur
void	`tracerLigne(java.awt.Point pt1, java.awt.Point pt2)` Trace une ligne noire entre les points pt1 et pt2
void	`tracerLigne(java.awt.Point pt1, java.awt.Point pt2, java.lang.String couleur)` Trace une ligne en couleur entre les points pt1 et pt2
void	`tracerLignePointAPoint(java.awt.Point pt1, java.awt.Point pt2)` Trace point par point une ligne noire du point pt1 au point pt2
void	`tracerLignePointAPoint(java.awt.Point pt1, java.awt.Point pt2, java.lang.String couleur)` Trace point par point une ligne en couleur du point pt1 au point pt2
void	`tracerPoint(java.awt.Point pt)` Affiche en noir le point de coordonnees (x, y)
void	`tracerPoint(java.awt.Point pt, java.lang.String couleur)` Affiche en couleur le point de coordonnees (x, y)

FIGURE 2 – La classe `Fenetre`

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`.

Question 4

Proposez une dernière réalisation du tracé du triangle dans laquelle le tracé d'un côté est exécuté par un objet qui implémente l'interface `Runnable`. Vous utiliserez une classe anonyme pour définir cet objet.

Exercice 9 – Produit matriciel parallèle

Nous souhaitons compléter notre classe `MatriceEntiere` de manière à pouvoir paralléliser le produit de matrices. Lors de cette opération, 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. Ces calculs sont indépendants, on peut donc 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 `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` (il s'agit donc d'une méthode

statique). 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 ligne `j` d'une matrice `m2`.

Question 3

Écrivez la fonction `main` d'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`.

Thème 3

Objectifs

- Méthodes synchronisées
- `notifyAll()`

Exercices TD

Exercice 10 – Producteur/Consommateur : cas de base

Nous reprenons le problème du producteur/consommateur et nous voulons maintenant programmer un système complet : la zone de stockage est accédée à la fois par un producteur qui dépose des données et par un consommateur qui les retire.

Question 1
Complétez la classe `Buffer` écrite dans le cas du producteur simple pour permettre des retraits. Vous ajouterez aussi un accesseur permettant de connaître le nombre d'éléments présents dans le stock.

Le producteur et le consommateur effectuent le même nombre d'opérations. Ce nombre est supérieur au nombre de cases du tampon.

Question 2
À quelle condition un producteur peut-il effectuer un dépôt ? À quelle condition un consommateur peut-il effectuer un retrait ?

On se propose de construire une solution dans laquelle, tant que la condition pour effectuer une opération n'est pas remplie, le processus qui la demande est "endormi" (appel à la méthode `sleep` de la classe `Thread`) avant de pouvoir retenter sa chance. Il sort de ces sommeils successifs lorsque la situation lui permet de réaliser son opération.

Question 3
Proposez une nouvelle version de la méthode `depot` dans laquelle, lorsque le producteur ne peut pas produire, il est mis en sommeil pendant 10 millisecondes avant de refaire une tentative.

Ajoutez aussi un constructeur à la classe producteur permettant de fixer, à la création du producteur, le nombre de dépôts qu'il va effectuer.

Question 4
Proposez une version du retrait "symétrique" de celle que vous venez d'écrire pour le dépôt. Ajoutez aussi un constructeur au consommateur permettant de fixer le nombre d'opérations qu'il exécute.

Question 5
Écrivez un programme permettant de tester votre système producteur/consommateur, dans lequel le producteur et le consommateur sont exécutés dans des threads distincts.

Ce programme fonctionne-t-il de manière satisfaisante ?

Question 6
Modifiez votre programme de test pour lancer en parallèle trois producteurs et trois consommateurs.
Ce programme fonctionne-t-il de manière satisfaisante ?

Question 7
De quel mécanisme dispose-t-on pour résoudre le problème mis en évidence à la question précédente ?
Proposez une implémentation garantissant que le contenu du buffer est cohérent. Pour une meilleure lisibilité, on ajoutera aux producteurs (resp. aux consommateurs) un identifiant qui sera transmis lors du dépôt (resp. retrait) dans le buffer.

Question 8
Quels sont les avantages et les inconvénients d'utiliser directement le moniteur de l'instance de `Buffer` pour réaliser l'exclusion mutuelle ?

On souhaite mettre en place un niveau de synchronisation plus fin pour permettre un meilleur parallélisme lors de l'accès à la ressource partagée. On décide donc de dissocier, dans les méthodes d'accès au buffer :
- les accès (test et modification) au nombre d'éléments du buffer, qui se font en exclusion mutuelle ;
- l'accès effectif à une case (lecture ou écriture), qui peut se faire en parallèle de l'accès à une autre case par un autre processus.

Question 9
L'exclusion mutuelle pour les accès au compteur est-elle suffisante pour garantir la cohérence des données ? Montrer qu'on peut avoir un producteur et un consommateur qui accèdent à la même case. Peut-on utiliser un mécanisme de synchronisation pour l'empêcher ?

Exercices TME

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 11 – Gestion des réservations d'une salle de spectacle

Nous souhaitons écrire un programme permettant de gérer les réservations d'une salle de spectacle de `nbRangs` × `nbPlacesParRang` places. Les places de la salle sont représentées par un tableau à deux dimensions de booléens : `placesLibres[i][j]` vaut **`true`** si la place d'indice `j` au rang `i` est libre, **`false`** si elle est occupée.

Les spectateurs arrivent par groupes pour faire leur réservation. Chaque groupe a un effectif `nb` qui lui est propre. Lorsqu'un groupe de `nb` personnes se présente pour réserver :
- s'il ne reste plus assez de places disponibles, la demande est refusée ;
- sinon, on cherche un ensemble de `nb` places libres sur un même rang où les spectateurs peuvent s'asseoir côte à côte ;
- si un tel ensemble n'existe pas, on répartit les `nb` spectateurs en remplissant au fur et à mesure les places encore disponibles dans la salle.

Le système de réservation est construit à partir d'une classe `Salle` et d'une classe `Groupe`. Chaque groupe a un identifiant unique et le nombre de personnes qui le composent est fixé à sa création. Nous considérons dans un premier temps que la seule action d'un groupe est de faire une demande de réservation. Lors de cette demande, le groupe transmet une référence sur lui-même.

Question 1
Écrivez le code de la classe `Groupe`. Cette classe doit-elle implémenter une interface particulière ? De quelle méthode a-t-on besoin dans la classe `Salle` ?

Nous nous intéressons maintenant à la classe `Salle`. Nous souhaitons décomposer la réservation en plusieurs opérations de manière à conserver des méthodes simples. Nous allons donc considérer les méthodes suivantes :
- `capaciteOK(int n)` : renvoie vrai s'il y a encore au moins `n` places libres dans la salle ;

- `nContiguesAuRangI (int n, int i)` : renvoie `-1` s'il n'y pas `n` places libres côte à côte au rang `i`. Sinon, renvoie la position `j` qui est la première du bloc de `n` places libres au rang `i` ;
- `reserverContigues (int id, int n)` : lorsque c'est possible, exécute la réservation de `n` places contiguës pour le groupe `id` et renvoie vrai. Renvoie faux sinon.
- `reserver (int id, int n)` : lorsque c'est possible, effectue une réservation de `n` places (contiguës ou non) pour le groupe `id` et renvoie vrai. Renvoie faux sinon.

Question 2
Parmi ces méthodes, quelles sont celles pour lesquelles il est nécessaire de garantir une exclusion mutuelle ? Comment peut-on réaliser cette exclusion mutuelle ?

Question 3
Écrivez le code de la classe `Salle`.

Question 4
Écrivez un programme de test dans lequel plusieurs groupes s'exécutant dans des threads différents demandent à effectuer des réservations.

Nous souhaitons maintenant permettre à un groupe d'annuler tout ou partie de ses réservations.

Question 5
Quelle information faut-il ajouter à la classe `Groupe` ? Proposez une structure de données pour gérer cette information et donnez la nouvelle implémentation de la classe. Donnez les modifications à apporter à la classe `Salle`.

Exercice 12 – Le pliage de la capote du Spider

La firme automobile UPM*Car* met au point son nouveau spider 3I001 (voir figure 3) pour lequel il faut concevoir un logiciel permettant de gérer une ouverture et une fermeture rapides de la capote. Nous nous intéressons ici à la phase d'ouverture (lorsqu'on replie la capote dans le coffre), et plus précisément à la synchronisation entre le mouvement de la capote et celui des vitres.

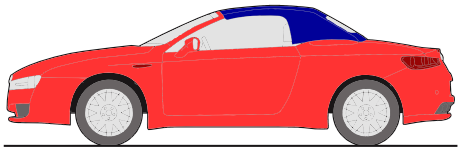

FIGURE 3 – Le nouveau spider 3I001 de chez UPM*Car*

La capote ne peut être rangée que lorsque les vitres sont ouvertes (en position basse). Pour obtenir de bonnes performances, les ingénieurs souhaitent paralléliser le mouvement des vitres. Il s'agit donc de réaliser une solution à trois threads : un thread principal qui pilote l'opération, et un thread pour le moteur de chaque vitre.

Afin d'obtenir un programme lisible, nous allons utiliser des types énumérés pour
- la position d'une vitre : `HAUTE`, `BASSE` ;
- l'opération demandée sur une vitre : `MONTER`, `DESCENDRE`, `NIL` s'il n'y a pas d'opération à effectuer ;
- le côté de la voiture concerné : `GAUCHE`, `DROITE` ;

(voir `http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html` pour la manipulation des types énumérés).

Question 1
Donnez le code Java pour les trois types énumérés.

Nous allons créer une classe `MoteurVitre` (avec un constructeur `MoteurVitre(Cote c)`) dont chaque instance pilotera une vitre. Le rôle du gestionnaire (pour l'instant) est de créer et démarrer les threads correspondant aux deux moteurs et de leur demander, si la vitre qu'ils pilotent est en position haute, de la faire descendre.

Question 2

Proposez une implémentation du gestionnaire de la capote. On ne se formalisera pas du fait que cette implémentation utilise des méthodes non encore écrites de la classe `MoteurVitre`.

Nous nous intéressons enfin à la programmation du moteur d'une vitre. Ce programme doit respecter les contraintes suivantes :

- il doit se tenir prêt à traiter une demande de mouvement en provenance du gestionnaire, mais ne rien faire tant que cette demande n'est pas arrivée ;
- l'opération demandée (qu'il s'agisse d'une montée ou d'une descente) prend un certain temps, que le programme simulera par un délai aléatoire. Cette opération doit être exécutée dans le thread du moteur ;
- le thread doit pouvoir potentiellement traiter une infinité de demandes (nous ne nous intéressons pas ici à la terminaison du système).

Question 3

Quelles sont les situations dans lesquelles l'exécution du thread peut se trouver bloquée ? Proposez une implémentation du blocage/déblocage.

Question 4

Complétez le programme de la classe `MoteurVitre`.

Thème 4

Objectifs

- Interface `Lock : lock(), unlock()`
- Interface `Condition : await(), signal()`

Exercices TD

Exercice 13 – Communication entre tâches, terminaison

Le but de cet exercice est de réfléchir sur l'une des principales difficultés de la programmation concurrente : qui fait quoi, et quand ? Il y a en général plusieurs réponses possibles à ces questions, nous proposons de réfléchir à la construction d'une solution, mais il existe des alternatives qui fonctionneront probablement très bien...

Nous allons appuyer notre réflexion sur une version simplifiée d'un problème de synchronisation classique : le problème du barbier.

Un salon de coiffure peut accueillir dans sa salle d'attente un nombre limité n de clients. Un client qui se présente à l'entrée du salon repart sans attendre si toute les chaises sont occupées. Sinon, il s'installe et attend que le coiffeur puisse s'occuper de lui. Le coiffeur a un comportement répétitif qui consiste, lorsqu'au moins un chaise est occupée, à faire entrer l'un des clients dans la pièce d'à côté pour le coiffer. La chaise devient disponible pour un nouveau client.

Question 1
Quelles sont les classes qui composent ce système ? Quelles sont celles qui devraient implémenter l'interface `Runnable` ?

Question 2
Quelles sont les variables qui définissent l'état de la salle d'attente ? Quelles sont les actions qui modifient ces variables ? Dans quelle classe doivent-elles être implémentées sous forme de méthode ? Doivent-elles être manipulées dans une section critique ?

Question 3
Quelles sont les conditions qui peuvent bloquer temporairement l'exécution d'un client ? Comment peut-on les implémenter ?

Question 4
Quelles sont les conditions qui peuvent bloquer temporairement l'exécution du coiffeur ? Comment peut-on les implémenter ?

Question 5
Comment peut-on gérer la terminaison du coiffeur ?

Exercice 14 – Producteur/Consommateur : cas optimisé

Nous reprenons le problème du producteur/consommateur et nous voulons maintenant programmer la solution optimisée qui améliore le parallélisme. Nous rappelons son principe :

- les accès (test et modification) au nombre d'éléments du buffer se font en exclusion mutuelle ;

– l'accès effectif à une case (lecture ou écriture) peut se faire en parallèle de l'accès à une autre case par un autre processus.

Question 1
Rappelez quels sont les mécanismes de protection nécessaires pour garantir la cohérence de l'accès aux données.

Question 2
On a vu qu'à cause de l'entrelacement des synchronisations, on ne pouvait pas utiliser deux blocs synchronisés pour protéger le compteur d'une part, la case d'autre part.
Peut-on protéger l'accès au compteur partagé par un bloc d'instructions synchronisées et utiliser un verrou pour protéger l'accès à une case ?

Question 3
Comment faire pour gérer séparément le blocage des producteurs et celui des consommateurs ? Proposez une implémentation de cette solution.

Question 4
Comment pourrait-on envisager d'améliorer les performances ?

Exercices TME

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 15 – Implémentation du problème du barbier

Question 1
Proposez une implémentation multi-threads du problème du barbier.

Exercice 16 – La capote du Spider, le retour

Nous reprenons l'exercice du pliage de la capote du spider, et nous voulons maintenant exécuter l'opération complète : une fois les vitres baissées, le gestionnaire peut activer le pliage de la capote, que nous implémenterons comme un simple affichage. Une fois la capote repliée, le gestionnaire s'assure que chaque moteur remet la vitre qu'il pilote dans sa position initiale.

Question 1
Comment le gestionnaire peut-il savoir que les vitres sont en position basse ?

Question 2
Proposez une nouvelle implémentation de la classe `MoteurVitre` et de la classe `GestionnaireCapote`. Le gestionnaire devra arrêter les moteurs lorsque l'exécution complète du pliage de la capote est terminée.

Question 3
En supposant que l'opération de pliage de la capote prenne un certain temps, y aurait-il un intérêt à l'implémenter dans un thread différent de celui du gestionnaire ?

Thème 5

Objectifs

– Synchronisation de threads

Exercices TD

Exercice 17 – Synchronisation de threads : le crible d'Eratosthène

Le crible d'Eratosthène est un algorithme de recherche des nombres premiers bâti sur une propriété simple : si un entier p n'a aucun diviseur premier, alors p est premier.

Nous souhaitons réaliser une application qui permette d'effectuer une recherche parallèle des nombres premiers. Cette application est basée sur une chaîne de threads. Chaque thread se voit attribuer à sa création un identifiant id.

Lorsque le thread `id` reçoit une valeur `p` à traiter :

– si `id` divise `p`, le thread `id` ne fait rien ;
– si `id` ne divise pas `p`, et si le thread `id` n'a pas de successeur, il se crée un successeur d'identifiant `p` ;
– si `id` ne divise pas `p`, et si le thread `id` a un successeur, il transmet la valeur de `p` à son successeur.

Le chaînage démarre avec la création d'un thread d'identifiant 2.

Question 1

On suppose que chaque thread (hormis le thread de la classe de test) exécute la méthode `run` d'une instance d'une classe `Erato`. Cette classe dispose d'un constructeur permettant de transmettre à l'instance la valeur de `id`. Explicitez le déroulement de l'application lorsque le thread qui exécute `Erato(2)` reçoit successivement toutes les valeurs entre 2 et 8.

Question 2

Quelle solution proposez-vous pour s'assurer que chaque valeur envoyée est effectivement traitée ?

Question 3

Nous nous plaçons dans la situation où chaque thread traite au fur et à mesure les valeurs qu'il reçoit. Dans quel cas un thread peut-il se retrouver (temporairement) bloqué ? Qu'en est-il du thread principal ?

Question 4

Quels sont les variables et objets nécessaires à la réalisation de l'attente ?

Question 5

Donnez les variables d'instance et le constructeur de la classe `Erato`.

Question 6

Écrivez une méthode permettant à un thread exécutant une instance de la classe `Erato` d'attendre qu'on lui envoie une valeur à traiter.

Question 7

Le chaînage des threads est unidirectionnel : un thread possède une référence vers son successeur uniquement. La transmission d'une valeur se fait donc par un appel à une méthode du successeur.

Écrivez une méthode `recevoir` qui permet à un thread de récupérer une valeur transmise par son prédécesseur.

Question 8
Écrivez une méthode `transmettre` qui effectue le traitement d'une valeur reçue (rien, création d'un thread successeur ou transmission au thread successeur suivant les cas).

Question 9
Quelles sont les actions effectuées dans la méthode `run` par un thread exécutant une instance d'`Erato` ?

Question 10
À quel moment un thread se termine-t-il ?

Question 11
Proposez une solution pour gérer la terminaison.

Question 12
Écrivez un programme permettant de tester la classe `Erato`. On supposera que l'instance d'identifiant `2` reçoit successivement les valeurs de `2` à `100`.

Exercices TME

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 18 – Gestion d'un convoyeur

Nous nous intéressons à un système de convoyage dans lequel un chariot est utilisé pour transporter des objets d'un point à un autre. Ce chariot a une capacité limitée en poids (`poidsMax`) et en nombre (`nbMax`) de marchandises transportées. Les marchandises à transporter font partie d'un stock de taille limitée. Elles ne sont pas identiques : leurs poids sont différents,

Le chariot est alimenté par un chargeur dont le rôle est d'extraire les objets du stock et des empiler sur le chariot tant que celui-ci n'est pas plein (on n'a atteint ni le poids maximum, ni le nombre d'objets maximum). Lorsqu'un chargement est plein, le chargeur doit attendre que le chariot soit à nouveau prêt à recevoir des marchandises (ce qui suppose qu'il a été déchargé, nous ne nous intéressons pas à cette opération pour l'instant).

Ces opérations sont répétées jusqu'à ce que le stock soit vide.

Question 1
Définissez une classe `AleaObjet` permettant de construire des marchandises à charger dont le poids est tiré aléatoirement entre deux bornes `min` et `max`.

Définissez une classe `AleaStock` permettant de manipuler un stock de `taille` éléments de type `AleaObjet`. On veut pouvoir remplir le stock, tester s'il est vide ou en extraire un élément.

Question 2
Nous nous intéressons à la la classe `Chargeur`. Écrivez un constructeur, et la boucle d'opérations du chargeur. Déduisez-en un ensemble d'opérations à implémenter dans la classe `Chariot`.

Question 3
Écrivez la classe `Chariot`. On utilisera un objet de type `ArrayList` pour stocker les éléments présents dans le chariot. Que peut-on utiliser pour réaliser l'opération d'attente ?

Question 4
Construisez une classe permettant de tester l'exécution correcte d'un chargement : le chargeur remplit le chariot et se met en attente tant que le chariot n'a pas été vidé (donc le programme ne se termine pas).

Nous souhaitons maintenant compléter notre système en regardant les opérations de déchargement : une fois que le chariot a été rempli, le déchargeur récupère toutes les marchandises qu'il contient. Il signale alors que le chariot est vide et prêt pour un nouveau chargement. Il doit répéter cette opération pour chaque chargement effectué (on ne gère pas pour l'instant la terminaison).

Question 5
Écrivez la classe `Dechargeur` et déduisez-en les opérations à ajouter à la classe `Chariot`.

Question 6
Complétez la classe `Chariot` et le programme de test.

Question 7
Proposez une solution pour gérer la terminaison du déchargeur et implémentez-la.